Universidad de Alcalá
Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

Detección del entorno 360º de un vehículo autónomo mediante LIDAR aplicando técnicas Deep Learning

Autor: Javier del Egido Sierra

Tutor: Rafael Barea Navarro

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

"Detección del entorno 360º de un vehículo autónomo mediante LIDAR aplicando técnicas Deep Learning"

Autor: Javier del Egido Sierra

Tutor/es: Rafael Barea Navarro

TRIBUNAL:

Presidente: Daniel Pizarro Pérez

Vocal 1º: Sancho Salcedo Sanz

Vocal 2º: Rafael Barea Navarro

Calificación: .....

Fecha: 20-10-2020

“Si le hubiera preguntado a la gente qué querían,
habrían dicho caballos más rápidos”

Henry Ford

Agradecimientos

La entrega de este Trabajo Fin de Máster supone la finalización de seis años de intenso aprendizaje, tanto profesional como personal, llenos de emociones.

Me gustaría agradecer en primer lugar a todos los integrantes del grupo de investigación Robesafe, en el que he tenido el placer de colaborar durante los últimos dos años. Con ellos he aprendido que la imaginación es el único límite, y que el compañerismo es la mejor herramienta de trabajo.

Me faltan palabras para demostrar el agradecimiento que tengo hacia mis amigos que, pese a la distancia impuesta por la presente pandemia, he sentido más cerca que nunca, demostrando su apoyo incondicional.

A mi novia Rocío, el tesoro que se escondía en mis estudios. Gracias por ser mi apoyo y amuleto para caminar juntos por la vida.

A mi familia, por ayudarme a alcanzar mis propósitos y enseñarme a ser la persona que soy.

Índice

Índice de ilustraciones

Índice de tablas

Resumen

Este trabajo analiza el estado del arte en detección y seguimiento de múltiples objetos dinámicos (DAMOT) en el entorno de un vehículo autónomo mediante técnicas basadas en *deep learning*, evaluando el rendimiento de diferentes metodologías y estableciendo un pipeline funcional con la solución óptima.

Los diferentes objetos que se encuentran en la escena son detectados a partir de una nube de puntos proporcionada por un sensor LIDAR, obteniendo una alta tasa de acierto con elevada precisión de localización 3D. Para ello, se evalúan diferentes redes neuronales sobre la base de datos KITTI [1] con el objetivo de obtener datos cuantitativos de su desempeño.

Posteriormente, se estudian diferentes procedimientos de seguimiento de los objetos detectados basados en técnicas tradicionales y *deep learning*, siendo capaz de identificar unívocamente cada objeto con el propósito de seguir su trayectoria. Al igual que en la etapa de detección, se extraen datos cuantitativos a través del análisis en la base de datos KITTI y la herramienta de evaluación 3D MOT desarrollada por AB3DMOT [2].

Una vez identificadas las técnicas de detección y seguimiento óptimas se aplicarán sobre diversas escenas urbanas obtenidas mediante el simulador de conducción CARLA [3] y la base de datos KITTI, analizando cualitativamente el funcionamiento sobre sensores de conducción autónoma tanto simulados como reales.

Finalmente, se implementará el pipeline óptimo en un sistema embebido Nvidia Jetson AGX Xavier sobre el vehículo eléctrico autónomo en desarrollo para el proyecto Techs4AgeCar, obteniendo un análisis cuantitativo del sistema en tiempo real.

Conceptos clave

Deep Learning, conducción autónoma, KITTI, LIDAR, DAMOT, CARLA

Abstract

This work studies the state-of-the-art detection and multi-object tracking (DAMOT) techniques into autonomous vehicles surroundings based on deep learning, analyzing the performance achieved by different methods and developing a functional pipeline with the optimal configuration.

Dynamic objects in scene are detected from a tridimensional LIDAR point cloud, obtaining a high success rate along with a precise 3D localization. To do so, different neural networks are evaluated on KITTI dataset with the aim of setting a comparison based on quantitative results.

On a second stage, several multi-object tracking approaches based on both traditional techniques and deep learning are applied, being able to uniquely identify each object in scene and follow its trajectory. Similarly to detection phase, quantitative results are obtained from KITTI dataset and 3D MOT evaluation tool provided by AB3DMOT.

When optimal detection and tracking techniques are identified they will be applied on several urban scenes obtained in CARLA driving simulator and KITTI tracking subset, analyzing the qualitative performance over simulated and real autonomous driving sensors.

Finally, an optimal DAMOT pipeline will be implemented on an embedded system on developing autonomous electric vehicle for Techs4AgeCar project, performing a quantitative analysis on a real-time application.

Key concepts

Deep Learning, autonomous driving, KITTI, LIDAR, DAMOT, CARLA

Resumen extendido

Desde la invención del automóvil a mediados del siglo XIX y con su progresiva popularización desde inicios del siglo XX, la complejidad en el diseño de estos ha crecido exponencialmente, estableciendo requisitos cada vez más restrictivos en cuanto a la seguridad de sus ocupantes frente a accidentes de tráfico y actualmente introduciendo nuevas tecnologías de evitación de colisiones para proteger a las personas externas al vehículo.

Ayudándose de la vanguardia tecnológica, la industria de la automoción integra actualmente novedosos avances de asistencia al conductor para evitar riesgos como atropello de peatones o salidas involuntarias de carril a partir de análisis de imagen. Sin embargo, el futuro de los vehículos pasa por la conducción autónoma, una apuesta que genera grandes expectativas en cuanto a la disminución de accidentes hasta tasas marginales debido a la alta capacidad de procesamiento del entorno basado en sensores. La alta variedad de sensores disponibles en el mundo de la conducción autónoma permite cubrir situaciones complejas para un conductor humano, además de mejorar la precisión y permitir observaciones en todo el entorno del vehículo y no solo en la parte frontal. Las cámaras RGB pueden extraer características visuales de los objetos en situaciones de luminosidad suficiente, mientras que sensores infrarrojos, ultrasonidos, LIDAR y radares pueden funcionar también con poca iluminación.

Con el objetivo de detectar y seguir objetos alrededor de un vehículo autónomo, este trabajo analiza el estado del arte de redes neuronales detectoras y algoritmos de seguimiento de objetos, buscando una configuración óptima que permita la ejecución en tiempo real sobre el vehículo Techs4AgeCar en un sistema embebido.

La etapa de detección estará basada en una red neuronal detectora de objetos a partir de una nube de puntos tridimensional proporcionada por un sensor LIDAR ubicado en la parte superior del vehículo, obteniendo una representación precisa del entorno 360º del vehículo. Con el objetivo de escoger el método más eficiente para el propósito de este trabajo se evalúan las diferentes redes sobre la base de datos KITTI, que proporciona nubes de puntos y *ground-truth* (datos reales de los objetos presentes) de escenas tomadas en entornos urbanos de la ciudad de Karlsruhe, Alemania, conociendo la capacidad de detección de los diferentes métodos, así como de localización de los objetos en la escena.

Una vez escogida la herramienta de detección más efectiva se analizarán diferentes estrategias de seguimiento de múltiples objetos empleando técnicas tradicionales y deep learning. Los algoritmos tradicionales de seguimiento de objetos emplean técnicas algebraicas altamente eficientes, computando por un lado predicciones de la posición futura de los objetos en el siguiente instante, para después comparar dichas localizaciones con las de los objetos realmente detectados en escena y establecer una relación, vinculando los elementos detectados entre un instante y el instante anterior. Con la evolución de las técnicas de aprendizaje de redes neuronales se han desarrollado nuevas estrategias de análisis de la escena para establecer mejores relaciones entre instantes de detección, como puede ser el seguimiento de objetos a través de imágenes RGB, pudiendo extraer características visuales que faciliten la diferenciación entre los distintos elementos en escena. Al igual que durante la etapa de detección, se establecerá un análisis cuantitativo a partir de la base de datos KITTI y la herramienta 3D MOT proporcionada por AB3DMOT, conociendo la eficiencia de los distintos algoritmos estudiados a la hora de identificar objetos unívocamente a través del desarrollo de una escena compuesta por un conjunto de datos tomados consecutivamente.

Con las etapas de detección y seguimiento de objetos óptimas se realizará una primera aplicación en el simulador de conducción hiperrealista CARLA. Este simulador incorpora los sensores más comunes en conducción autónoma, como LIDAR, cámaras, radar, IMU, obteniendo un alto realismo de la escena simulada evitando los elevados costes de dichos sensores. El sistema se aplicará sobre diversas escenas urbanas con el objetivo de poder realizar un primer análisis cualitativo del desempeño en una escena urbana compleja.

Además, previo a la aplicación sobre el vehículo autónomo, el sistema será también analizado cualitativamente sobre una escena de la base de datos de KITTI, con el objetivo de estudiar visualmente el funcionamiento sobre sensores LIDAR reales.

Finalmente se establecerá un pipeline óptimo de detección y seguimiento de objetos dinámicos para ser implementado en un sistema embebido de forma que opere con los sensores integrados en el vehículo Techs4AgeCar. El sistema será analizado cuantitativamente a través de diferentes escenas empleando entornos y sensores reales.

Capítulo 1.
Introducción.

El ser humano requiere indispensablemente de movilidad, entendida como la capacidad de transportarse, con una doble función: como medio de transporte para llevar a cabo tareas productivas, obtención de alimentos y establecimiento de relaciones sociales, o con el propio fin de practicar la capacidad de caminar [4]. Para la primera de las funciones, la humanidad ha sido capaz de desarrollar inventos que facilitan el desplazamiento a lugares más lejanos, como los transportes individuales o los colectivos.

La evolución social, lejos de disminuir el problema de la necesidad de transporte, ha incrementado la necesidad de realizar viajes cada vez más largos. El modelo urbanístico diseñado a partir del siglo XX establece un sistema dependiente del automóvil para todas las tareas básicas, produciendo una proliferación de estos, generando una mayor complejidad en el diseño de infraestructuras con el objetivo de evitar retenciones en los medios de transporte.

Gracias a la evolución en los sistemas de seguridad activa y pasiva implementados en los vehículos, la evolución del número de víctimas de accidentes de tráfico por mil habitantes en España se mantiene estable en valores bajos pese a que el número de vehículos por mil habitantes se haya duplicado, como se muestra en Ilustración 1. Por lo tanto, el número de víctimas de accidentes por vehículo del parque de automóviles ha disminuido más del 50%.

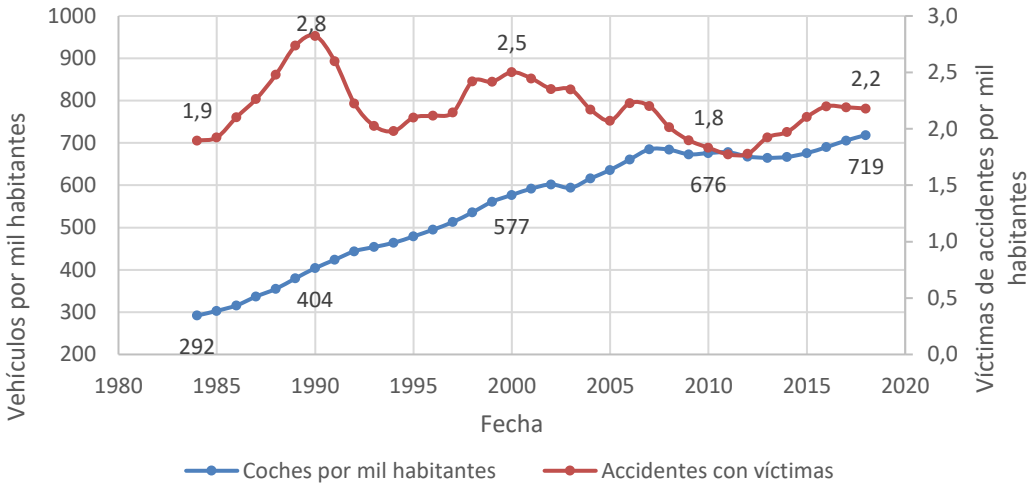


Ilustración 1 Análisis del tráfico en España.

El análisis de la disminución de víctimas de accidentes frente al crecimiento del número de vehículos en circulación evidencia la importancia en la inversión en tecnologías que permitan incrementar la seguridad en los vehículos autopropulsados. Actualmente, la industria automovilística presenta una gran innovación en cuanto a técnicas de evitación de víctimas de accidentes, como los ya implantados sistemas de frenada de emergencia frente a peatones o vehículos que obstaculizan el paso o seguimiento de carriles en carretera. Siguiendo este camino, los vehículos ganan cada vez más inteligencia, implementando primero sistemas de asistencia a la conducción (ADAS) para evolucionar en sistemas de conducción autónoma en los que el conductor interviene solo en ocasiones en las que el vehículo no es capaz de reaccionar.

Con el objetivo de cuantificar el nivel de autonomía de un vehículo, la Sociedad de Ingenieros Automotrices (SAE) define el baremo de acuerdo con el estándar J3016 [5], estableciendo seis niveles en función del nivel de intervención del conductor. En los tres primeros niveles el conductor es el encargado de controlar el vehículo y supervisar la acción de los diferentes ADAS que pueden integrarse, mientras que en los tres niveles superiores el vehículo toma el control de la conducción, siendo capaz de realizar maniobras de conducción complejas y solo pudiendo retornar el control al humano en el nivel más bajo. Se puede encontrar una descripción más detallada en la Ilustración 2, donde se indican los sistemas que pueden incluir los vehículos, así como las tareas que debe realizar el conductor, en cada nivel de autonomía.

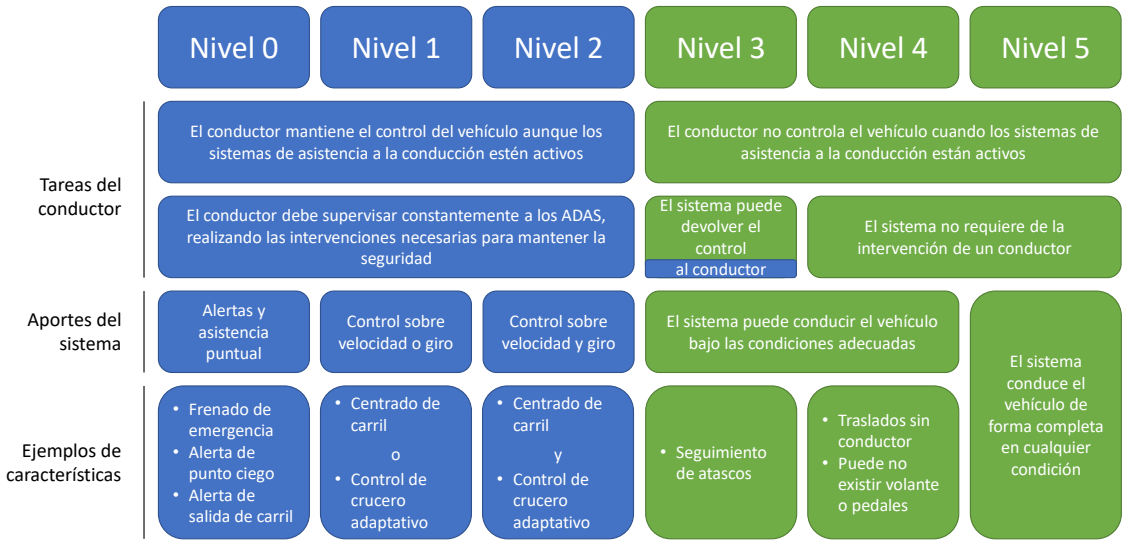


Ilustración 2 Niveles de conducción autónoma

Hasta la fecha, las tecnologías desarrolladas han permitido comercializar vehículos hasta un nivel de autonomía 3, en los que el conductor puede ser requerido para tomar el control en ciertas situaciones en las que el vehículo no distingue con claridad la acción a realizar. Estos vehículos otorgan un alto nivel de comodidad al conductor ya que pueden centrarse automáticamente en el carril cuando éste se encuentra correctamente definido, así como seguir al vehículo precedente tanto en movimiento como en atascos.

El desarrollo de vehículos de nivel 4 requiere una alta capacidad de entendimiento del entorno, debido a que pueden circular sin la necesidad de un conductor. Para ello, el automóvil debe ser capaz de conocer su posición y su ruta a seguir para aplicar un control autónomo, y detectar todos los obstáculos que pueden interferir en el camino para aplicar acciones sobre el control. Un fallo en uno de los sistemas puede provocar un grave accidente, por lo que los niveles de redundancia y fiabilidad de estos deben ser elevados.

Mientras que los algoritmos de control para seguimientos de trayectorias sobre robots autónomos llevan décadas de desarrollo y mejora, las técnicas de detección de obstáculos han experimentado un importante auge en la última década gracias al crecimiento en la capacidad computacional, siendo un campo de experimentación de nuevas técnicas que deben ser capaces de lograr precisión en tiempo real procesando una elevada cantidad de datos procedentes de diversos sensores.

El objetivo de este Trabajo Fin de Máster es la elaboración de un procedimiento óptimo de detección y seguimiento de objetos en el entorno de un vehículo autónomo en tiempo real evaluando diferentes técnicas de aprendizaje profundo, estableciendo análisis cuantitativos y cualitativos del método establecido sobre entornos urbanos simulados y reales.

Para ello se analizará el estado del arte en los campos de detección y seguimiento de objetos, estudiando las diferentes propuestas y la idoneidad de su aplicación en un vehículo para el procesamiento en tiempo real y obteniendo información cuantitativa de su funcionamiento sobre el conjunto de datos KITTI [1] empleando sus herramientas asociadas, así como la herramienta 3D MOT desarrollada por AB3DMOT [2]. Una vez definido el sistema se analizará cualitativamente sobre escenas urbanas en el simulador de conducción autónoma CARLA [3] y la base de datos KITTI, obteniendo una primera observación de la funcionalidad del conjunto. El objetivo final es la incorporación del sistema de detección y seguimiento de objetos sobre el hardware embebido Nvidia Jetson AGX Xavier en el vehículo autónomo Techs4AgeCar, evaluando cualitativamente el funcionamiento en diferentes situaciones.

Para la consecución de estos objetivos se realizará un TFM siguiendo los distintos capítulos que se detallan a continuación.

Capítulo 1. Introducción.
Capítulo 2. Estado del arte.
Capítulo 3. Herramientas aplicadas.
Capítulo 4. Desarrollo e implementación de sistema de detección y seguimiento.
Capítulo 5. Análisis del sistema.
Capítulo 6. Conclusiones generales y trabajos futuros.

Capítulo 2.
Estado del arte.

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

En este capítulo se estudiará el estado del arte en los campos de detección y seguimiento de los objetos en el entorno de vehículos autónomos, procesando información obtenida mediante diversos tipos de sensores con el objetivo de localizar e identificar a los vehículos y peatones circundantes mientras estos permanezcan en el campo de visión.

2.1 Detección de objetos dinámicos

La correcta detección de los elementos que pueden aparecer en la trayectoria del vehículo es crucial para obtener un elevado nivel de seguridad y evitar accidentes. Tradicionalmente se han aplicado técnicas de análisis de imagen, y más tarde de

nubes de puntos LIDAR, mediante complejos algoritmos con el objetivo de encontrar formas similares a las esperadas por los objetos como peatones o vehículos.
Con las mejoras acontecidas en el campo del procesamiento gráfico, los tiempos de análisis y la precisión de las técnicas basadas en *deep learning* han permitido superar ampliamente los resultados obtenidos mediante técnicas tradicionales, pudiendo encontrar un mayor número de objetos en situaciones más complejas mediante el aprendizaje por parte de redes neuronales profundas.

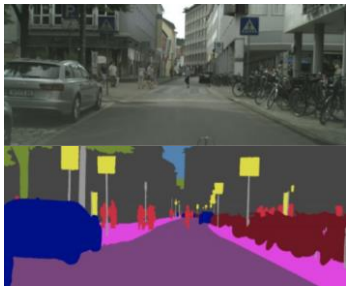
Actualmente la detección de objetos dinámicos se realiza principalmente mediante el procesamiento de la información obtenida mediante dos tipos de sensores: cámaras RGB o LIDAR.

2.1.1 Procesamiento de cámara RGB.

Mediante el procesamiento de imágenes a color aplicando redes neuronales se puede conseguir el reconocimiento de objetos de diverso tipo en una escena. En el estado del arte se encuentran diversas redes que permiten su localización tanto en el plano imagen [6] como en el mundo tridimensional, aplicando complejas proyecciones [7] [8], tanto enmarcándolos en un cuadro delimitador, como identificándolos a nivel de pixel [9]. Sin embargo, un pequeño error en la posición del objeto en píxeles, así como en las matrices de calibración empleadas, puede conducir a una gran diferencia en la localización del objeto cuando este se encuentra a una gran distancia.

a) Detección en cuadro delimitador 2D [6]

b) Detección en cuadro delimitador 3D [7]



c) Detección a nivel de píxel [9]

Ilustración 3 Detección de objetos en imagen.

El estudio de la escena a partir de la información de cámaras RGB presenta dificultades como posibles oclusiones de objetos al analizar un plano subjetivo desde el vehículo, condiciones de luz específicas que permitan una visión clara, o el cálculo de matrices proyectivas y de calibración de alta precisión para evitar errores.

2.1.2 Procesamiento de sensor LIDAR.

Los sensores LIDAR (*Light Detection and Ranging*) permiten obtener información tridimensional precisa, evitando problemas de proyección como los sucedidos al analizar imágenes 2D. Estos sensores retornan una nube de puntos cubriendo todo el entorno del vehículo, permitiendo analizar la escena al completo con alta precisión empleando un único sensor.

La información obtenida puede analizarse siguiendo diferentes estrategias, como puede ser un agrupamiento (*clustering*) de puntos que se consideran pertenecientes a un mismo objeto [10], reduciendo el análisis a una vista de pájaro bidimensional que permita estudiar la escena sin oclusiones [11], dividiendo la escena en voxels tridimensionales para aplicar redes convolucionales bidimensionales [12] o tridimensionales [13], o procesando la totalidad de la nube de puntos [14].

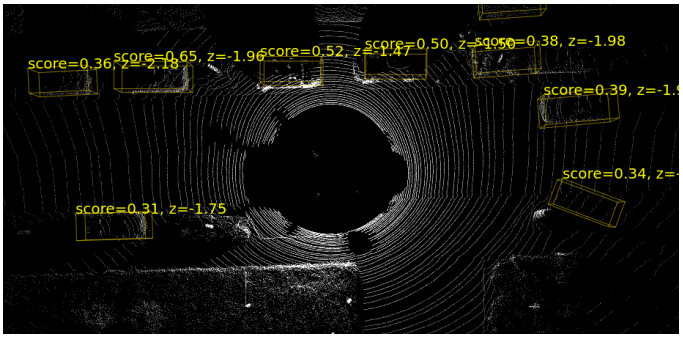


Ilustración 4 Vista de pájaro de detección sobre LIDAR en escena KITTI.

2.2 Seguimiento de objetos dinámicos

Una vez que el vehículo conoce la localización de los distintos objetos en su entorno, así como el tipo de objeto al que pertenece (peatón, vehículo, ciclista, etc.) es importante poder asignarles un identificador para conocer cuál es la trayectoria de cada uno de los objetos con el progreso del tiempo, de forma que el sistema puede llegar a predecir dónde se encontrarán y reaccionar de forma anticipada.

Al igual que los métodos de detección, las técnicas de seguimiento se han realizado hasta la fecha aplicando algoritmos con complejos cálculos. Actualmente se están desarrollando técnicas de aprendizaje profundo para poder identificar a los objetos y predecir sus posiciones futuras con mejor resultado.

2.2.1 Seguimiento de objetos en plano bidimensional

Algunas técnicas emplean el seguimiento de objetos en planos bidimensionales, ya sean subjetivos desde el vehículo como el plano imagen, o la vista de pájaro, que permite evitar oclusiones de los objetos. Los métodos de seguimiento tradicionales [15] emplean filtros de Kalman para predecir la posición de los objetos detectados en el siguiente instante, así como el algoritmo Húngaro [16] para asociar dichas predicciones con las nuevas detecciones, manteniendo el identificador del objeto mientras este sea percibido. Otras técnicas se valen de información visual para extraer características de los objetos detectados en la escena y facilitar la asociación [17], mostrando su resultado en la Ilustración 5.

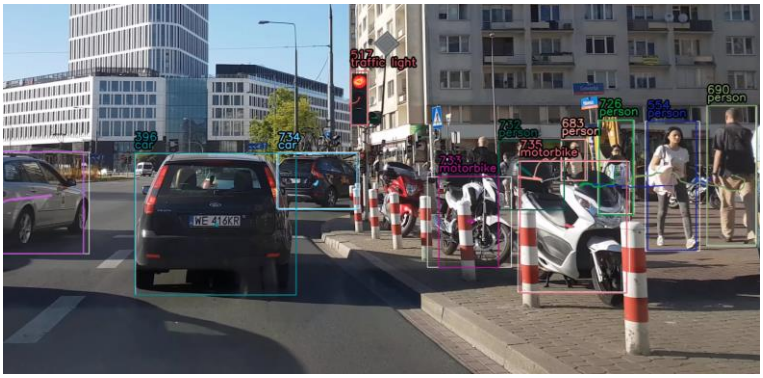


Ilustración 5 Seguimiento de objetos en plano imagen.

2.2.2 Seguimiento de objetos en el espacio tridimensional

Con el objetivo de conocer todas las variables como velocidad y aceleración de los objetos en su magnitud real evitando distorsiones, el seguimiento de objetos puede realizarse en el espacio tridimensional. Para ello la forma más sencilla y eficiente es la extensión del filtro de Kalman bidimensional a tres dimensiones [2], mostrando su resultado aplicando diferente identificador a cada objeto en la Ilustración 6. Otros trabajos mantienen el filtro 2D en vista de pájaro, recuperando la altura mediante otras técnicas [18].

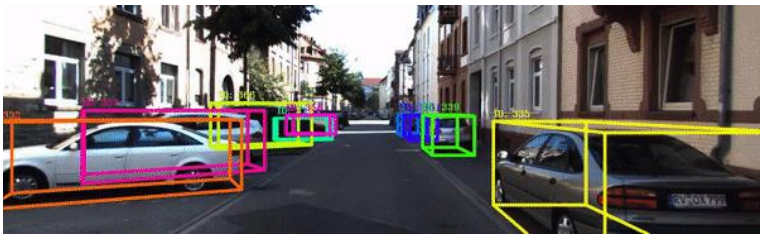


Ilustración 6 Seguimiento de objetos en plano tridimensional. Proyección a imagen para visualización.

Como se ha introducido anteriormente, en la actualidad los sistemas complejos anteriormente implementados mediante algoritmos se están sustituyendo por sofisticadas técnicas de aprendizaje profundo, obteniendo un mejor resultado gracias a las mejoras introducidas en las unidades de procesamiento gráfico.

A continuación, se desarrollarán los métodos más innovadores en cuanto a detección y seguimiento de objetos mediante técnicas tradicionales y/o *deep learning*.

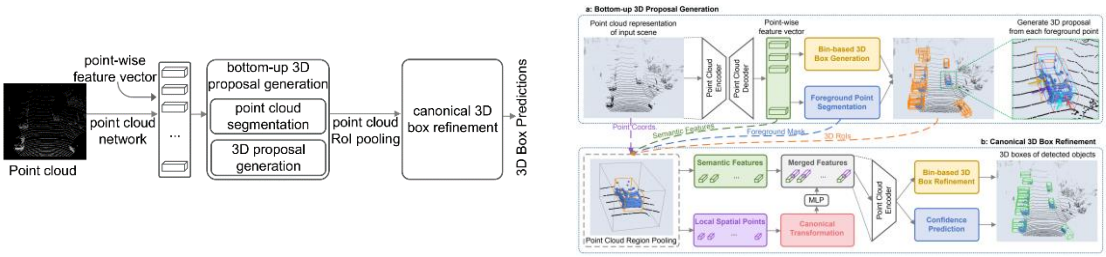
2.3 Detección de objetos mediante aprendizaje profundo.

En la segunda mitad de la década de 2010 se han sucedido diversas redes neuronales que analizan nubes de puntos para identificar objetos urbanos como vehículos, peatones y ciclistas, incrementando progresivamente el nivel de precisión en sus detecciones introduciendo sucesivas mejoras.

Inicialmente PointRCNN [14] estudia el conjunto de la nube de puntos tridimensional sin necesidad de realizar un procesamiento previo. SECOND [19] y PointPillars [12] discretizan la información, realizando procesamiento tridimensional en el primer caso o en formato bidimensional en vista de pájaro aprovechando la eficiencia de las convoluciones bidimensionales en el segundo. Como evolución a PointRCNN, Part-A^2 [20] estudia la pertenencia de cada punto de la nube a un objeto, y posteriormente los agrega para formar el objeto tridimensional. Finalmente, PV-RCNN [21] combina ambos conceptos, integrando técnicas de análisis de nube de puntos discretizada en vóxeles tridimensionales para extracción de características y nube de puntos completa con el objetivo de mantener la precisión de la información.

2.3.1 PointRCNN

A partir del desarrollo de PointNet [22], que analizaba la nube de puntos a partir de detecciones realizadas sobre imagen bidimensional y con el objetivo de mejorar el resultado, Shaoshuai Shi et al. desarrollaron PointRCNN [14], siendo la primera red neuronal en analizar únicamente una nube de puntos sin procesar para obtener detecciones tridimensionales precisas.



a) Descripción general de arquitectura de PointRCNN *b) Especificación de la aportación*

Ilustración 7 Arquitectura de PointRCNN

PointRCNN hace uso de una división en dos etapas para obtener un refinamiento de las predicciones realizadas, como se muestra en la Ilustración 7.

La primera fase, denominada *Bottom-up 3D Proposal Generation* estudia la pertenencia de cada punto de la nube de puntos a un objeto, aumentando la precisión respecto a otros métodos que emplean directamente cuadros delimitadores tridimensionales [23]. Esta etapa produce propuestas de cuadros delimitadores 3D para cada punto analizado.

La segunda etapa, *Canonical 3D box refinement*, realiza una ampliación de la información, computando características añadidas a la localización de los puntos como la distancia al sensor para compensar la disminución de resolución con la lejanía. Finalmente, PointRCNN produce una lista de características que definen al objeto detectado: localización tridimensional, dimensiones, orientación, tipo de objeto y nivel de confianza.

2.3.2 SECOND

Como yuxtaposición a la propuesta de PointRCNN, SECOND [19] emplea una discretización de la nube de puntos en forma de vóxeles tridimensionales con el objetivo de conseguir una inferencia precisa en un tiempo más reducido.

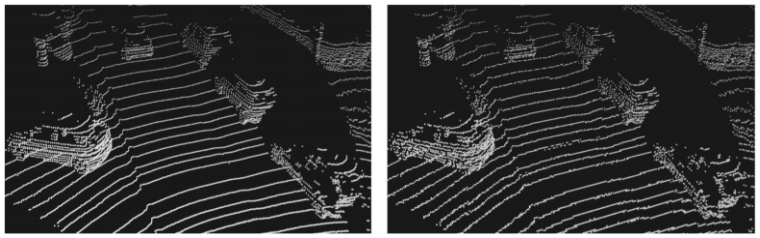


Ilustración 8 Comparativa de nube de puntos sin procesamiento y voxelizada

Una correcta discretización de la nube de puntos permite una disminución de la carga de procesamiento a la vez que mantiene la precisión de la información original, como se muestra en la Ilustración 8, donde la diferencia es imperceptible.

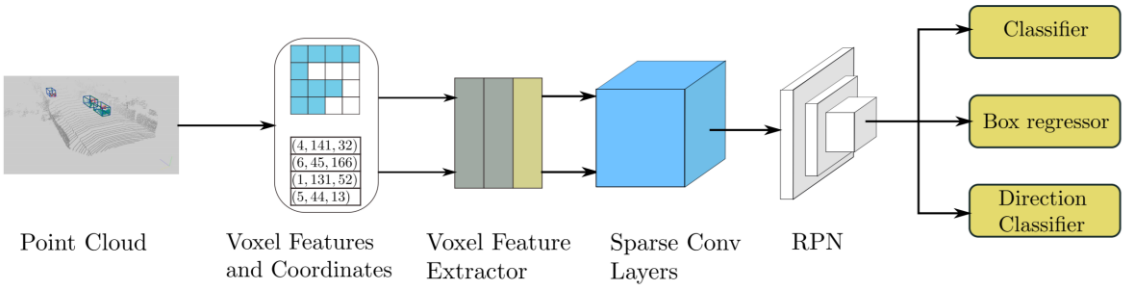


Ilustración 9 Arquitectura de red de SECOND

La red consta de cuatro fases consecutivas. La primera de ellas sigue la estructura de trabajo de [24], discretizando la nube de puntos en vóxeles tridimensionales computando la pertenencia de cada punto de la nube al mismo, almacenando únicamente el número de píxeles en cada vóxel. A continuación, una red neuronal extrae características de cada punto y de cada vóxel, concatenando la información para obtener un vector ampliado. La tercera etapa realiza convoluciones para transformar la información tridimensional a una vista de pájaro bidimensional, obteniendo una pseudo-imagen. La fase final consta de una *Region Proposal Network* similar a [25] que analiza los mapas de características para generar cajas tridimensionales que encuadren a los objetos identificados.
SECOND consigue un incremento en la velocidad de procesamiento gracias a la discretización de la nube de puntos en cubos tridimensionales, obteniendo resultados precisos con menor coste computacional.

2.3.3 PointPillars

Siguiendo la estela de SECOND, PointPillars [12] introduce una discretización de la nube de puntos tridimensional a columnas verticales con el objetivo de simplificar el análisis, obteniendo una cuadrícula bidimensional en vista de pájaro, almacenando las características extraídas de la información en diversos canales. Este procedimiento agiliza el procesamiento aplicando convoluciones 2D de alta eficiencia en unidades de procesamiento gráfico, resultando una red de alta velocidad con resultados precisos.

La arquitectura de la red está compuesta por tres etapas, como se muestra en la Ilustración 10, una primera fase de codificación de características para conversión de nube de puntos a pseudo-imagen, una etapa intermedia bidimensional de procesamiento, y un paso final para inferir cuadros delimitadores tridimensionales.

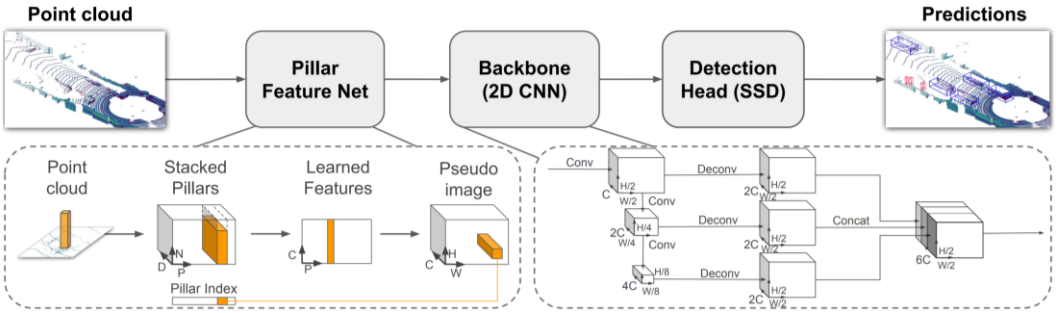


Ilustración 10 Arquitectura de red de PointPillars

La primera etapa discretiza la nube de puntos en una rejilla en vista de pájaro, formando pilares que contienen la información de los puntos que pertenecen a su interior. La información xyz y reflectancia de cada punto es ampliada con la distancia a la media de los puntos de un pilar y al centro geométrico del mismo hasta obtener un total de nueve dimensiones. Una versión simplificada de [22] es empleada para codificar la información y obtener una pseudo-imagen que pueda analizar.

La segunda etapa aplica convoluciones bidimensionales para procesar las características obtenidas, similar a la aplicada en VoxelNet [24], para finalmente obtener cajas tridimensionales que contengan a los objetos detectados aplicando Single Shot Detector [25] empleando la intersección de la unión (IoU) entre la predicción y los datos etiquetados.

PointPillars obtiene un mejor resultado que las redes predecesoras empleando un tiempo de detección hasta cinco veces menor que el que estas utilizaban para realizar un procesamiento completo de la nube de puntos.

2.3.4 Part-A^2

Como evolución de la red PointRCNN, que procesa la nube de puntos sin realizar un procesamiento previo, Part-A^2 [20] hace uso de la localización de puntos que, debido a las características físicas del sensor LIDAR, pueden impactar en el interior del vehículo en lugar de en la forma exterior, por ejemplo, al atravesar un cristal.

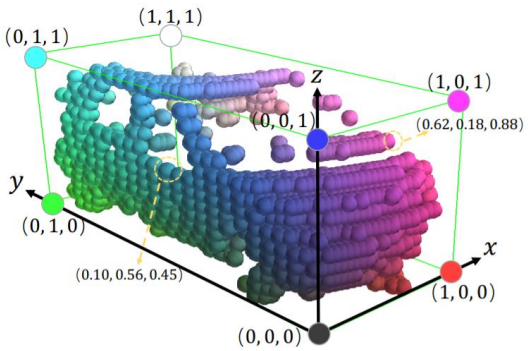


Ilustración 11 Localización de puntos interiores al objeto

Como muestra la Ilustración 11, algunos puntos del sensor LIDAR atraviesan los vidrios del vehículo para impactar en las partes interiores, principalmente a través del parabrisas. Es esta característica la principal ventaja que Part-A^2 intenta explotar.

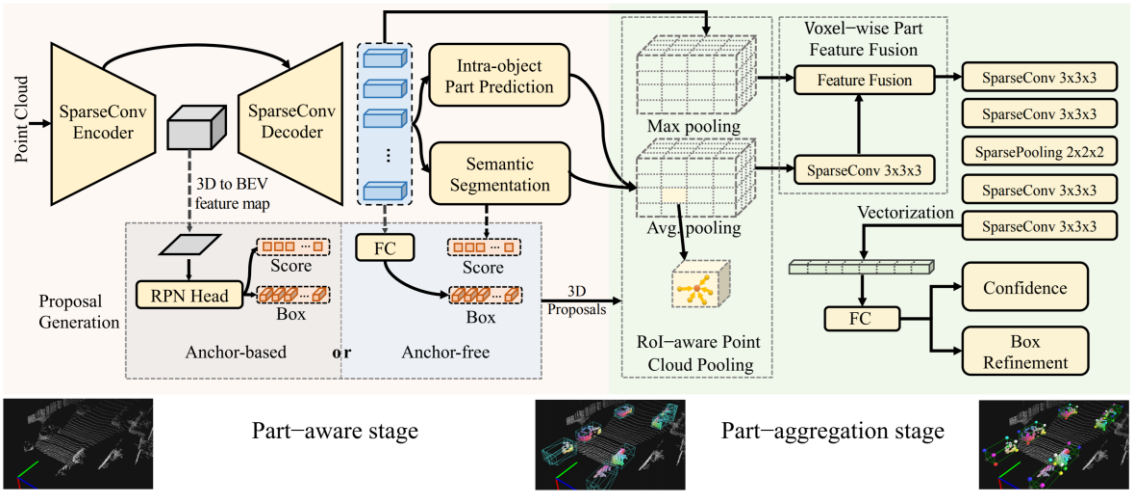


Ilustración 12 Arquitectura de red de Part-A^2-anchor

La red Part-A^2 hace uso de un procesamiento en dos etapas. La primera de ellas extrae características de la red aprendiendo a estimar los puntos interiores a los objetos a detectar, a la vez que genera propuestas de cajas tridimensionales que encuadren al objeto en cuestión. La segunda etapa, encargada de la agregación de los puntos para identificar a los objetos, se divide en dos estrategias: sin emplear *anchors* para ahorrar coste computacional siguiendo el modelo previo [14], o empleándolos para obtener una mejor detección, como se muestra en la Ilustración 12.

2.3.5 PV-RCNN

La última de las redes analizadas, PV-RCNN [21] propone una conjunción de las dos estrategias previamente presentadas. Por un lado, aprovecha la eficiencia computacional de la discretización de la nube de puntos en vóxeles para generar predicciones precisas, mientras que conserva el manejo de la nube de datos sin preprocesamiento para mantener la precisión de la información, combinando la información en una estrategia en dos etapas, precedidas de una primera fase de convoluciones.

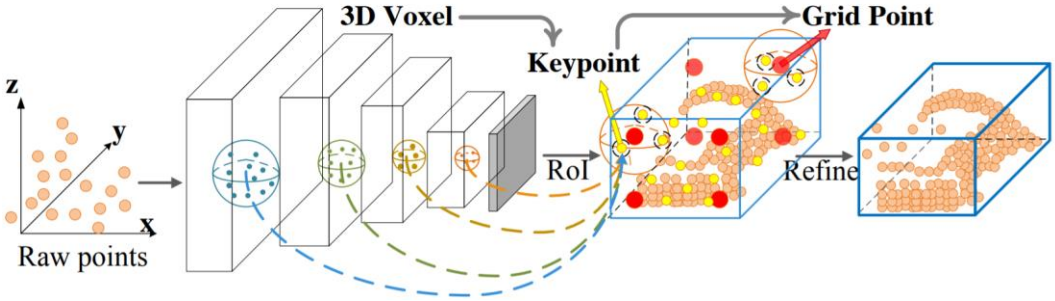


Ilustración 13 Arquitectura de red de PV-RCNN

El primer paso de PV-RCNN discretiza la nube de puntos en vóxeles tridimensionales, aplicando una serie de convoluciones sobre una vista de pájaro de la información procesada, generando propuestas de cajas tridimensionales para contener al objeto detectado. A continuación, se identifican una serie de puntos clave *keypoints* empleados en la fase final de refinamiento de las predicciones gracias a las diferentes características extraídas. Gracias a las cajas previamente definidas y a los *keypoints* extraídos, la siguiente etapa combina la información para obtener una predicción más precisa.

2.3.6 Comparativa de resultados oficiales

Con el objetivo de establecer una comparación entre las distintas redes neuronales para la detección de objetos presentadas, los desarrolladores de métodos de detección de objetos toman como referencia el análisis cuantitativo sobre el *dataset* KITTI en el conjunto de *test*, permitiendo conocer la precisión media (AP) al detectar coches, peatones y ciclistas en un conjunto de imágenes tomadas en entornos urbanos.

A continuación, la Tabla 1 muestra una comparativa del AP sobre KITTI junto con su frecuencia de funcionamiento extraídos de sus correspondientes publicaciones, relevante para una implementación en tiempo real sobre el *framework* OpenPCDet [26].

Método	**Frecuencia (Hz)**	**Coche (AP)**	**Peatón (AP)**	**Ciclista (AP)**
PointPillars [12]	62	79.05	**52.08**	75.78
PointRCNN [14]	10	85.94	49.43	73.93
SECOND [19]	40	83.13	51.07	70.51
Part-A^2 [20]	14	77.86	44.50	62.73
PV-RCNN [21]	12.5	**90.25**	-	**78.60**

Tabla 1 Comparativa de resultados de detección 3D oficiales sobre el set KITTI test

Como se puede comprobar, los resultados de precisión de la red PV-RCNN [21] mejoran con amplio margen a los de redes predecesoras (aunque no aportan datos de detección sobre peatones). Sin embargo, el complejo sistema que compone este método hace que el tiempo de procesamiento sea hasta seis veces inferior que el de redes como SECOND [19] o PointPillars [12], que se benefician de la discretización en vóxeles de la nube de puntos presentando valores de precisión similares con un rendimiento más optimizado.

Los sistemas en tiempo real para conducción autónoma suelen procesar información a velocidades superiores a los 10 Hz debido a la frecuencia de publicación de datos por parte de los sensores. Todos estos parámetros serán tenidos en cuenta durante la evaluación de las redes presentadas en este Trabajo Fin de Máster.

2.4 Seguimiento de objetos mediante técnicas tradicionales

Consecutivamente a la etapa de detección se emplea un módulo de seguimiento de objetos en la escena a través de los diferentes instantes de detección. Estos módulos tienen la función de identificar unívocamente a los objetos encontrados mientras estos sean detectados por los sensores, pudiendo conocer su trayectoria completa. Para ello tradicionalmente se han empleado una serie de algoritmos que trabajan de forma conjunta para elaborar predicciones de posición en el siguiente instante y asociarlas con las detecciones recibidas en base a modelos de velocidad constante.

Partiendo de la detección de objetos de forma tridimensional gracias a la información de un sensor LIDAR, el módulo de seguimiento inicializa un seguidor basado en filtro de Kalman para cada detección. A continuación, todos los filtros de Kalman son actualizados para conocer una posición futura estimada. En el siguiente instante de detección, las posiciones estimadas son comparadas con las detecciones obtenidas, pudiendo realizar asociaciones entre ellas gracias a técnicas como el algoritmo Húngaro [16] de minimización de distancias. Además, un módulo de memoria analiza las detecciones y los seguidores no vinculados, estimando si se debe inicializar un nuevo seguidor para dicha detección y si antiguos seguidores no enlazados deben ser eliminados en base a parámetros prestablecidos en su configuración.

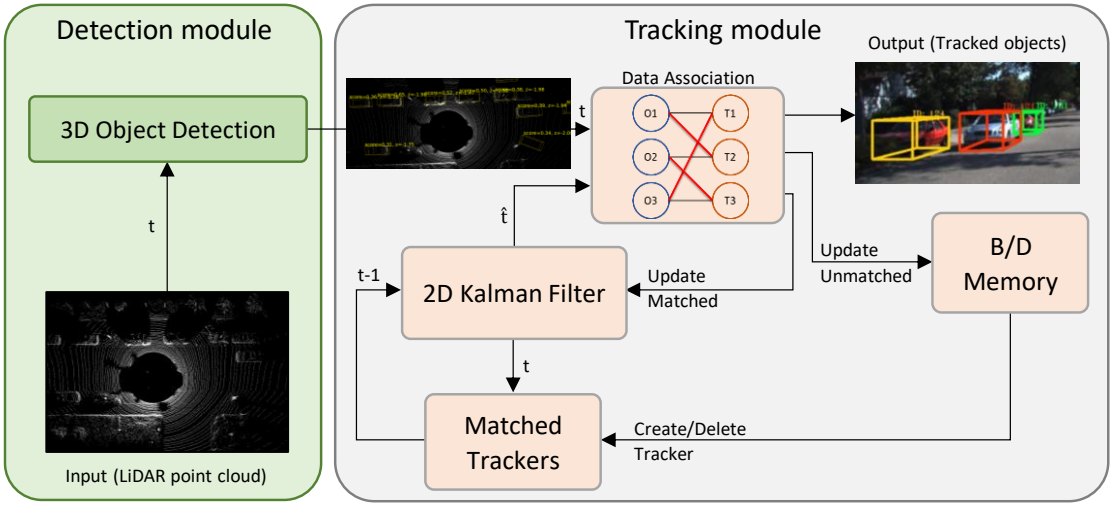


Ilustración 14 Sistema de detección y seguimiento de objetos tridimensionales

Esta técnica se puede desarrollar tanto en el espacio bidimensional en el marco de la imagen [17] como en vista de pájaro para evitar oclusiones entre objetos y distorsiones de magnitud [18], así como en el espacio tridimensional para conocer verdaderamente todas las variables del objeto [2].

Algunas mejoras sobre esta técnica introducen extracción de características visuales de los objetos detectados a través de una imagen [17], añadiendo complejidad a la asociación de datos para evitar el intercambio de identidades en los objetos.
En el desarrollo de este Trabajo Fin de Máster se evaluarán las diferentes metodologías presentadas en este epígrafe.

2.4.1 DeepSORT

En una primera aproximación, se implantará y evaluará el rendimiento del algoritmo DeepSORT [17], el cual presenta como entradas la localización del objeto sobre un plano de imagen y la propia imagen con el objetivo de extraer características visuales para la asociación de identidades.

DeepSORT se basa en una implementación anterior sin características visuales [15], ayudándose de un filtro de Kalman bidimensional con estados (u, v, γ, h, $\dot{u}$, $\dot{v}$, $\dot{\gamma}$, $\dot{h}$), siendo (u, v) el centro de la caja 2D, γ la relación de aspecto, h su altura, y sus respectivas velocidades.

Cada objeto instancia un filtro de Kalman, que incluye un contador para conocer cuántos instantes el identificador no ha sido asociado a ningún objeto. Si este contador supera el umbral Age_{max}, el identificador es eliminado. De forma opuesta, cada objeto no asociado genera un filtro de Kalman, declarado tentativo las primeras veces que se ha detectado. Si dicho objeto y su nuevo identificador coinciden consecutivamente un número de veces superior al umbral min_hits, el identificador es creado con éxito.

En el algoritmo SORT la asociación entre objetos y seguidores se realiza mediante una optimización de distancias aplicando el algoritmo Húngaro [16]. DeepSORT introduce las métricas de apariencia aplicando similitud coseno junto con las de movimiento aplicando la distancia de Mahalanobis [27], combinándolas mediante una suma ponderada, de forma que la diferencia en movimiento permite asociación entre instantes poco separados y la métrica de apariencia permite la recuperación de identificadores no asociados durante un tiempo elevado.

Para la implementación a partir de detección 3D LIDAR se ha realizado en este TFM una adaptación mediante la cual los objetos detectados en el espacio tridimensional

son proyectados al plano imagen empleando las calibraciones de cada escena. Estos objetos son seguidos sobre la imagen y posteriormente devueltos a coordenadas tridimensionales, recuperando la información proveniente de la detección para su correcto emplazamiento en la escena tridimensional.

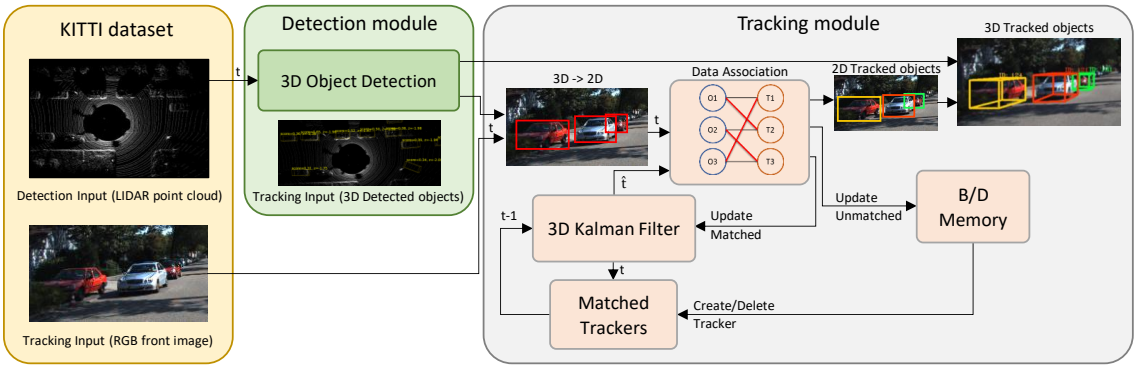


Ilustración 15 Estructura DAMOT con características visuales DeepSORT

La extracción de características visuales es llevada a cabo por una red neuronal convolucional (CNN) entrenada sobre la base de datos de re-identificación de personas MARS [28], que dispone de más de un millón de imágenes con 1261 peatones. Aunque la red está entrenada específicamente para obtener características de peatones, se aplica también para vehículos y ciclistas debido a que el objetivo es el mantenimiento de características básicas como el color. Los resultados de esta extrapolación, presentados en 5.2.2, muestran un desempeño aceptable al aplicar el sistema sobre coches, peatones y ciclistas en el dataset KITTI.

Capa	**Tamaño de salida**
Convolucional 1	32 x 128 x 64
Convolucional 2	32 x 128 x 64
Max Pool 3	32 x 64 x 32
Residual 4	32 x 64 x 32
Residual 5	32 x 64 x 32
Residual 6	64 x 32 x 16
Residual 7	64 x 32 x 16
Residual 8	128 x 16 x 8
Residual 9	128 x 16 x 8
Dense 10	128
Batch & normalization	128

Tabla 2 Arquitectura de red de extracción de características visuales

La red neuronal convolucional empleada para la extracción de características visuales a partir de imagen RGB es de pequeño tamaño, permitiendo una ejecución rápida en equipos con recursos limitados. La arquitectura se describe en la Tabla 2, donde se puede apreciar que la salida es un vector unidimensional con 128 parámetros que describe al objeto en cuestión. La diferencia entre los vectores obtenidos en un instante y el siguiente permite emparejar detecciones a través del tiempo, obteniendo vectores prácticamente idénticos cuando se trate del mismo objeto.

2.4.2 AB3DMOT

El algoritmo AB3DMOT [2] está diseñado específicamente para conducción autónoma y amplía la base de SORT [15] empleando un filtro de Kalman tridimensional. De esta forma el sistema es capaz de conocer la verdadera magnitud de las variables de desplazamiento y tamaño del objeto, evitando distorsiones por proyección al plano bidimensional de imagen.

Empleando predicción y asociación tridimensional se evita posibles errores por oclusiones que suceden en el espacio bidimensional de forma que el objeto está claramente separado del resto, aplicando el algoritmo Húngaro a la intersección sobre la unión (IoU) entre detecciones y seguidores como cajas tridimensionales.

El hecho de no aplicar extracción de características hace que el algoritmo presente una alta eficiencia computacional, llegando a funcionar a frecuencias superiores a 200 Hz sobre CPU, mientras que el uso del espacio tridimensional hace que el sistema alcance unas tasas de acierto en el dataset KITTI muy elevadas.

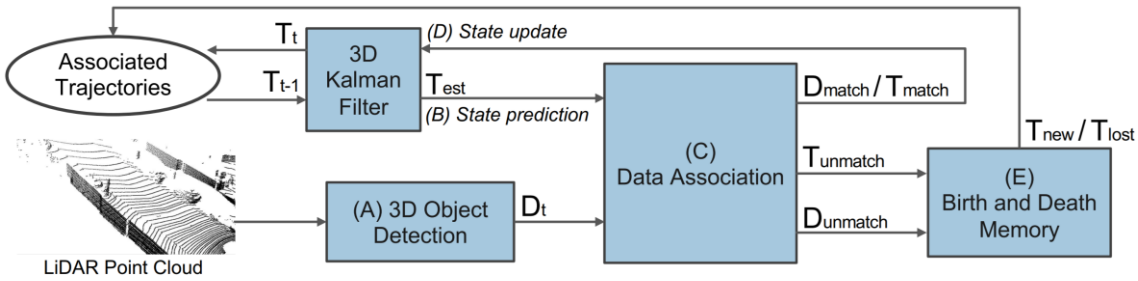


Ilustración 16 Esquema de funcionamiento de AB3DMOT

El resto del sistema de control de seguidores asociados durante un número de instantes para su eliminación o creación se implementa de forma similar a la explicada en 2.4.1 DeepSORT.

Además, AB3DMOT presenta una herramienta de validación adicional sobre el dataset KITTI. El código de evaluación de KITTI comprueba el correcto desempeño del seguimiento según la proyección al plano de la imagen, mientras que la herramienta 3D MOT evalúa la asociación en el espacio tridimensional, obteniendo resultados más realistas.

2.5 Seguimiento de objetos mediante técnicas de aprendizaje profundo

Al igual que en los sistemas de detección de objetos, actualmente el estado del arte trabaja para conseguir métodos de seguimiento de objetos basados parcial o totalmente en técnicas de aprendizaje profundo con la intención de extraer características complejas de la información y obtener una mejor identificación de los objetos.

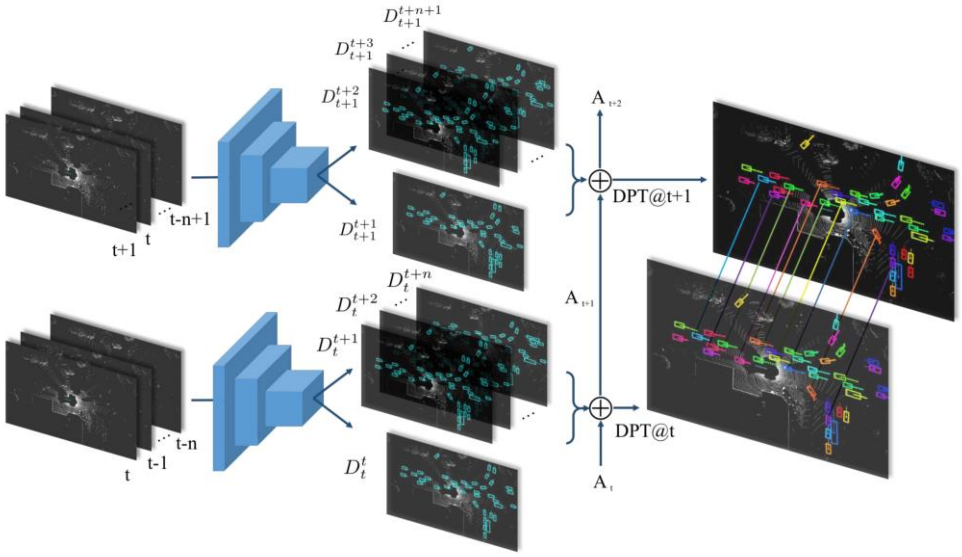


Ilustración 17 Arquitectura de red de Fast and Furious [13]

Algunas técnicas aplican redes neuronales para la etapa de asociación de datos exclusivamente [29] introduciendo características de localización y apariencia evitando algoritmos de optimización. Otras metodologías proponen una sustitución completa de los algoritmos tradicionales por una red neuronal [13] o un conjunto de ellas [30] procesando información exclusivamente de LIDAR o LIDAR con cámaras RGB respectivamente explotando al máximo la información de los sensores.

Un ejemplo de estas técnicas está presente en el método Fast and Furious [13], mostrado en la Ilustración 17. En él se aplica una red neuronal profunda que es capaz de realizar detección tridimensional, así como el seguimiento de dichos objetos y la predicción de su posición en el futuro, adquiriendo una mayor robustez frente a oclusiones.

Sin embargo, estas técnicas son todavía recientes en el campo de la detección de objetos y suelen ser desarrolladas por empresas dedicadas al transporte de pasajeros con vistas a la automatización de sus sistemas, por lo que, aunque sus avances son compartidos con la comunidad científica, la herramienta no está disponible para su aplicación.

Capítulo 3.
Herramientas aplicadas.

El entrenamiento y validación de redes neuronales requiere del procesamiento de un gran conjunto de datos del que extraer características. Los *datasets* o conjuntos de datos han sido desarrollados en paralelo con la investigación en aprendizaje profundo, obteniendo cada vez un mayor número de sensores recolectados y de diferente naturaleza, así como en múltiples situaciones de tráfico y meteorológicas para tratar de cubrir los eventos que pueden suceder en un funcionamiento real.

Uno de los conjuntos de datos más importantes en el ámbito de la navegación autónoma es KITTI [1], desarrollado por una colaboración entre el Karlsruhe Institute of Technology y el Toyota Technological Institute de Chicago. En él se recogen diferentes escenas para el estudio de la detección y el seguimiento de objetos, así como herramientas para el análisis de os resultados.

Posteriormente se han desarrollado otros *datasets* con mayor riqueza gracias a la utilización de un mayor número de sensores o la aportación de más escenas, como nuScenes [31] o Waymo Open Dataset [32].

Sin embargo, los conjuntos de datos estáticos, aunque permiten un alto nivel de precisión en el entrenamiento, apenas recogen situaciones complejas para el sistema, por lo que no son suficientemente potentes para evaluar el comportamiento en un entorno complicado como es la conducción real.

Para cubrir este déficit actualmente la industria automovilística está apostando por los simuladores de conducción realistas, incluyendo los sensores estándar en conducción autónoma para poder evaluar el comportamiento de los sistemas en entornos tan complicados como se puedan imaginar sin producir riesgos ni incrementar costes.

El objetivo final de este TFM es la implementación de un sistema funcional de detección y seguimiento de objetos sobre un vehículo autónomo en desarrollo del grupo RobeSafe para el proyecto Techs4AgeCar. En él se incluyen sensores punteros en conducción autónoma como cámaras de visión estéreo y sensor LIDAR.

El sistema final de detección y seguimiento será instalado en un contenedor Docker, de forma que el sistema permita retrocompatibilidad con las sucesivas actualizaciones del sistema embebido en el que se encuentra sin que afecte a su funcionalidad.

3.1 Dataset KITTI

El conjunto de datos de KITTI ha sido empleado como estándar internacional en entrenamiento y evaluación de técnicas para conducción autónoma desde su introducción en 2012, recolectando información de los sensores más comunes en conducción autónoma equipados sobre un vehículo controlado manualmente. Los datos son manualmente etiquetados para obtener información precisa de los elementos que se encuentran en el campo de visión de la cámara frontal.

Los sensores que conforman el *dataset* son:

- Dos cámaras en escala de grises (FL2-14S3M-C) y dos cámaras a color (FL2-14S3C-C) para el estudio de profundidad a partir de información estéreo.
- Un LIDAR Velodyne HDL-64E de 64 haces que capta nubes de puntos 360º a una frecuencia de 10 Hz.
- Un sistema de localización inercial y GPS (OXTS RT3003) funcionando a 100 Hz.

El sensor LIDAR proporcionará la nube de puntos necesaria para identificar los objetos en el entorno, mientras que la imagen de una de las cámaras RGB aportará características visuales para algunos de los métodos de seguimiento de objetos.

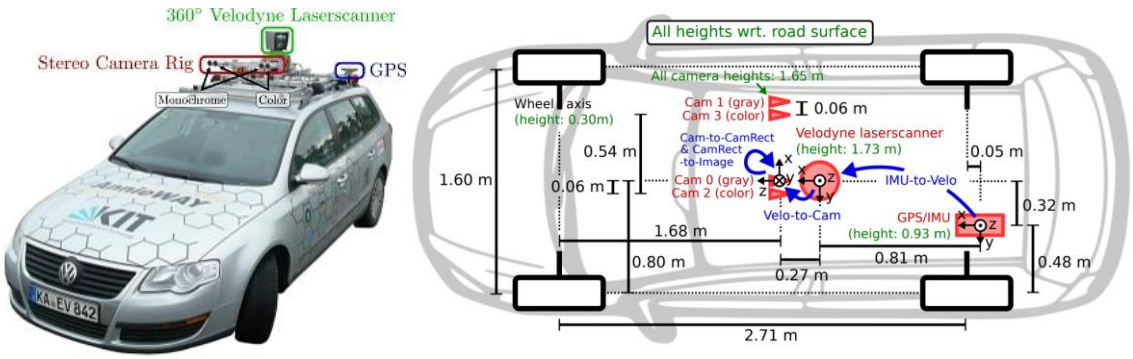


Ilustración 18 Disposición de sensores en conjunto de datos KITTI

KITTI proporciona diferentes conjuntos de datos según la tarea a realizar. Así, para entrenamiento y validación de detección de objetos se emplea KITTI object, mientras que para el análisis de técnicas de seguimiento de objetos proporciona KITTI tracking.

3.1.1 KITTI object dataset

KITTI object es el subconjunto de datos empleado para herramientas de detección de objetos en tres dimensiones. Para llevar a cabo la tarea proporcionan imágenes, calibraciones de sensores, nubes de puntos y etiquetas de objetos sincronizadas.
Las etiquetas permiten conocer las características de los objetos que pertenecen al campo de visión de la cámara, y se almacenan siguiendo el formato mostrado en la Tabla 3.

Las calibraciones proporcionan matrices de transformación entre la cámara principal (cámara RGB izquierda) y el sensor LIDAR, y el módulo IMU/GPS y LIDAR, así como las matrices de proyección de las cámaras para computar la transformación entre mundo tridimensional e imagen.

Para realizar la evaluación se almacenan los resultados de la inferencia por parte del algoritmo siguiendo el formato descrito en la Tabla 3, de forma que los códigos proporcionados comparan los objetos detectados con los identificados manualmente para establecer el nivel de precisión del método.

Nombre	Descripción
Tipo	Describe el tipo de objeto: Coche, Furgoneta, Camión, Peatón, Persona sentada, Ciclista, Tranvía, miscelánea o sin definir.
Truncado	Valor entre 0 y 1 indicando si el objeto se sale de su caja
Ocluido	Entero 0,1,2,3 indicando el nivel de oclusión: 0: totalmente visible 1: parcialmente ocluido 2: muy ocluido 3: desconocido
Alpha	Ángulo de observación [-π, π]
Bounding Box	Caja bidimensional que encuadra al objeto. Define esquina superior izquierda e inferior derecha en píxeles.
Dimensiones	Dimensiones en espacio tridimensional
Localización	Localización en espacio tridimensional según ejes de cámara
Rotación	Rotación del objeto según eje vertical [-π, π]
Puntuación	Los resultados incluyen una valoración de la seguridad en la detección

Tabla 3 Descripción de las etiquetas de objetos en KITTI object

3.1.2 KITTI tracking dataset

El conjunto de datos proporcionado para herramientas de seguimiento de objetos sigue un esquema similar al explicado en 3.1.1 KITTI object dataset, ampliando la información etiquetada para añadir un identificador único a cada objeto, que no se repite mientras dure la secuencia.

Además, la información es presentada como una serie de instantes tomados consecutivamente de forma que se puede analizar la trayectoria de los objetos en la escena, mientras que en 3.1.1 KITTI object dataset la información se distribuía de forma aleatoria en distintos escenarios para aumentar la variedad de situaciones.

3.2 Simulador CARLA

El simulador de conducción hiperrealista de código abierto CARLA [3] proporciona una fuente de información para evaluar sistemas de conducción autónoma completos en entornos urbanos. La fácil adaptación del simulador a cualquier sistema autónomo, así como la integración de los sensores más comunes en navegación autónoma hacen que el sistema sea prácticamente un sustituto de entornos reales sin realizar modificaciones. CARLA permite modificar aspectos dinámicos del vehículo, características de los sensores e incluso las condiciones meteorológicas de la escena, aplicando un alto realismo a situaciones complejas como lluvia o noche.

Ilustración 19 CARLA: Escena diurna con peatones en ciudad y escena nocturna con lluvia en carretera

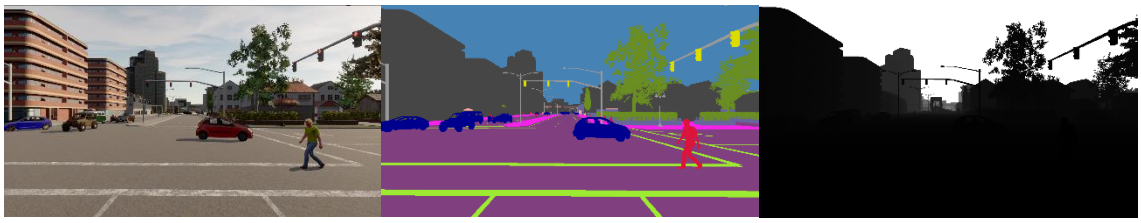
3.2.1 Motor gráfico

El simulador CARLA está desarrollado sobre el motor gráfico Unreal Engine 4 [33], lo que aporta un gran realismo gráfico y físico a la par que flexibilidad gracias a su diseño puntero y la variedad de plugins disponibles. CARLA se basa en la metodología servidor-cliente, realizando la simulación en el servidor y facilitando la conexión al mismo de uno o varios clientes, llamados agentes, que son los objetos dinámicos disponibles (vehículos, peatones, ciclistas). Los agentes se comunican con el servidor para enviar comandos de control, que el simulador procesa, devolviendo la actualización al cliente.

3.2.2 Sensores de conducción autónoma

El simulador CARLA está diseñado específicamente para entrenamiento y validación de sistemas de conducción autónoma, por lo que incluye todos los elementos necesarios para ello. Los vehículos presentan un alto realismo en su dinámica física para simular correctamente el control del vehículo, mientras que los módulos de percepción disponen de una suite de sensores totalmente modificables.

La versión 0.9.10, la última disponible al redactar este Trabajo Fin de Máster, incluye una gran variedad de sensores que continúan mejorando y ampliándose en cada versión. En cuanto a cámaras, el simulador provee cámaras a color RGB, cámaras de profundidad y cámaras con segmentación semántica artificial, añadiendo efectos de distorsión de lente, sensibilidad luminosa o tiempo de exposición. La Ilustración 20 muestra un ejemplo de la información proporcionada por los tres sensores de visión artificial integrados en el simulador.



a) RGB *b) Segmentación semántica* *c) Profundidad*

Ilustración 20 Sensores de visión artificial en simulador CARLA

CARLA permite incorporar múltiples sensores basados en tecnología LIDAR con características modificables, pudiendo disponer de información obtenida por un sensor de muy altas prestaciones como 128 haces sin coste alguno, suponiendo un elevado ahorro en la implantación de estas tecnologías. Una de las mejoras introducidas en la última versión disponible del simulador ha mejorado el realismo proporcionado por este sensor, pasando de representar los vehículos como objetos cuadrangulares a seguir su forma exacta como lo haría un sensor LIDAR real, como se puede apreciar en la Ilustración 21.

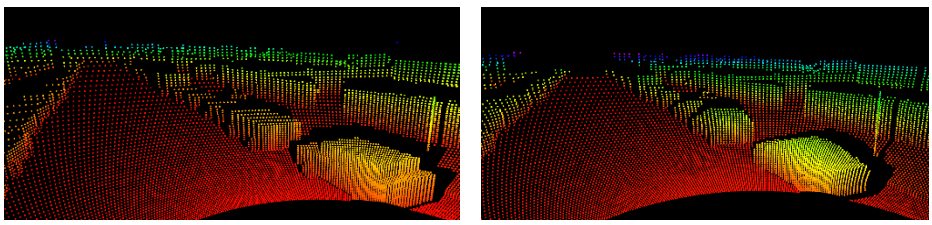


Ilustración 21 Comparativa de sensor LIDAR entre versiones CARLA 0.9.9 y 0.9.10

Además, incorpora otros sensores como GPS, sensores inerciales IMU, radar, LIDAR semántico o cámaras de evento DVS para cubrir todas las opciones posibles en navegación autónoma.

3.2.3 Ecosistema CARLA y ROS

Para aumentar las posibilidades del simulador, CARLA dispone de múltiples *add-ons* disponibles a través de su GitHub que permite añadir funcionalidades específicas en función del trabajo a realizar, como la posibilidad de diseñar escenarios con objetos dinámicos concretos para añadir repetibilidad, editores de mapas, módulos de aprendizaje por refuerzo (*reinforcement learning*) así como un paquete de comunicación mediante ROS.

ROS [34] (Robotics Operative System) es una plataforma de desarrollo diseñada específicamente para aportar flexibilidad a la comunicación entre sensores y actuadores en robots. Presenta una serie de herramientas y librerías que facilitan el intercambio de información a través de protocolos Ethernet.

El paquete CARLA-ROS-bridge permite establecer un enlace entre los datos del simulador y el ecosistema ROS, de forma que la información del simulador es visible a través de diversos tópicos, y a la vez es posible actuar sobre los agentes del simulador. De esta forma, el control de los vehículos de forma externa se implementa de forma sencilla, a la vez que se recibe la información de los diversos sensores mediante mensajes estandarizados para poder realizar su procesamiento. La Ilustración 22 muestra las interacciones entre CARLA-ROS-Bridge, el servidor y los clientes del simulador y el usuario.

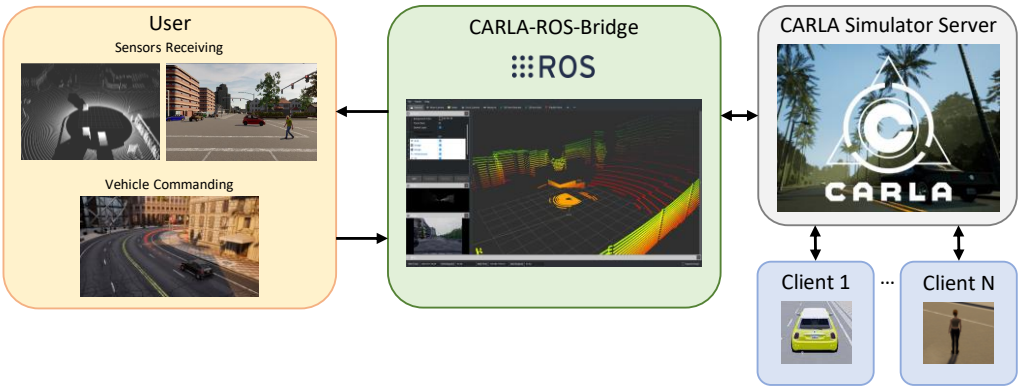


Ilustración 22 Esquema de interacciones de CARLA-ROS-Bridge

3.3 Proyecto Techs4AgeCar

El proyecto Techs4AgeCar, continuación de SmartElderlyCar, es una investigación financiada por el Ministerio de Ciencia e Innovación con el objetivo de desarrollar un sistema de conducción autónoma en la Universidad de Alcalá implementando las tecnologías más avanzadas.
El vehículo ha evolucionado en las divisiones hardware, software y estructural a partir de un chasis de vehículo eléctrico de código abierto TABBY EVO, adquiriendo sucesivas mejoras que progresan sus capacidades de conducción autónoma gracias al trabajo colaborativo entre el grupo de investigación GROBIS perteneciente a la Universidad de Vigo en el anterior proyecto, y el grupo de investigación RobeSafe, de la Universidad de Alcalá.

Ilustración 23 Vehículo Techs4AgeCar

3.3.1 Sensores de percepción para navegación autónoma

Una parte esencial de la navegación autónoma es la percepción del entorno. Para ello, el vehículo integra un LIDAR Velodyne VLP-16 en su parte superior, así como una cámara de visión estéreo a color ZED. Estos sistemas se agregan en una plataforma integrada en la parte superior del vehículo de forma que las relaciones entre los sensores permanecen constantes.

El procesamiento de estos sensores se integrará en el módulo de desarrollo Nvidia Jetson AGX Xavier, aplicando la máxima potencia en unidad de procesamiento gráfico en un sistema embebido disponible actualmente.
La Ilustración 24 muestra la plataforma compuesta por los sensores y el sistema embebido mencionados, indicando la posición de los mismos.

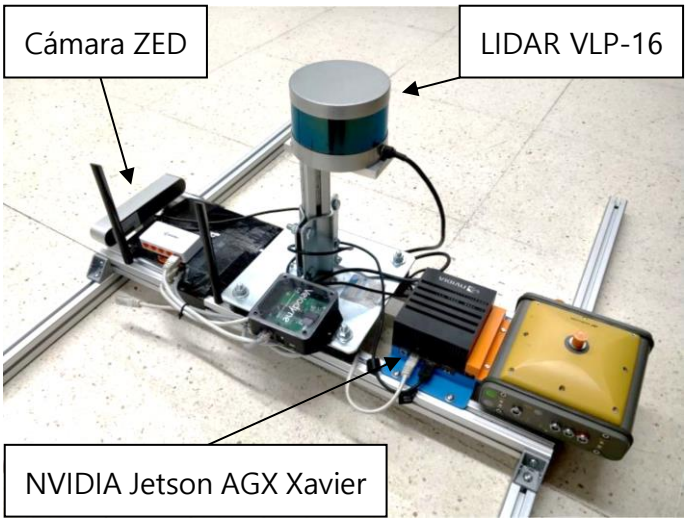


Ilustración 24 Plataforma de sensores del vehículo Techs4AgeCar

3.3.1.1 LIDAR VLP-16

Los sensores de distancia activos basados en láser permiten una elevada precisión a la hora de reconstruir el entorno de forma tridimensional en comparación con otros métodos como radar o sensores ultrasónicos. El dispositivo emite un pulso láser en una dirección concreta, calculando el tiempo transcurrido entre la emisión y la recepción por el sensor integrado para conocer la distancia de impacto de dicho punto sobre una superficie cualquiera. La agregación de este sistema en diferentes orientaciones y elevaciones genera una nube de puntos tridimensional.

El vehículo Techs4AgeCar integra un sensor LIDAR Velodyne VLP-16, integrando 16 haces horizontales con 360º de campo de visión y una elevación sobre la horizontal de ±15º, con capacidad de generar 300.000 puntos por segundo a frecuencias entre 5 y 20 Hz hasta una distancia de 100 metros. La Ilustración 25 recoge las características físicas del sensor junto con una nube de puntos obtenida con el mismo.

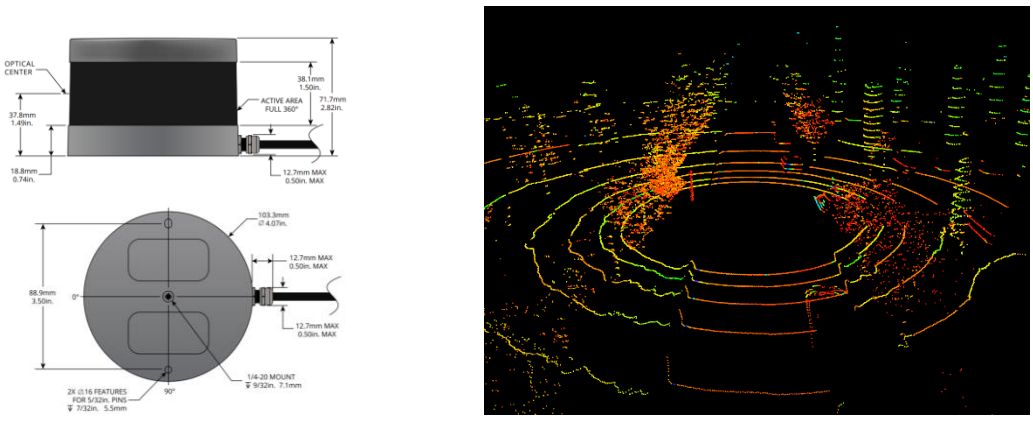


Ilustración 25 Formato Velodyne VLP-16 y nube de puntos en Campus Externo UAH

3.3.1.2 Cámara ZED

Una parte esencial de la percepción del entorno es la extracción de características que enriquezcan la información. Por ello, el vehículo Techs4AgeCar integra la cámara de visión estéreo ZED, pudiendo obtener mapas de profundidad en el campo de visión, así como imágenes RGB para percepción de objetos.

Esta cámara permite obtener percepción de profundidad con elevada precisión hasta 20 metros de distancia, procesando video de alta calidad con una tasa de refresco de hasta 100 Hz. Además, su ecosistema permite una fácil integración con entornos de desarrollo como MATLAB, Unreal Engine o Python, así como su procesamiento en sistemas embebidos de NVIDIA.

Ilustración 26 Formato de cámara ZED, visión RGB y mapa de profundidad

3.4 Docker

Docker es una herramienta de código abierto que permite el desarrollo de módulos en contenedores independientes. Gracias a ello, un proyecto puede dividirse en bloques con diferentes requisitos, realizándose cada uno de ellos en un contenedor independiente, pudiendo establecer comunicaciones entre los mismos. Cada contenedor puede tener instalada una versión distinta del sistema operativo y versiones diferentes de librerías necesarias para los distintos sistemas.

El desarrollo de este TFM implementa la solución Docker de forma que la herramienta desarrollada queda aislada del sistema anfitrión, pudiendo exportarse con relativa facilidad a otros equipos sin afectar a las instalaciones que estos pudieran tener configuradas, evitando la posible obsolescencia producida con la evolución de las herramientas aplicadas.

Capítulo 4.
Desarrollo e implementación de sistema de detección y seguimiento.

La elevada complejidad del entorno de circulación de un vehículo autónomo hace imprescindible la integración de un sistema eficiente de detección y seguimiento de vehículos, peatones y ciclistas con el objetivo de aumentar su seguridad y evitar posibles accidentes.

Para ello, este Trabajo Fin de Máster integrará las técnicas del estado del arte más apropiadas en cada caso introducidas en el Capítulo 2, procesando la información obtenida por un sensor LIDAR y, en su caso, una cámara RGB, estableciendo comunicaciones mediante el espacio de trabajo ROS e integrando el sistema en una plataforma de desarrollo embebida NVIDIA Jetson AGX Xavier para su funcionamiento en tiempo real, como se muestra en la Ilustración 27.

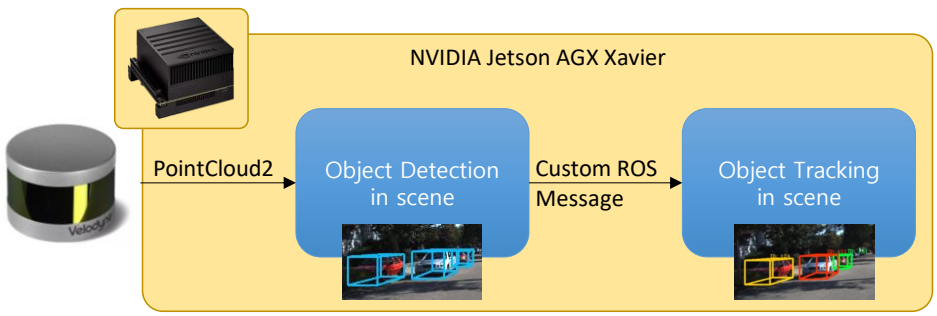


Ilustración 27 Procedimiento de detección y seguimiento de objetos en escena

4.1 Espacio de trabajo para redes de detección de objetos

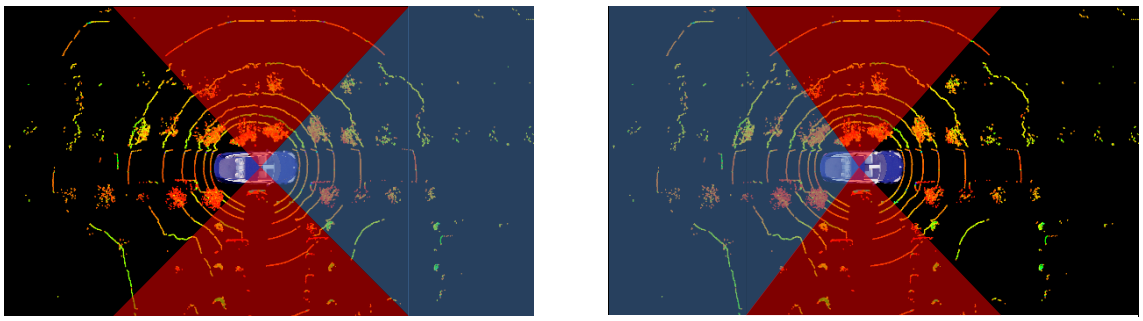
La integración de las diferentes redes neuronales presentadas en el Capítulo 2 se realizará empleando el espacio de trabajo OpenPCDet [26], que dispone de la implementación de las distintas redes neuronales de detección de objetos a partir de nube de puntos tridimensional para su implementación y estudio.

OpenPCDet es un proyecto de código abierto desarrollado por OpenMMLab. Su principal aportación es integrar en un mismo espacio de trabajo diferentes redes neuronales especializadas en la detección de objetos en tres dimensiones a partir de nube de puntos LIDAR para facilitar su ejecución y comparativa. Este ecosistema proporciona las arquitecturas de red del estado del arte, así como un conjunto de modelos entrenados previamente sobre el dataset KITTI, ahorrando el tiempo de procesamiento computacional requerido para su entrenamiento (entre 1 y 5 horas empleando la potencia equivalente a ocho tarjetas TITAN XP).

Gracias a la conexión mediante ROS, el sistema puede recibir la nube de puntos de diversas fuentes, como la base de datos KITTI, el simulador de conducción CARLA o el Velodyne VLP-16 del vehículo Techs4AgeCar con sencillez empleando el estándar PointCloud2, realizando un procesamiento y devolviendo la información correspondiente a los objetos detectados en un mensaje personalizado (*custom*).

Debido a que KITTI solo dispone de etiquetado de datos en el campo de visión de la cámara frontal, las redes neuronales entrenadas con esta base de datos no pueden inferir sobre elementos externos a dicha apertura de campo. Para obtener una detección en todo el entorno del vehículo autónomo aprovechando la total cobertura ofrecida por el sensor LIDAR frente al limitado campo de visión de la cámara RGB se pueden aplicar diferentes técnicas.

La primera de ellas es el doble análisis de la nube de puntos, tomando inicialmente la información obtenida por el sensor para detectar objetos en la parte frontal y posteriormente rotando la nube 180 grados de forma que ubica en el eje X positivo el segmento de nube de puntos que inicialmente se encontraba por detrás del vehículo. Esta técnica, aunque es la más efectiva puesto que mantiene el punto de vista con el que fue entrenada la red, requiere de doble inferencia sobre la nube de puntos para detectar objetos en todo el entorno, reduciendo la velocidad de ejecución a la mitad.



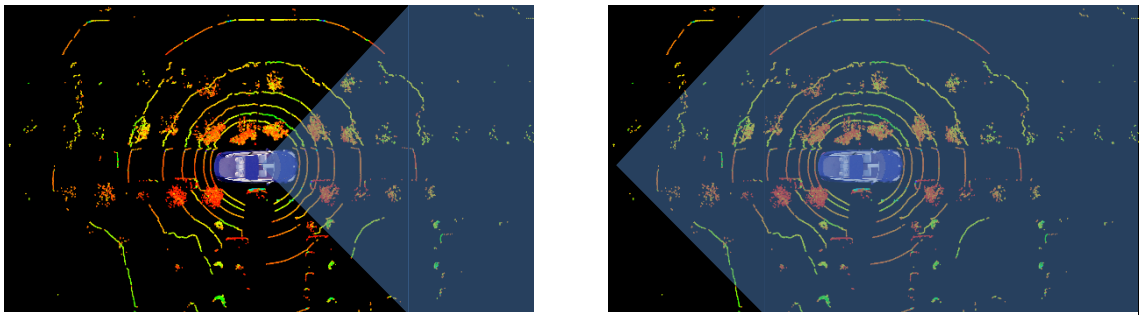
a) Análisis frontal *b) Análisis trasero con giro*

Ilustración 28 Giro de nube de puntos para procesamiento 360º

La Ilustración 28 muestra la aplicación de este método, analizando la zona sombreada en cada caso, primero la parte delantera en Ilustración 28 a) y, aplicando un giro a la nube de puntos, la parte trasera en Ilustración 28 b).

Puesto que el área cubierta del entrenamiento no tiene un ángulo de visión de 180 grados, la aplicación de este sistema no es capaz de cubrir las áreas laterales del vehículo (Ilustración 28 zona roja), en las que, aunque el movimiento propio es más limitado, pueden desplazarse otros objetos que supongan una amenaza para la seguridad.

El otro método de análisis de todo el entorno del vehículo consiste en desplazar la nube de puntos a lo largo del eje X de forma que parte de la información que antes quedaba en la parte trasera del vehículo entra dentro del área de análisis. De esta forma, el punto de vista cambia, pero es capaz de seguir detectando los objetos de forma correcta. Aunque pierde precisión, mantiene el tiempo de cómputo, por lo que es la mejor técnica para un sistema de tiempo real en un medio embebido.



a) Situación inicial *b) Análisis tras desplazamiento*

Ilustración 29 Traslación de nube de puntos para procesamiento 360º

La Ilustración 29 a) muestra cómo originalmente el campo de visión (zona sombreada) sólo cubre la parte frontal de la nube de puntos, mientras que al realizar la traslación (Ilustración 29 b) la red neuronal procesa también la información de la parte trasera del vehículo.

Debido a la eficiencia computacional de esta segunda opción, así como los buenos resultados obtenidos, presentados en el Capítulo 5, se escoge para ser implementada en el sistema de detección empleado. Para ello, la nube de puntos es trasladada mediante un parámetro configurable al inicio de la ejecución, siendo fijada en 20 metros para el análisis realizado en este TFM. Por tanto, la detección realizada cubre una rejilla rectangular de 70 x 80 metros, cubriendo 50 metros en el frontal y 20 metros en la parte trasera, así como 40 metros por cada lado del vehículo.

4.2 Espacio de trabajo para seguimiento de objetos

Mientras que el entorno de trabajo para detección de objetos integra todas las redes analizadas del estado del arte, los sistemas de seguimiento de objetos se presentan de forma aislada en sus integraciones por parte de los creadores. Así, para cada técnica de seguimiento se ha debido extraer y adaptar el código a las necesidades del estudio.

El algoritmo de seguimiento de objetos DeepSORT [17] ha sido modificado para incluir detecciones realizadas sobre LIDAR en lugar de sobre la imagen a partir de la implementación proporcionada en [35], extrayendo características visuales a partir de la proyección de la detección realizada sobre LIDAR a imagen. El nuevo algoritmo incluye recepción de imagen y seguimiento sobre ella mediante mensajes de ROS, de forma que por cada detección realizada sobre la nube de puntos se actualiza el seguimiento de los objetos.

Por otra parte, el algoritmo AB3DMOT [2] tan solo necesita conocer las características del objeto detectado en cuanto a localización y dimensiones, ya que no realiza extracción de características a partir de la imagen. Este método ha sido adaptado para su funcionamiento en tiempo real mediante la plataforma ROS, recibiendo un mensaje personalizado con la información necesaria sobre los objetos detectados.

Ambos métodos permiten la visualización de los resultados mediante mensajes tipo Marker en la herramienta Rviz, pudiendo observar la superposición de la nube de puntos con los objetos detectados, así como la permanencia de los identificadores gracias a los distintos colores asignados a cada marcador, como se puede ver en la Ilustración 30.

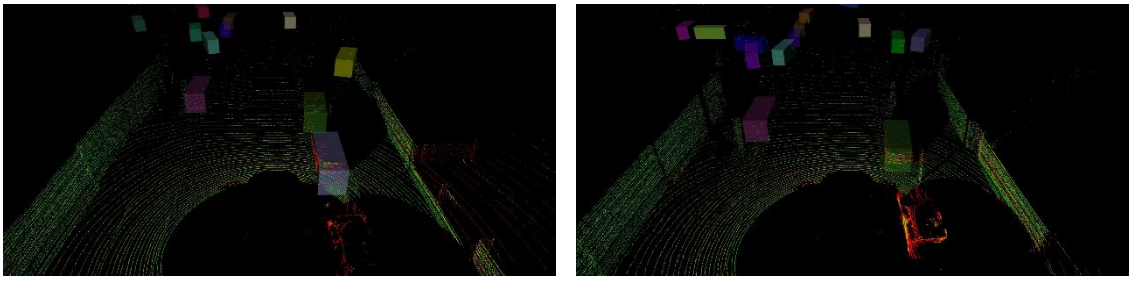


Ilustración 30 Muestra de distintos instantes en seguimiento de objetos en KITTI

4.3 Sistema embebido NVIDIA Jetson AGX Xavier

Los sistemas de conducción autónoma deben procesar una elevada cantidad de información procedente de diversos sensores con el objetivo de entender su movimiento y el de los objetos que lo rodean. Sin embargo, variables como el espacio o el consumo deben optimizarse para ser implementables en un vehículo real, buscando otorgar la máxima comodidad interior con el mínimo volumen de vehículo.

Por ello, las unidades de procesamiento empleadas en vehículos autónomos se integran en sistemas embebidos de alta potencia, disponiendo de una elevada capacidad de cómputo en un tamaño reducido. Debido a las ventajas de la aplicación de redes neuronales en unidades de procesamiento gráfico, los sistemas embebidos desarrollados presentan una potente unidad GPU, así como almacenamientos rápidos basados en discos de estado sólido y elevado tamaño de memoria RAM.

Actualmente los sistemas embebidos que presentan un mayor rendimiento en un menor tamaño son las distribuidas bajo la plataforma NVIDIA Jetson. Estos kits de desarrollo permiten implementar las tecnologías más modernas en redes neuronales para el máximo aprovechamiento de la tarjeta, como librerías CUDA, cuDNN o TensorRT.

Con el objetivo de disponer de la mayor capacidad de procesamiento para una conducción ágil y segura, este TFM desarrolla su implementación sobre el sistema embebido NVIDIA Jetson AGX Xavier, la unidad más potente disponible en el momento de la realización del proyecto.

Módulo	**Características NVIDIA Jetson AGX Xavier**
GPU	512-Core Volta GPU. 64 Tensor Cores
Velocidad	5 TeraFLOPS (FP16) / 10 TeraFLOPS (INT8)
CPU	8-Core ARM 64-Bit
Memoria RAM	16 GB LPDDR4
Almacenamiento	32 GB eMMC
Conectividad	3x USB 3.1 / 4x USB 2.0 / Ethernet Gigabit
Tamaño	100 x 87 mm
Potencia	30W

Tabla 4 Especificaciones NVIDIA Jetson AGX Xavier

Como se puede ver en la Ilustración 24, las reducidas dimensiones del sistema embebido empleado permiten su sencilla integración en el módulo sensorial del vehículo Techs4AgeCar, encontrándose próximo a los sensores que proveen la información.

Además, su escaso consumo de potencia facilita su uso en vehículos eléctricos donde la optimización de la duración de la batería es fundamental, obteniendo el mejor resultado posible con un consumo inferior al de sistemas empleados en equipos más potentes de escritorio como la tarjeta NVIDIA GeForce GTX 1080Ti, con unos requisitos de hasta 250W dedicados exclusivamente al procesamiento gráfico.

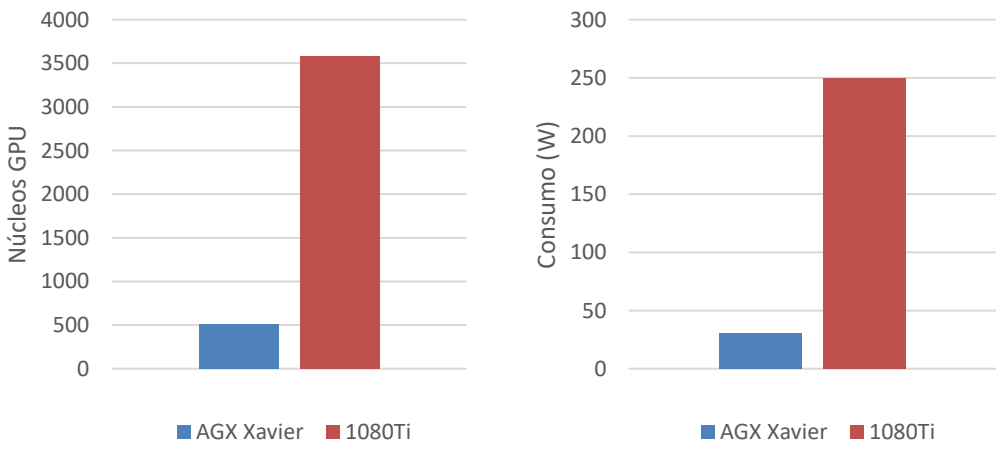


Ilustración 31 Comparativa entre sistema embebido AGX Xavier y de escritorio 1080Ti

Siguiendo los datos indicados en la Ilustración 31, el sistema de escritorio empleado en el desarrollo de este Trabajo Fin de Máster tiene una capacidad de cómputo 7 veces superior a la ofrecida por el sistema embebido, con un consumo 8.3 veces superior dedicado íntegramente al procesamiento gráfico, mientras que la potencia del sistema embebido se dedica también a otras tareas.

Por lo tanto, la implementación de un sistema embebido es a escala más eficiente que el uso de tarjetas gráficas de escritorio. La distribución del procesamiento de los diversos sensores en múltiples sistemas embebidos para su paralelización resultaría en un consumo de potencia menor que el uso de una tarjeta convencional.

Capítulo 5.
Análisis del sistema.

La validación del sistema desarrollado requiere de la realización de análisis cuantitativos y cualitativos de los resultados obtenidos en su funcionamiento.

El sistema empleado para realizar la validación previa a la implementación sobre hardware embebido está compuesto por una CPU i7-9700 y GPU GeForce GTX 1080Ti, disponiendo de 16GB de módulo RAM.

El módulo de detección presentará un análisis cuantitativo sobre el *dataset* KITTI de forma que primero se conocerá qué técnica de detección de objetos en espacio tridimensional es la más efectiva.

A continuación, el módulo de detección será integrado junto con ambas técnicas de seguimiento para realizar un análisis cuantitativo sobre las escenas de seguimiento de objetos del conjunto de datos KITTI, conociendo qué método presenta mejores resultados.

Una vez conocida la configuración óptima, se formará un sistema de detección y seguimiento de múltiples objetos (DAMOT), que será evaluado cuantitativamente sobre escenas simuladas mediante CARLA y reales procedentes de la base de datos KITTI y obtenidas sobre el vehículo Techs4AgeCar, mostrando su resultado sobre el propio sensor LIDAR y una cámara RGB en diferentes videos demostrativos.

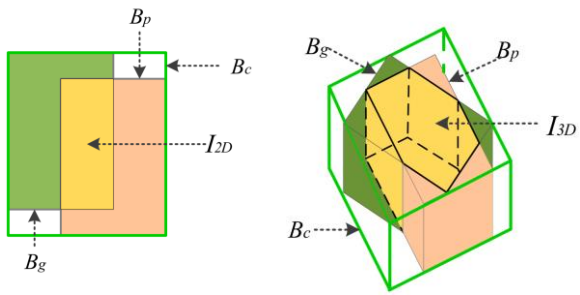
5.1 Análisis cuantitativo del detector de objetos

Con el objetivo de conocer la eficiencia de las redes neuronales detectoras de objetos a partir de nubes de puntos se establece un análisis de su precisión sobre el subconjunto *object detection test* de la base de datos KITTI no empleado durante el entrenamiento, pudiendo establecer comparativas sobre un banco de pruebas estandarizado. Esta base de datos permite realizar un entrenamiento y su posterior evaluación sobre las clases coche, peatón y ciclista empleando 7481 nubes de puntos de entrenamiento y 7518 de prueba.

5.1.1 Métricas de validación de algoritmos de detección

Con el objetivo de establecer una comparativa entre los diferentes métodos empleados para un mismo fin, en este caso la detección de objetos, se deben emplear las mismas métricas aplicadas sobre un mismo conjunto de datos. A continuación, se detallan los parámetros más comunes para la medición del rendimiento de los diferentes algoritmos de detección.

Intersection over Union (IoU): define el área de coincidencia entre la salida proporcionada por el algoritmo y la esperada, con respecto al área de la unión de ambas salidas. Puede ser extrapolado al ámbito tridimensional, computando la intersección de volúmenes en lugar de superficies. La Ilustración 32 recoge ambas opciones, donde en naranja (B_p) se muestra la zona predicha por la red, en verde (B_g) la zona esperada según los datos etiquetados, y en amarillo (I_{2D}, I_{3D}) la intersección de ambas.



a) 2D IoU *b) 3D IoU*

Ilustración 32 Intersección sobre la Unión (IoU) 2D y 3D

Para tener en cuenta el nivel de confianza en la detección (*score*) se define otra serie de valores como:

- *True Positive* (TP): si la confianza en la detección supera el umbral de confianza y la IoU supera el umbral de intersección, se considera correcto.
- *False Positive* (FP): si la confianza supera el umbral de confianza, pero el solape entre la detección y la salida esperada es demasiado bajo, se considera que el método cree que hay un objeto donde realmente no lo hay.
- *False Negative* (FN): es el número de objetos etiquetados por la base de datos que no han sido reconocidos por el algoritmo.

En función de estos parámetros se pueden calcular los términos de precisión y recuperación (*Precision y Recall*) y a partir de ellos obtener las curvas *Precision-Recall*, que muestran la exactitud en cada predicción y la capacidad de detectar todos los objetos esperados en función del umbral de confianza de la detección.

- *Precision*: se calcula como la división del número de objetos correctamente detectados entre todos los objetos que la red ha detectado, ya estén realmente presentes o no. Permite conocer la precisión en la detección.

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

- *Recall* : es el cálculo de los objetos correctamente detectados, dividido entre la suma total de objetos etiquetados en la base de datos. Permite conocer qué parte de los objetos presentes en la escena han sido detectados.

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

La métrica estándar en detección de objetos sobre KITTI es la precisión media (*Average Precision, AP*) sobre el conjunto de los objetos a detectar. Para ello se realizan inferencias sobre el conjunto de nubes de puntos aportadas por el *dataset*, calculando una curva *Precision-Recall* para evaluar la exactitud de la detección realizada para cada objeto con respecto al porcentaje de objetos que es capaz de recuperar. El área bajo la curva realizada es la métrica AP, permitiendo conocer con un único valor cuántos objetos es capaz de detectar y con qué precisión.

5.1.2 Comparativa de resultados

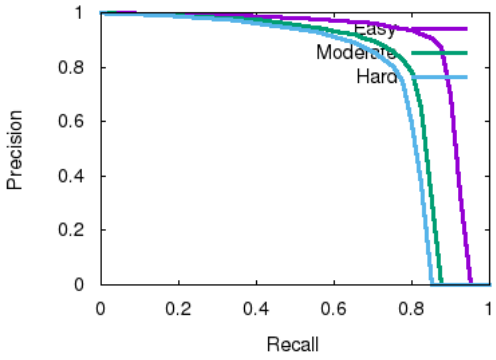
Aplicando las herramientas proporcionadas por el paquete de desarrollo de KITTI se pueden obtener las curvas *Precision-Recall*, así como los valores AP para el conjunto de escenas de detección de la base de datos, pudiendo establecer una comparativa fiable entre las diversas redes neuronales empleadas para detección de objetos sobre las mismas situaciones.

Como se puede observar en las ilustraciones Ilustración 33, Ilustración 34 y Ilustración 35, el conjunto de curvas Precision-Recall para todas las redes validadas, evaluando las detecciones de las clases coche, peatón y ciclista consecutivamente, presentan un elevado índice de precisión en sus detecciones, así como un porcentaje alto de recuperación de objetos.

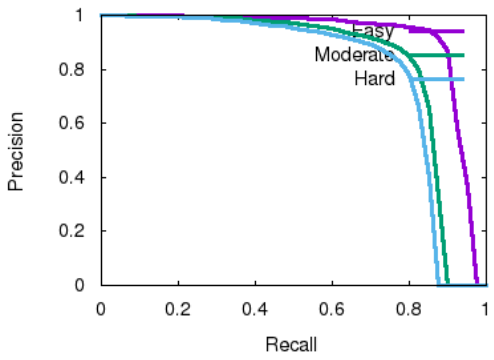
Concretamente, la Ilustración 33 muestra que los resultados alcanzados para la detección de otros coches alcanzan valores muy elevados tanto de precisión en su detección como en porcentaje de elementos recuperados.

Por otra parte, la Ilustración 34 infiere que la detección de peatones presenta curvas más deficientes, debido en gran parte a la escasez de puntos LIDAR que impactan sobre ellos por sus reducidas dimensiones. En esta clase, la red Part-A^2 es la más eficiente.

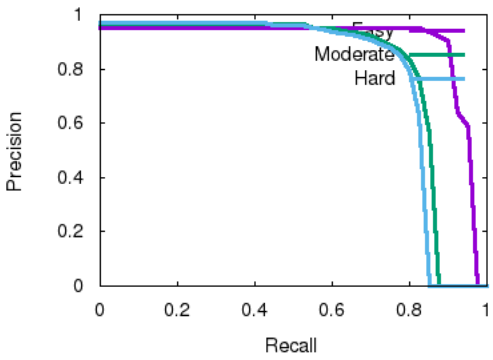
Por último, la detección de ciclistas, representada en la Ilustración 35, resulta en un desempeño intermedio entre las dos clases anteriores, obteniendo mejores resultados en las redes que procesan la nube de puntos sin voxelización previa (Part-A^2, PV-RCNN y PointRCNN).



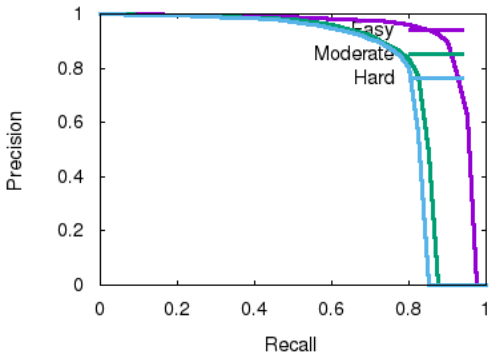
a) PointPillars



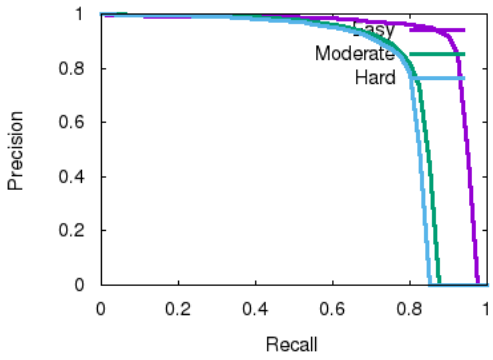
b) SECOND



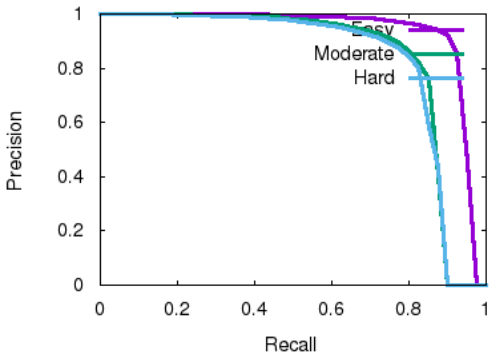
c) PointRCNN



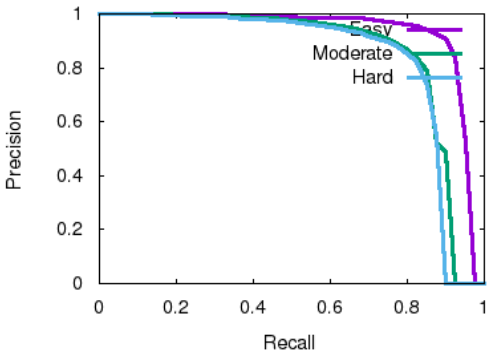
d) PointRCNN-IoU



e) Part-A^2-Free

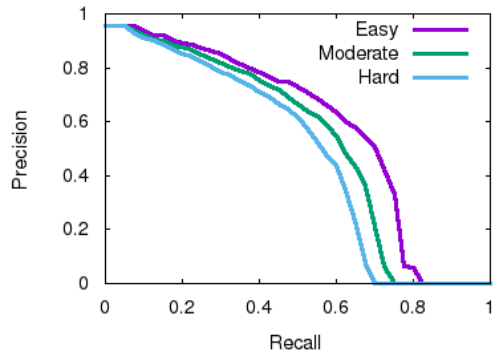


f) Part-A^2-Anchor

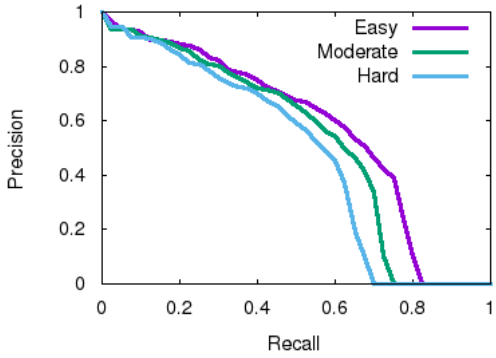


g) PV-RCNN

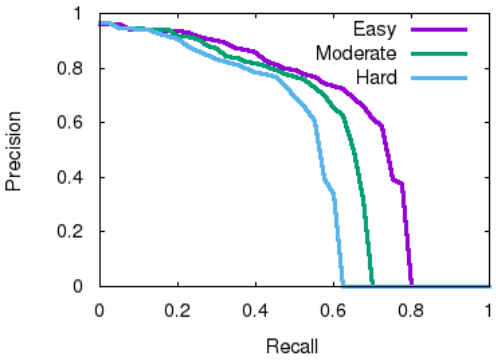
Ilustración 33 Curvas Precision-Recall detectores de objetos 3D para clase coche



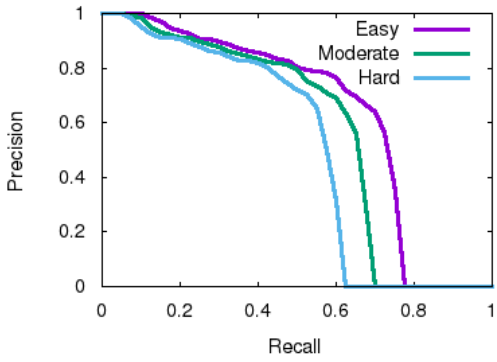
a) PointPillars



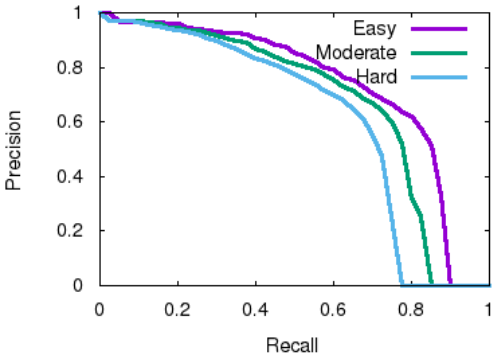
b) SECOND



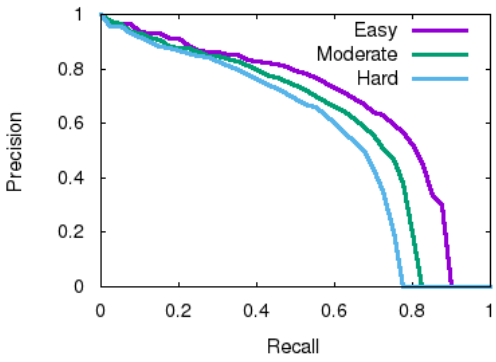
c) PointRCNN



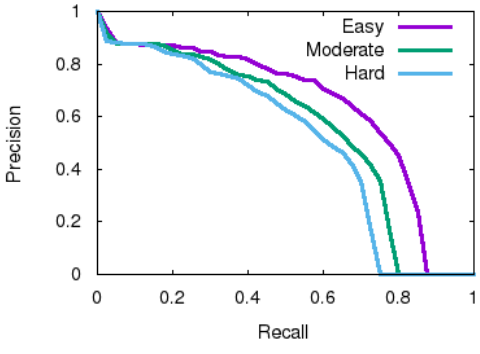
d) PointRCNN-IoU



e) Part-A^2-Free

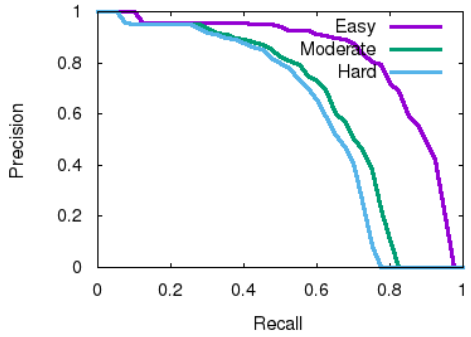


f) Part-A^2-Anchor

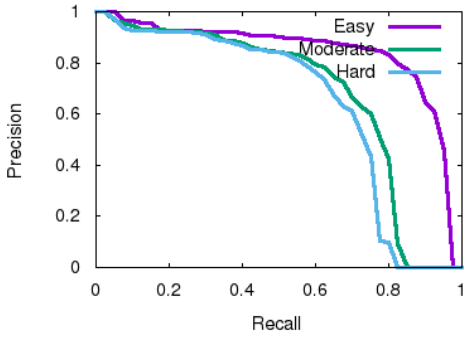


g) PV-RCNN

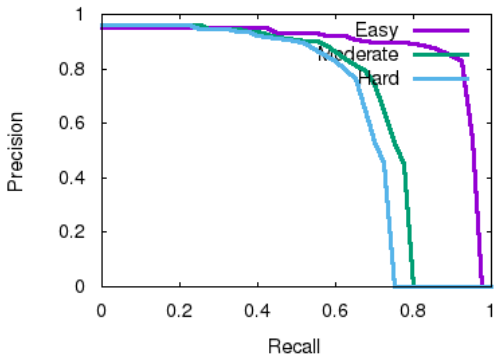
Ilustración 34 Curvas Precision-Recall detectores de objetos 3D para clase peatón



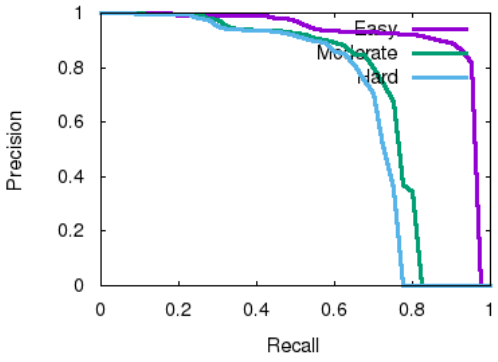
a) PointPillars



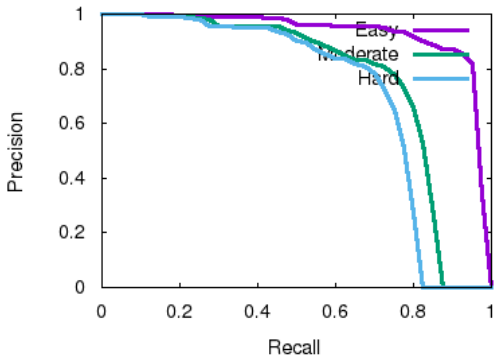
b) SECOND



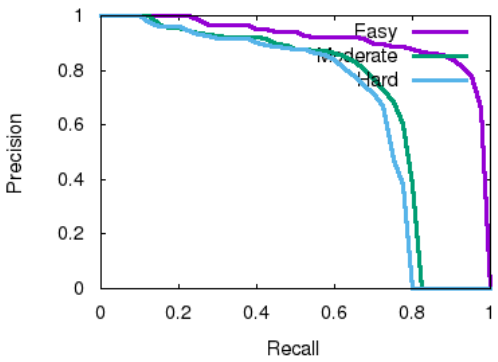
c) PointRCNN



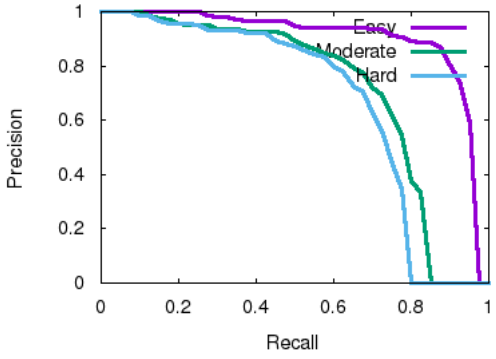
d) PointRCNN-IoU



e) Part-A^2-Free



f) Part-A^2-Anchor



g) PV-RCNN

Ilustración 35 Curvas Precision-Recall detectores de objetos 3D para clase ciclista

Con el objetivo de establecer un análisis más preciso se comprueban los valores numéricos obtenidos por las distintas opciones en las tres clases detectadas.

Método	**Entrada de información**	**Frecuencia** (Hz)	**Coche** (AP)	**Peatón** (AP)	**Ciclista** (AP)
PointPillars [12]	Voxelización	**41.7**	**86.46**	**57.75**	**80.05**
SECOND [19]		19.8	88.61	56.55	80.59
PointRCNN [14]	Sin discretizar	6.3	88.94	61.89	85.01
PointRCNN-IoU [14]		6.3	89.01	62.69	87.48
Part-A^2-Free [20]		5.6	89.12	**70.31**	**87.65**
Part-A^2-Anchor [20]		7.5	**89.56**	65.69	85.50
PV-RCNN [21]	Ambas	4.6	89.35	63.12	86.06

Tabla 5 Comparativa de detectores de objetos sobre dataset KITTI

Como se puede observar en la Tabla 5, las redes neuronales basadas total o parcialmente en análisis de nube de puntos sin preprocesamiento presentan un mayor *Average Precision* que aquellas que realizan una discretización de la información. Sin embargo, el tratamiento de la nube de puntos en su conjunto hace que la red neuronal presente un mayor coste computacional, operando a frecuencias mucho más bajas que las redes con procesamiento.

En el campo de la navegación autónoma se considera que los sistemas en tiempo real deben procesar información a una tasa superior a los 10 Hz, frecuencia común en los sensores de percepción, por lo que las redes basadas en voxelización de la nube de puntos cumplen este requisito.

Por todo lo anteriormente expuesto, la red más eficiente aplicando una situación de compromiso entre la frecuencia de funcionamiento y la precisión en la detección es PointPillars [12].

5.2 Análisis cuantitativo del seguidor de objetos

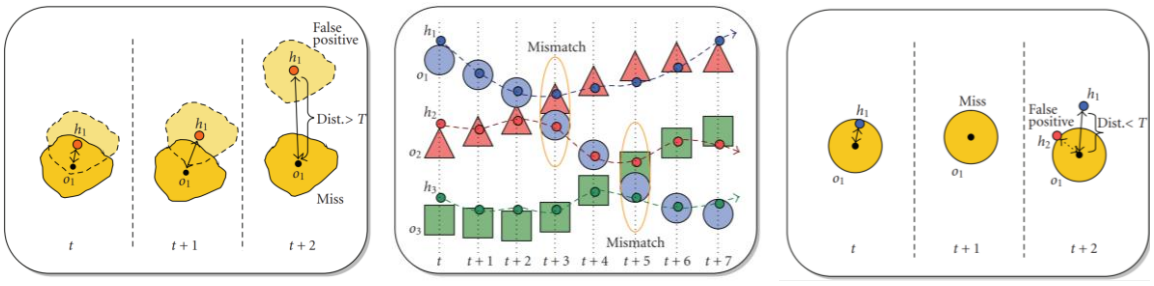
Tomando el detector de objetos PointPillars, se realizan inferencias sobre las 21 escenas que provee el conjunto de datos KITTI en su modalidad de *tracking*, tomando la totalidad de la información del subconjunto *test*.

Para llevar a cabo el análisis se emplea la herramienta 3D MOT propuesta en AB3DMOT [2] de forma que evalúa la coincidencia tridimensional entre el objeto real presente en la escena y el seguido por el algoritmo (3D IoU) en lugar de comprobar su intersección con los datos etiquetados en el plano de la imagen.

5.2.1 Métricas de validación de algoritmos de seguimiento

Los desarrollos de algoritmos de seguimiento de objetos emplean el estándar de métricas CLEAR para MOT [36], estableciendo parámetros de validación específicos para este tipo de algoritmos como *Multi-Object Tracking Precision* (MOTP), *Multi-Object Tracking Accuracy* (MOTA), Falsos Positivos, Falsos Negativos, *Identity Switches* y Fragmentaciones.

Las diferentes variables tienen como objetivo cuantificar el correcto funcionamiento esperado del sistema, esto es, encontrar todos los objetos en el entorno de validación con precisión y mantener una identificación consistente para cada elemento a lo largo de la secuencia.



a) *b)* *c)*

Ilustración 36 Situaciones a evaluar en algoritmos de seguimiento de objetos

La Ilustración 36 trata de reflejar la casuística a analizar durante la evaluación de algoritmos de seguimiento de objetos.

Así, la Ilustración 36 a) indica que, si la distancia entre un objeto y su seguidor supera un cierto umbral, el objeto se declara falso positivo mientras que el seguidor no se vincula a ninguna detección.
El gráfico de la Ilustración 36 b) muestra el suceso de un *Identity Switch* (IDs) o cambio de identidad, ya que el seguidor h_2 se vincula inicialmente con el objeto o_2, pero en el instante t+3, al producirse un cruce con el objeto o_1, el seguidor intercambia su vinculación a este objeto, ocurriendo un *mismatch* o discordancia, así como en el instante t+5 con el elemento o3.
Finalmente, la Ilustración 36 c) expresa una situación en la que un seguidor h_1 es vinculado con el objeto o_1 en el instante inicial, declarándose *miss* o perdido en t+1. En el instante final el objeto puede ser vinculado con su identificador inicial (correcto) o con otro seguidor que se encuentre más cercano (incorrecto).

Las métricas más importantes son aquellas que evalúan el error en el posicionamiento y el error en el seguimiento, MOTP y MOTA respectivamente.

MOTP: Cuantifica el error total en la estimación de la posición para los seguidores vinculados a objetos para toda la secuencia, normalizado por el número de vinculaciones detección – seguidor.

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{3}$$

MOTA: Evalúa los errores cometidos por el algoritmo de seguimiento teniendo en cuenta las pérdidas, los falsos positivos y las discordancias, conociendo la capacidad del algoritmo de mantener una identidad para cada objeto con independencia de la precisión en la localización de los mismos.

$$MOTA = 1 - \frac{\sum_t (misses_t + FP_t + mismatching_t)}{\sum_t g_t} \tag{4}$$

Estas métricas tienen en cuenta el desempeño del algoritmo para un valor de confianza (*confidence score*) fijo, por lo que la consecución de un funcionamiento óptimo requiere de un ajuste manual del umbral sobre ese valor. Con el objetivo de analizar el sistema según el parámetro de confianza, AB3DMOT introduce métricas

integrales, extendiendo MOTA y MOTP en función del valor de seguridad en la detección, realizando una pseudo-integración de la curva *Precision-Recall* obtenida.

AMOTA (Average MOTA): Calcula la media de los valores MOTA para distintos valores de recuperación. Para ello computa el número de pérdidas, falsos positivos y confusiones según el valor de recuperación. L representa el número de valores de confianza escogidos para integración, empleando cuarenta valores entre 0% y 100%.

$$AMOTA = \frac{1}{L} \sum_{r \in \left\{\frac{1}{L}, \frac{2}{L}, \ldots, 1\right\}} \left(1 - \frac{misses_r + FP_r + mismatching_r}{g_t}\right) \quad (5)$$

AMOTP (Average MOTP): Calcula la media de los valores MOTP para distintos valores de recuperación. Para ello integra el MOTP según el valor de recuperación. L representa el número de valores de confianza escogidos para integración, empleando cuarenta valores entre 0% y 100%.

$$AMOTP = \frac{1}{L} \sum_{r \in \left\{\frac{1}{L}, \frac{2}{L}, \ldots, 1\right\}} \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \quad (6)$$

5.2.2 Comparativa de resultados

La herramienta 3D MOT realiza un análisis sobre las escenas KITTI tracking obteniendo tanto métricas CLEAR [36] tradicionales como las nuevas métricas integrales propuestas por AB3DMOT [2].

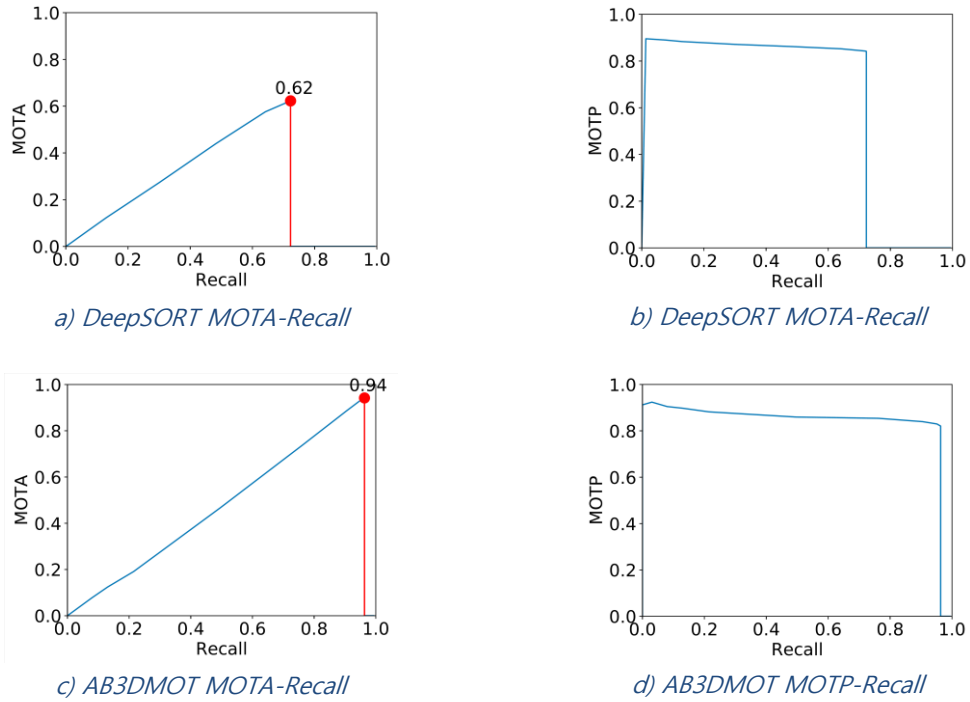
Los métodos analizados son los explicados en el punto 2.4 Seguimiento de objetos mediante técnicas tradicionales, esto es, DeepSORT modificado para extracción de características de la imagen a partir de la proyección de detección LIDAR sobre imagen, y AB3DMOT modificado para funcionamiento en tiempo real mediante mensajes ROS.

Método	AMOTA (%)	AMOTP (%)	MOTA (%)	MOTP (%)	IDs
DeepSORT [17]	24.89	63.08	62.27	**84.13**	843
AB3DMOT [2]	**39.90**	**79.31**	**94.20**	82.06	**150**

Tabla 6 Comparativa de seguidores de objetos sobre dataset KITTI sobre clase coche

Como refleja la Tabla 6 el algoritmo de seguimiento DeepSORT presenta peores resultados que AB3DMOT. Uno de los principales factores que pueden afectar a este rendimiento es el hecho de que DeepSORT realiza un seguimiento sobre variables en imagen bidimensional, produciendo oclusiones y distorsionando las verdaderas magnitudes, mientras que AB3DMOT emplea filtros de Kalman y asociación en el espacio tridimensional.

Además, el hecho de que DeepSORT extraiga información visual de objetos posiblemente ocluidos debido a la diferencia de altura entre el detector LIDAR y la cámara puede producir confusión al asociar mediante características visuales.
Por tanto, el algoritmo de seguimiento de objetos óptimo es AB3DMOT [2].



a) DeepSORT MOTA-Recall *b) DeepSORT MOTA-Recall*

c) AB3DMOT MOTA-Recall *d) AB3DMOT MOTP-Recall*

Ilustración 37 Gráficas MOTA y MOTP vs Recall

La Ilustración 37 muestra la variación de los resultados obtenidos para las métricas CLEAR MOT tradicionales en función del umbral de detección establecido, reflejando la importancia de la introducción de las nuevas métricas integrales AMOTA y AMOTP, así como la capacidad de encontrar el punto óptimo de funcionamiento.

5.2.3 Estudio de mejora sobre AB3DMOT

Una vez decidida la implantación del algoritmo AB3DMOT para el seguimiento de objetos en la escena se debe realizar un estudio de ablación que permita, en base a la modificación de sus parámetros, obtener el mejor resultado.
Para ello, este análisis procesa las escenas de la base de datos KITTI usando el detector PointPillars aplicando y variaciones sobre la configuración del algoritmo de seguimiento.

- Age_{max}: duración máxima en *frames* de un seguidor (filtro de Kalman) sin asociar
- Min_hits: número de instantes que un objeto y un nuevo seguidor tentativo deben coincidir para declarar al seguidor.
- IoU_threshold: umbral de coincidencia entre detección y seguidor para considerar válido el emparejamiento.

Sobre la configuración por defecto (Age_{max} = 1, min_hits = 1, IoU_thr = 0.1) se realizan diversas variaciones de cada uno de los parámetros, estudiando su resultado en métricas estándar.

Age_{max}	**min_hits**	**IoU_thr**	**AMOTA (%)**	**AMOTP (%)**	**MOTA (%)**	**MOTP (%)**	**IDs**
1	1	0.1	**39.90**	79.31	94.20	82.06	150
1	1	0.01	39.84	70.96	95.13	81.84	54
1	1	0.25	39.37	**79.35**	89.10	82.42	682
1	**3**	**0.1**	**39.54**	**71.24**	**91.38**	**83.23**	**2**
1	5	0.1	39.26	71.36	88.84	**83.68**	3
2	1	0.1	39.49	79.24	94.91	81.48	154
3	1	0.1	39.50	79.15	**95.16**	81.15	152

Tabla 7 Comparativa AB3DMOT estudio de mejora sobre clase coche

Según los datos ofrecidos en la Tabla 7, una disminución del umbral de intersección hasta 0.01% supone prácticamente los mismos valores MOTA y MOTP, reduciendo a un 36% los intercambios de identificadores entre objetos, lo que mejora considerablemente el seguimiento inequívoco de estos. Por otra parte, un ligero

aumento del número mínimo de coincidencias entre detección y seguidor para declarar al seguidor como válido (min_hits) permite reducir los intercambios de identificadores a valores prácticamente nulos, evitando el principal problema del seguimiento de objetos. La modificación de la duración máxima de los identificadores sin asociar (Age_{max}) apenas tiene efectos sobre las diferentes métricas estudiadas.

En resumen, la configuración óptima del seguidor de objetos AB3DMOT a la vista del análisis realizado se obtiene empleando los valores:

$$Age_{max} = 1,\ min\_hits = 3,\ IoU\_thr = 0.1$$

5.3 Análisis cuantitativo del sistema embebido

La aplicación óptima, entendida como el sistema de seguimiento y detección de objetos con menor latencia y mejores resultados, es instalada sobre el sistema embebido NVIDIA Jetson AGX Xavier. La arquitectura de este dispositivo está basada en el procesador ARM, por lo que el software y las librerías empleadas deben adaptarse al nuevo ecosistema. Para ello, el kit de desarrollo proporciona tanto la imagen de Ubuntu, el sistema operativo anfitrión y diversos contenedores Docker según el software a utilizar, basados en el sistema de soporte L4T.

En este caso, el sistema embebido empleado utiliza Ubuntu 18.04, sobre el que se instala un contenedor Docker con preinstalación de PyTorch 1.6 y Python3, necesarios para acelerar el funcionamiento del módulo de detección. Además, para la recepción y publicación de resultados se instala el *framework* ROS Melodic, de forma que el sistema de detección y seguimiento puede recibir nubes de puntos procedentes del sensor LIDAR mediante mensajes PointCloud2, devolviendo una lista de objetos detectados y seguidos en el entorno.

Módulo	**Frecuencia AGX Xavier**	**Frecuencia PC**	**Incremento**
Detección	7.3 Hz	41.7 Hz	5.7x
Seguimiento	15 Hz	100 Hz	6.7x

Tabla 8 Comparativa de rendimiento entre sistema embebido y ordenador

Siguiendo los datos proporcionados en la Tabla 8 encontramos que en ambos equipos el cuello de botella se encuentra en la velocidad de inferencia de la red neuronal detectora de objetos, pues es la que requiere de un mayor nivel de cómputo procedente de la unidad de procesamiento gráfico, mientras que el algoritmo de seguimiento se aplica sobre la CPU.
Sin embargo, la diferencia de velocidad de procesamiento es mayor en el caso del módulo de seguimiento ya que el sistema embebido presenta una GPU de altas prestaciones mientras que su CPU tiene unas capacidades menores. En ambos casos la diferencia de velocidad no llega a septuplicarse, mientras que la potencia máxima del sistema embebido es ocho veces menor que la consumida por la tarjeta gráfica integrada en el ordenador sobremesa, confirmando que la eficiencia del sistema embebido es, a escala, superior a la proporcionada por un ordenador convencional.

5.4 Análisis cualitativo sobre base de datos KITTI

Previo a la implementación del sistema sobre el vehículo Techs4AgeCar se analiza cualitativamente su funcionamiento sobre la información proveniente de un sensor LIDAR real, ubicado también en la parte superior del vehículo.

La escena analizada, correspondiente al subconjunto de seguimiento de objetos, escena 0001, desarrolla un recorrido en L a lo largo de dos calles, con la mayoría de los vehículos estacionados en los laterales a lo largo de 443 instantes (44.3 segundos) sincronizando información proveniente de cámara RGB frontal con la nube de puntos correspondiente.

La Ilustración 38 muestra la consistencia en la identificación de diversos objetos a lo largo de 20 *frames* consecutivos (2 segundos) en un entorno complejo sobre la imagen RGB frontal. El objetivo de esta proyección es poder realizar un mejor entendimiento de la escena, sin realizar ningún tipo de procesamiento sobre imagen para realizar detección o seguimiento de objetos.

Por otra parte, la Ilustración 39 proporciona una representación de la información obtenida por el sensor LIDAR en vista de pájaro para esos mismos instantes de detección, junto con diversos marcadores tetraédricos representando a los vehículos seguidos en escena para los mismos instantes.

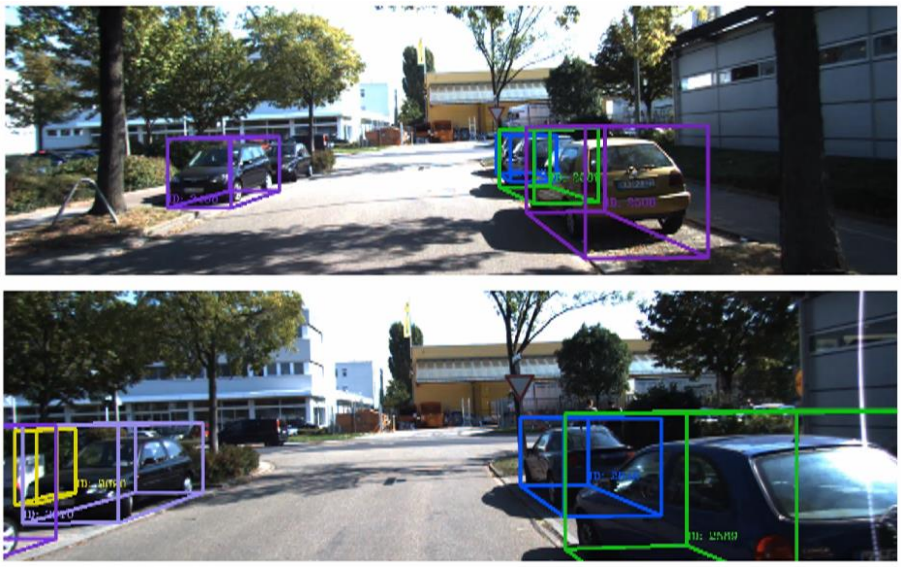


Ilustración 38 Detección y seguimiento en dataset KITTI en t, t+20 frames. Sensor RGB

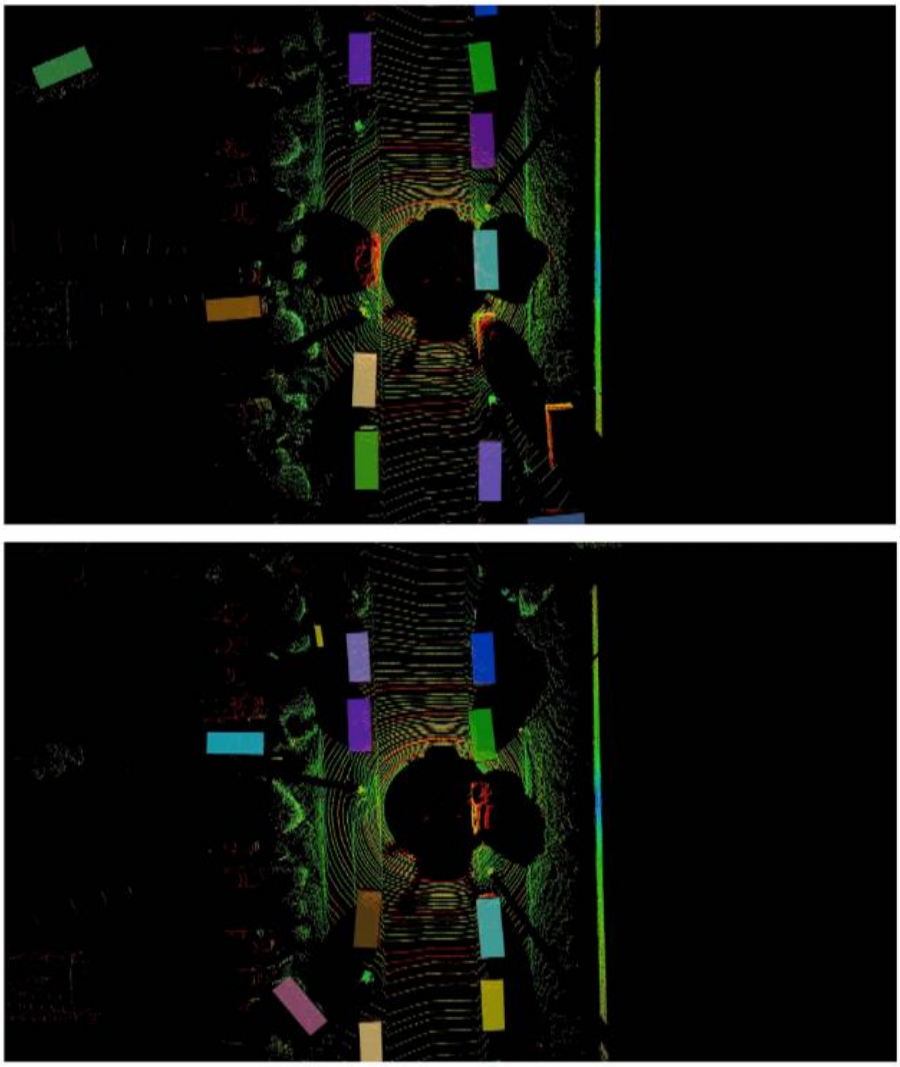


Ilustración 39 Detección y seguimiento en dataset KITTI en t, t+20 frames. Sensor LIDAR

Con el objetivo de permitir una observación en profundidad del desempeño del sistema sobre la escena se proporciona un video mostrando el funcionamiento sobre el sensor RGB y LIDAR utilizados por la base de datos KITTI en el siguiente enlace: https://youtu.be/TcP7sa0mUWI.

5.5 Análisis cualitativo sobre simulador CARLA

El simulador de conducción autónoma CARLA permite obtener escenas hiperrealistas empleando diversos sensores estándar en el sector, como cámaras RGB y LIDAR.

Gracias al uso de un simulador, la cámara puede ubicarse en cualquier lugar del espacio, de forma que en este caso permite obtener una visualización en vista de pájaro de la escena para ubicar los vehículos detectados y seguidos. La configuración escogida para el sensor cámara RGB se muestra en la Tabla 9, obteniendo una perspectiva en vista de pájaro.

Parámetro	**Valor**
Posición	[0, 0, 20] m
Orientación	[90, -90, 0] º
Tamaño	1280 x 720 px
Apertura	90º

Tabla 9 Ajustes de cámara RGB en simulador CARLA

Además, se implantará un sensor LIDAR de 64 haces en la parte superior del vehículo de forma similar al empleado en la base de datos KITTI, a partir del cual se detectarán los objetos en el entorno del coche. La configuración empleada sobre el sensor LIDAR para el análisis cualitativo realizado se muestra en la Tabla 10.

Parámetro	**Valor**
Posición	[0, 0, 2.5] m
Rango	100 m
Haces	64
Puntos por segundo	640.000
Apertura horizontal	2º / -26º

Tabla 10 Ajustes de sensor LIDAR en simulador CARLA

Debido a las altas prestaciones del sensor LIDAR implantado y al realismo de este, la detección se lleva a cabo con éxito de forma similar a la realizada sobre la información obtenida en el mundo real.

Con el objetivo de demostrar la validez del sistema de detección y seguimiento de objetos implementado en este TFM se diseñan tres escenarios de prueba en el simulador, modificando tanto los objetos en el entorno como las condiciones climatológicas. En cada uno de ellos se estudiará la capacidad de mantener los identificadores asociados a los objetos, mostrando su funcionamiento sobre una representación de la información obtenida mediante el sensor LIDAR, así como de una cámara RGB ubicada en vista de pájaro.

5.5.1 Escenario 1. Rotonda en día lluvioso

El primero de los escenarios representa un ambiente urbano con densidad de tráfico baja circulando en una rotonda, con una climatología lluviosa.

En este caso, la Ilustración 40 muestra en vista de pájaro la imagen a color para un instante determinado, junto con la visión de los mismos sensores 20 *frames* después. Por otro lado, la Ilustración 41 proporciona la representación de la nube de puntos para esos mismos momentos.

Se puede observar cómo se mantiene la identificación del objeto magenta, el único que permanece dentro del campo de visión de los señores, entre los distintos momentos, tanto en el sensor RGB, donde se muestra un tetraedro que contiene al vehículo, como en la información LIDAR, superponiendo cajas tridimensionales a los puntos que impactan con el vehículo.

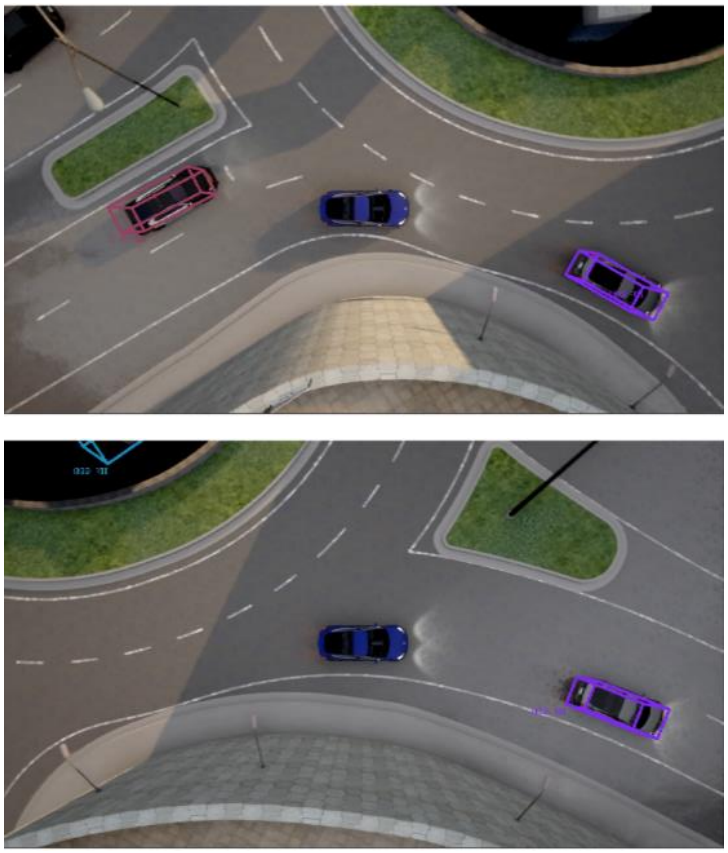


Ilustración 40 Detección y seguimiento en CARLA en t, t+20. Escenario 1. Sensor RGB

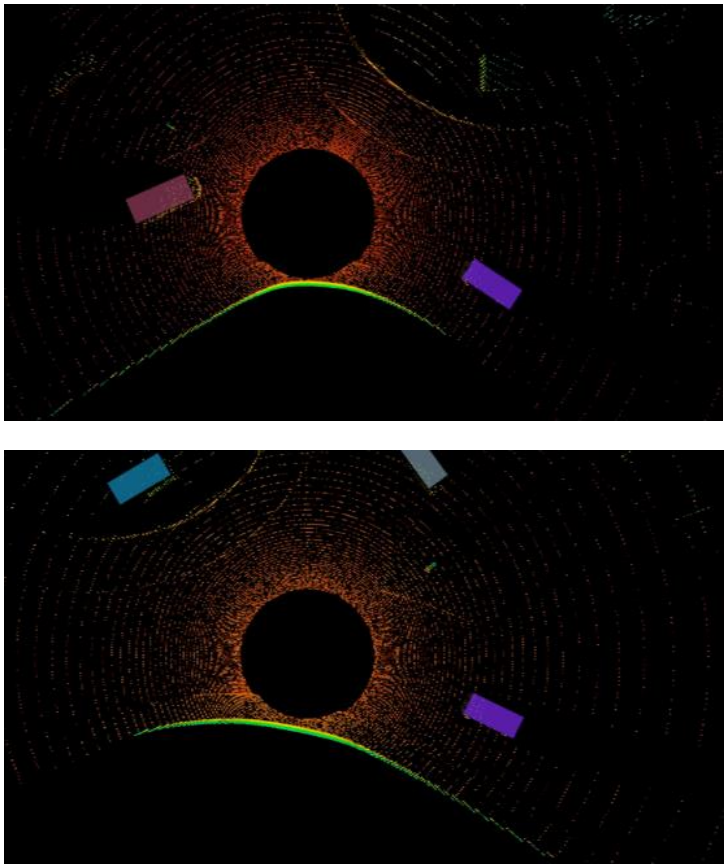


Ilustración 41 Detección y seguimiento en CARLA en t, t+20. Escenario 1. Sensor LIDAR

5.5.2 Escenario 2. Calle nocturna con aparcamientos laterales

En este escenario se reproduce una situación cotidiana de un vehículo circulando por una calle estrecha con multitud de vehículos aparcados en los laterales. Además, las condiciones de luminosidad son muy reducidas, siendo una escena nocturna, destacando la ventaja del uso de un sensor LIDAR frente a otros sistemas pasivos.

Como se puede observar en la Ilustración 42, el sistema es capaz de detectar la mayor parte de los objetos de su entorno, siendo alguno de los no detectados camiones o furgonetas de grandes dimensiones para los que el sistema de detección no ha sido entrenado. También se puede apreciar, tanto en esta, como en la Ilustración 43, el éxito del método en el seguimiento de los objetos mientras el vehículo principal se mueve entre ellos, pues consigue mantener la mayoría de los objetos en escena.

El elevado nivel de desempeño del sistema en esta situación es debido al empleo de un sensor activo tipo LIDAR para el reconocimiento del entorno, de forma que es el propio dispositivo el que emite luz para poder extraer información, mientras que otros sistemas basados en sensores pasivos, como por ejemplo cámaras RGB, apenas percibirían los objetos a su alrededor debido a la baja luminosidad de la escena.

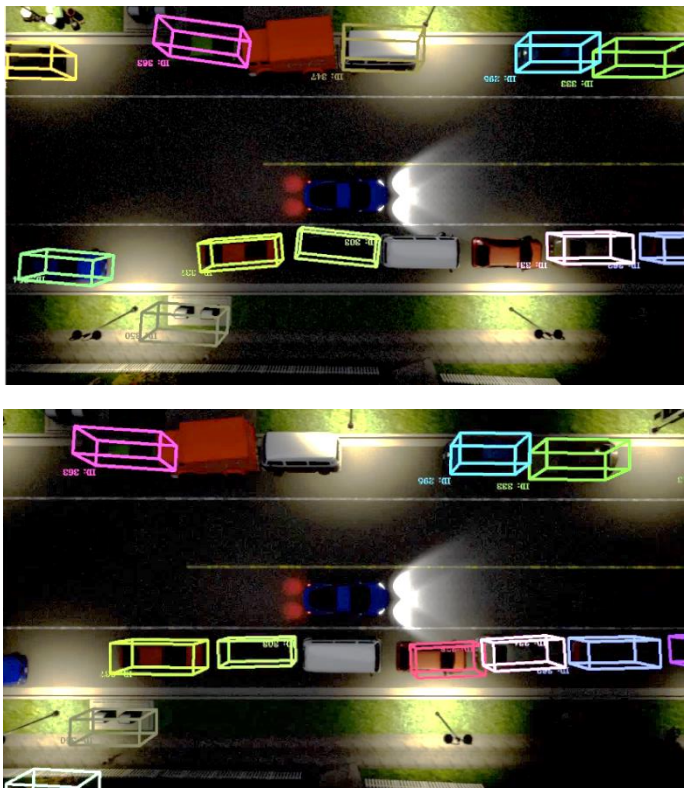


Ilustración 42 Detección y seguimiento en CARLA en t, t+20. Escenario 2. Sensor RGB

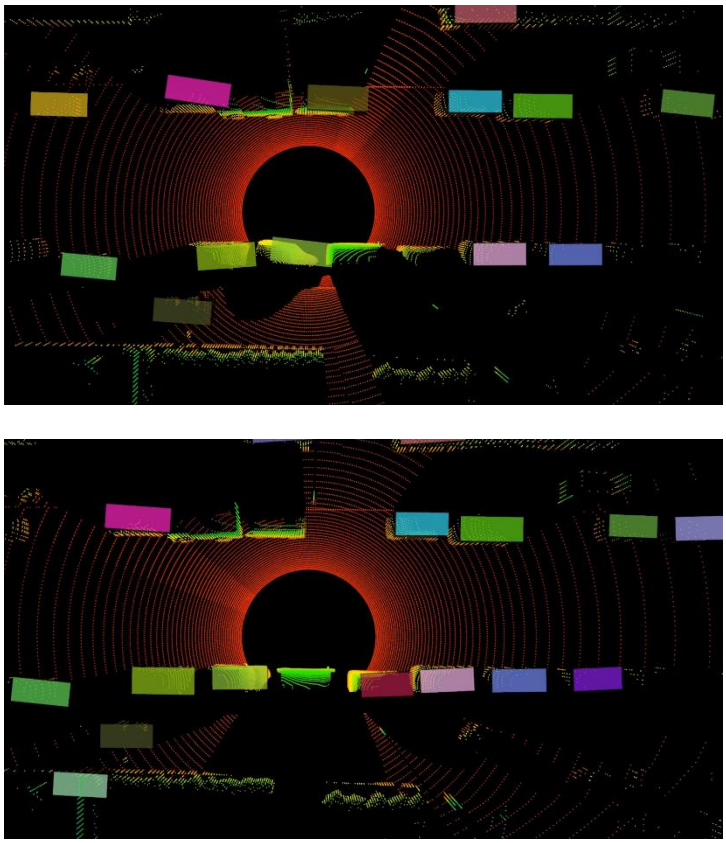


Ilustración 43 Detección y seguimiento en CARLA en t, t+20. Escenario 2. Sensor LIDAR

5.5.3 Escenario 3. Giro en cruce con ciclista espontáneo

Este último escenario es uno de los propuestos por el simulador CARLA para evaluar el comportamiento de los vehículos autónomos en las tipologías señaladas por el organismo NHTSA norteamericano [37] como una de las más frecuentes en los accidentes urbanos. En la escena, el vehículo pretende realizar un giro a la derecha en un cruce, cuando aparece de repente un ciclista en una zona no prevista para su paso, evaluando si el coche es capaz de reaccionar ante este evento evitando el choque.

Como se puede apreciar, tanto en la Ilustración 44 como en la Ilustración 45, el sistema es capaz de detectar al ciclista antes de producir el choque, encuadrándolo en un tetraedro vertical. Además, puede observarse cómo el sistema también está reconociendo un vehículo en la parte inferior derecha de la imagen, más alejado.

Ilustración 44 Detección y seguimiento en CARLA en t, t+20. Escenario 3. Sensor RGB

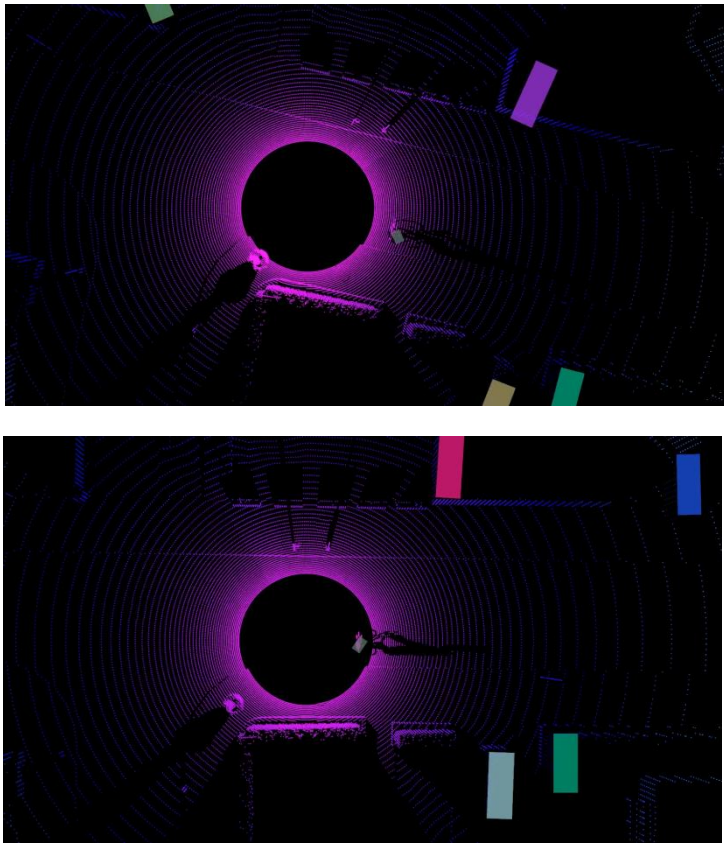


Ilustración 45 Detección y seguimiento en CARLA en t, t+20. Escenario 3. Sensor LIDAR

5.5.4 Conclusiones

Según los resultados extraídos en las simulaciones realizadas, el método de detección y seguimiento de objetos desarrollado en este TFM es válido para ser implementado en vehículos autónomos, resultando un correcto funcionamiento tanto en situaciones de lluvia, escasa luminosidad o eventos imprevistos en carretera.

Con el objetivo de permitir una observación en profundidad del desempeño del sistema sobre la escena se proporciona un video mostrando el funcionamiento sobre el sensor RGB y LIDAR en el simulador CARLA en el siguiente enlace: https://youtu.be/t5Dp4fbGUAw.

5.6 Análisis cualitativo sobre vehículo Techs4AgeCar

El objetivo final del presente Trabajo Fin de Máster es la implementación del sistema de detección y seguimiento de objetos sobre el vehículo Techs4AgeCar del grupo RobeSafe. Para ello, el vehículo dispone de un sensor LIDAR VLP-16, que será empleado para detección de objetos en todo el entorno, así como de una cámara estéreo frontal, donde se proyectarán los elementos seguidos para facilitar la comprensión de la escena. El hardware sobre el que se implementará el método desarrollado es un sistema embebido Nvidia Jetson AGX Xavier.

Ilustración 46 Detección y seguimiento en Techs4AgeCar en t, t+20 frames. Sensor RGB

La Ilustración 46 muestra la consistencia en la identificación de diversos objetos a lo largo de 20 *frames* consecutivos (2 segundos) en un entorno más complejo sobre la imagen RGB frontal, con más vehículos y oclusiones que en el análisis sobre simulación.

Además, el sensor LIDAR que porta el vehículo Techs4AgeCar, mostrado en la Ilustración 47, dispone de tan solo 16 haces, lo que reduce la precisión de la información con respecto a los usados en CARLA y KITTI con 64 haces, dificultando la detección de los objetos del entorno. Aun así, se puede ver cómo mantiene coherentemente las identificaciones de los objetos a su alrededor.

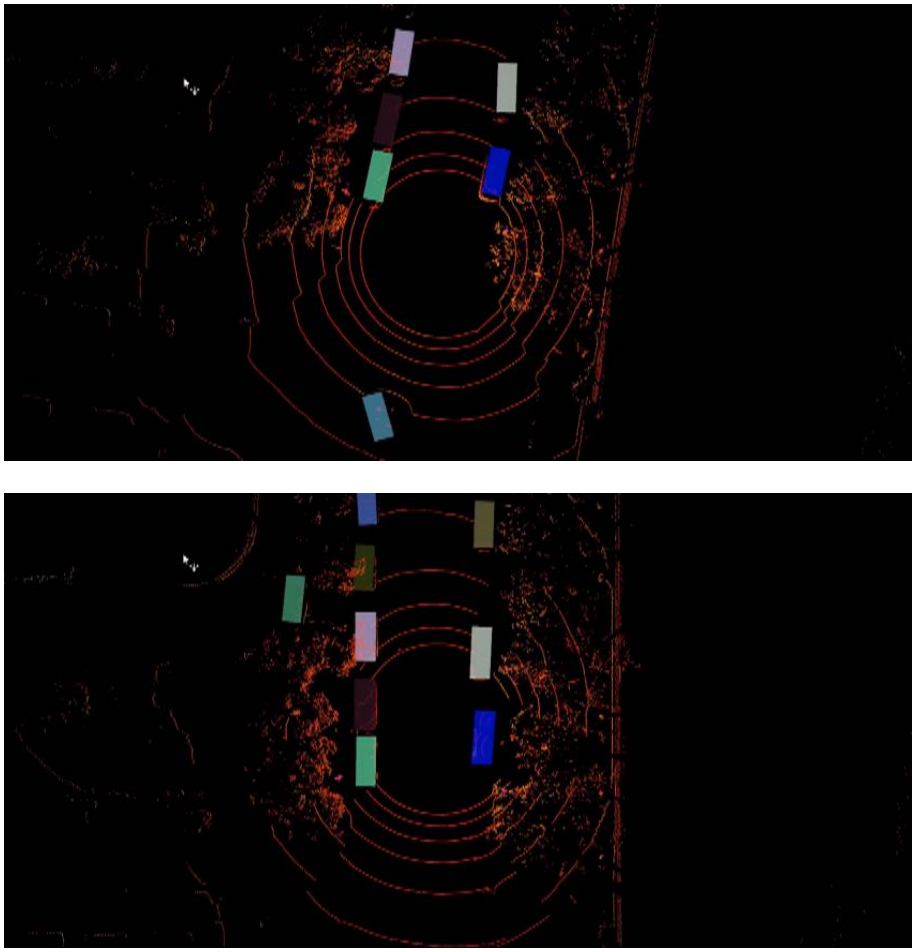


Ilustración 47 Detección y seguimiento en Techs4AgeCar en t, t+20 frames. Sensor LIDAR

Con el objetivo de permitir una observación en profundidad del desempeño del sistema sobre la escena se proporciona un video mostrando el funcionamiento sobre el sensor RGB y LIDAR proporcionados por la base de datos KITTI en el siguiente enlace: https://youtu.be/pt-L9Epl0Cw.

Capítulo 6.
Conclusiones generales y trabajos futuros.

El desarrollo del presente Trabajo Fin de Máster ha permitido la implementación de un sistema de procesamiento sensorial con el objetivo de identificar obstáculos en el entorno de un vehículo autónomo. El método desarrollado emplea la información extraída por un sensor LIDAR integrado en el vehículo para identificar otros vehículos, peatones y ciclistas alrededor del mismo, trazando su trayectoria a través de su recorrido en la escena. Su integración en un sistema embebido permite una fácil aplicación sobre el vehículo autónomo en desarrollo Techs4AgeCar.

A lo largo del desarrollo del proyecto se han justificado las actuaciones realizadas mediante diversos análisis cuantitativos y cualitativos que ilustran el rendimiento de los sistemas que lo forman.

El campo de la navegación autónoma en vehículos terrestres está en constante evolución. Por ende, las capacidades de los sistemas de los que disponemos actualmente se verán pronto superadas por nuevas tecnologías capaces de aportar un procesamiento cada vez mayor, lo que permitirá alcanzar un mayor grado de inteligencia artificial progresivamente.

Es por eso por lo que esta implementación deberá ser actualizada frecuentemente con las técnicas que presente el estado del arte, tanto en el ámbito software, desarrollando nuevos algoritmos de detección y seguimiento de objetos, como en el hardware, a través del uso de nuevos y más precisos sensores junto con unidades de procesamiento más potentes.

Actualmente, el progreso conseguido durante este Trabajo Fin de Máster debe ser coordinado con otras tareas de la conducción autónoma como el control, el planificador o el módulo de toma de decisiones, de forma que el vehículo reaccione ante los obstáculos detectados por este módulo, consiguiendo así una conducción mucho más segura.

Además, es probable que se alcance una mejora sustancial en la detección de objetos en el entorno del vehículo real del proyecto Techs4AgeCar ampliando virtualmente la resolución de su sensor LIDAR de 16 haces a 64 mediante la aplicación del Trabajo Fin de Grado "Aumento de resolución de la nube de puntos generada por un LIDAR mediante técnicas de Deep Learning", presentado recientemente por Bruno Valdebenito en la Escuela Politécnica Superior de la Universidad de Alcalá. Este trabajo implementa una novedosa técnica de hiper-resolución aplicada a nubes de puntos tridimensionales con el objetivo de proporcionar información más rica en detalle a los sistemas de procesamiento, suponiendo un sustancial ahorro económico en la implementación de sensores de mayor coste aplicando técnicas de aprendizaje profundo.

También es ampliable el sistema de seguimiento de objetos implementado, de forma que coordine la extracción de características de una o múltiples cámaras equipadas en el vehículo junto con la información tridimensional para conseguir una identificación más robusta. Actualmente los sistemas de seguimiento de obstáculos trabajan para la integración de algoritmos de predicción avanzada de la posición de los objetos, de forma que además de conocer la trayectoria de los mismos en instantes pasados, el sistema sea capaz de predecir dónde se van a encontrar pasado un tiempo, introduciendo mayor seguridad a la conducción al prever futuros obstáculos en el área de circulación.

Anexo I. Pliego de Condiciones

A continuación, se describe el material necesario para el desarrollo del presente Trabajo Fin de Máster, tanto hardware como software.

Hardware empleado

Para la investigación y el desarrollo de la técnica implementada se ha empleado un ordenador sobremesa con los siguientes componentes:

- Intel Core i7 9770
- NVIDIA 1080Ti
- 16GB RAM DDR4
- 500GB SDD

El sistema completo se ha implementado en un sistema embebido específico:

- Nvidia Jetson AGX Xavier 32 GB

Software empleado

El desarrollo y la implementación se han llevado a cabo sobre Ubuntu 18.04 debido a su flexibilidad de integración y la disponibilidad en ambas plataformas hardware empleadas.

Dentro del sistema operativo se han instalado diversos elementos software:

- Docker
- Python 3, numpy, flask-socketio, eventlet, pillow, h5py, Keras, TensorFlow
- ROS
- CARLA y Unreal Engine

Anexo II. Presupuesto

El presente anexo detalla el presupuesto necesario para llevar a cabo las tareas descritas en este Trabajo Fin de Máster relativo a mano de obra, hardware y software. Tanto los costes de material como impuestos se encuentran actualizados a fecha de octubre de 2020.

Coste Hardware

Los costes de hardware del ordenador se detallan a continuación indicando el coste de los componentes que lo conforman.

Material	Precio unitario	Unidades	Subtotal
Ordenador sobremesa	1.490 €	1	1.490 €
Nvidia 1080Ti	900 €	1	900 €
SSD 500GB	90 €	1	90 €
i7-9700	300 €	1	300 €
Otros componentes	200 €	1	200 €
Nvidia Jetson Xavier AGX	1000 €	1	1.000 €
Costes HW totales			**2.490 €**

Tabla 11 Costes de material hardware

Coste Software

El software no supone un coste para el proyecto puesto que se aplican herramientas libres o proporcionadas por la Universidad de Alcalá.

Software	Subtotal
Ubuntu 18.04	-
CARLA	-
Python	-
Microsoft Office	-
Costes SW	-

Tabla 12 Costes de material software

Coste de personal

El coste de personal se estima con un salario de 20 € brutos por hora trabajada, teniendo en cuenta la duración de cada fase del proyecto, aplicando una jornada de 4 horas diarias.

Fase del proyecto	Meses	Días	Subtotal
Estudio del estado del arte	0,5	11	880 €
Análisis y propuesta de mejora	1	22	1.760 €
Aplicación de mejora	2,5	55	4.400 €
Obtención y análisis de resultados	0,5	11	880 €
Prueba vehículo	1	22	1.760 €
Documentación	1,5	33	2.640 €
Coste de personal			**12.320 €**

Tabla 13 Costes de personal

Coste de ejecución totales

Los costes de ejecución totales incluyen los costes en recursos físicos y humanos.

Costes de ejecución	Subtotal
Costes Hardware	2.490 €
Costes Software	-
Costes de personal	12.320 €
Subtotal final	**14.810 €**

Tabla 14 Costes de ejecución totales

Gastos generales y beneficio industrial

Siguiendo la tipificación del PEM 2020, los gastos generales se estiman en un 13% de los costes de ejecución totales, mientras que el beneficio industrial se sitúa en torno a un 6% de los costes de ejecución totales.

Concepto	Subtotal
Costes de ejecución material	14.810,00 €
Gastos generales (13%)	1.925,30 €
Beneficio industrial (6%)	889,60 €
Presupuesto de ejecución por contrata	17.624,90 €

Tabla 15 Gastos generales, beneficio industrial y presupuesto de ejecución

Importe total del presupuesto

El presupuesto final de este Trabajo Fin de Máster finaliza con la aplicación del IVA, estipulado en 21%, a partir del presupuesto de ejecución por contrata.

Concepto	Subtotal
Presupuesto de ejecución	17.624,90 €
Porcentaje de IVA	3.701,02 €
Importe final	**21.325,92 €**

Tabla 16 Presupuesto total

El presupuesto final estimado para la realización de este Trabajo Fin de Máster asciende a un total de 21.324,92 € (veintiún mil trescientos veinticuatro con noventa y dos euros).

Anexo III. Manual de usuario

En este anexo se detallará el proceso de instalación y ejecución de los diferentes entornos explicados y empleados para la reproducción de los resultados presentes en este Trabajo Fin de Máster.

1. Dataset KITTI

El banco de datos KITTI consiste en un conjunto de información de diversa procedencia, como calibración, cámara, nube de puntos, etc. Que puede ser descargada de la página web oficial http://www.cvlibs.net/datasets/kitti/

Los análisis cuantitativos acometidos, tanto para detección de objetos con la propia herramienta de KITTI, como de seguimiento con la herramienta de AB3DMOT, deben conocer la dirección en la que se encuentra descargada la base de datos en el ordenador para poder iterar sobre ella y estudiar sus resultados.

Para poder realizar el análisis cuantitativo de la escena del dataset KITTI se ha hecho uso del paquete de ros del repositorio https://github.com/iralabdisco/kitti_player, que permite la publicación de los datos almacenados en diversos mensajes de ROS para simular una entrada en tiempo real al sistema desarrollado. Gracias a ello se publican sincronizadamente con una frecuencia establecida la nube de puntos, para realizar detección y seguimiento, junto con la imagen para realizar la proyección de los objetos a plano bidimensional y estudiar su funcionamiento.

2. Simulador CARLA

La versión del simulador CARLA empleada para el análisis del funcionamiento del sistema desarrollado en este Trabajo Fin de Máster es 0.9.10 Pre-release, una versión todavía en desarrollo debido a que la mejora introducida en esta última modificación sobre el sensor LIDAR permite un reconocimiento más preciso de los objetos al adaptarse la nube de puntos a la forma del vehículo de forma correcta, mientras que las versiones anteriores simplificaba la nube de puntos impactando sobre cubos que contenían al vehículo.

En la fecha de desarrollo del trabajo esta versión puede descargarse directamente montada en el siguiente enlace: https://carla-releases.s3.eu-west-3.amazonaws.com/Linux/CARLA_0.9.10-Pre_Ubuntu18.tar.gz

Sin embargo, para instalar otras versiones estables, o esta misma cuando haya terminado su desarrollo, se deben seguir los pasos especificados en esta página web: https://carla.readthedocs.io/en/latest/build_linux/

Ambas opciones deben disponer del motor gráfico Unreal Engine 4 previo a la instalación del simulador. Las instrucciones de instalación se pueden encontrar también en el enlace anterior, además de en la lista de comandos mostrada a continuación.

Además, para poder publicar la información sensorial recogida por el vehículo simulado es necesario instalar el paquete de ROS desarrollado específicamente para ello: https://github.com/carla-simulator/ros-bridge

Comandos para instalación de Unreal Engine 4 y simulador CARLA

#Dependencies

```
sudo apt-get update &&
sudo apt-get install wget software-properties-common &&
sudo add-apt-repository ppa:ubuntu-toolchain-r/test &&
wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key|sudo apt-key add - &&
sudo apt-add-repository "deb http://apt.llvm.org/xenial/ llvm-toolchain-xenial-8
main" &&
sudo apt-get update
sudo apt-get install build-essential clang-8 lld-8 g++-7 cmake ninja-build
libvulkan1 python python-pip python-dev python3-dev python3-pip libpng-dev
libtiff5-dev libjpeg-dev tzdata sed curl unzip autoconf libtool rsync libxml2-dev
libxerces-c-dev &&
pip2 install --user setuptools &&
pip3 install --user -Iv setuptools==47.3.1 &&
pip2 install --user distro &&
pip3 install --user distro
```

#Unreal Engine 4

```
git clone --depth=1 -b 4.24 https://github.com/EpicGames/UnrealEngine.git
~/UnrealEngine_4.24
cd ~/UnrealEngine_4.24
wget
https://carla-releases.s3.eu-west-3.amazonaws.com/Linux/UE_Patch/430667-136367
43-patch.txt ~/430667-13636743-patch.txt patch --strip=4 <
~/430667-13636743-patch.txt
./Setup.sh && ./GenerateProjectFiles.sh && make
cd ~/UnrealEngine_4.24/Engine/Binaries/Linux && ./UE4Editor
```

#CARLA

```
sudo apt-get install aria2
git clone https://github.com/carla-simulator/carla
cd ~/carla
./Update.sh
echo ‘export UE4_ROOT=~/UnrealEngine_4.24’ >> ~/.bashrc
make launch
make PythonAPI
make package
echo ‘export PYTHONPATH=$PYTHONPATH:~/carla/PythonAPI/carla/dist/
carla-<carla_version_and_arch>.egg’ >> ~/.bashrc
```

#CARLA ROS bridge

```
mkdir -p ~/carla-ros-bridge/catkin_ws/src
cd ~/carla-ros-bridge
git clone https://github.com/carla-simulator/ros-bridge.git
cd ros-bridge
git submodule update —init
cd ../catkin_ws/src
ln -s ../../ros-bridge
source /opt/ros/<kinetic or melodic>/setup.bash
cd ..
rosdep update
rosdep install --from-paths src --ignore-src -r
catkin_make
echo ‘source ~/carla-ros-bridge/catkin_ws/devel/setup.bash’ >> ~/.bashrc
```

Comandos de ejecución del simulador CARLA

#Lanzar servidor en Terminal 1

```
cd ~/carla
./CarlaUE4.sh #Puede estar en algún subdirectorio
```

#Lanzar cliente en Terminal 2

```
cd ~/carla/PythonAPI/examples
python manual_control.py
```

#Abrir CARLA ROS bridge Terminal 3

```
roslaunch carla_ros_bridge carla_ros_bridge.launch
```

#Lanzar múltiples vehículos circulando autónomamente

```
cd ~/carla/PythonAPI/examples
python spawn_npc.py
```

#Cambiar ciudad de simulación a Town02 (o cualquier otra)

```
cd ~/carla/PythonAPI/utils
python config.py -m Town02
```

3. OpenPCDet

El *framework* de detección de objetos empleado para el desarrollo de este Trabajo Fin de Máster es OpenPCDet. Presenta una sencilla instalación a través del código modificado en un *fork* propio disponible en GitHub:
https://github.com/JavierEgido/OpenPCDet
Para obtener un funcionamiento suficientemente ágil es necesario disponer de una tarjeta gráfica de alto rendimiento, así como una instalación de drivers acorde a sus características, junto con CUDA.
Además, este repositorio ha sido desarrollado bajo la versión Python 3.6, por lo que es necesario disponer de su correcta instalación en el equipo.

Comandos para instalación de OpenPCDet

```
#PyTorch
python3.6 -m pip install torch==1.5 torchvision

#spconv
git clone https://github.com/traveller59/spconv.git --recursive
sudo apt-get install libboost-all-dev
sudo apt install cmake>=3.13.2
python3.6 setup.py bdist_wheel
cd ./dist
pip install *.whl          #Instalar fichero .whl generado

#OpenPCDet
cd
git clone https://github.com/muzi2045/OpenPCDet.git
cd ~/OpenPCDet
python3.6 -m pip install -r requirements.txt
python3.6 setup.py develop
```

Comandos para ejecución de OpenPCDet

```
cd ~/OpenPCDet/tools
python3.6 single_inference.py
```

4. AB3DMOT

El sistema de seguimiento está basado en el código proporcionado en un *fork* propio del github de Xinshuo Weng, https://github.com/JavierEgido/AB3DMOT.
Para poder realizar inferencias y visualización sobre imagen en tiempo real se han desarrollado dos códigos alternativos a main.py y visualization.py, de forma que tanto la entrada como la salida son mensajes de ROS.

Comandos para instalación de AB3DMOT

```
git clone https://github.com/xinshuoweng/Xinshuo_PyToolbox
cd Xinshuo_PyToolbox
pip install -r requirements.txt
cd
git clone https://github.com/xinshuoweng/AB3DMOT.git
pip install -r requirements.txt
echo 'export PYTHONPATH=${PYTHONPATH}:~/AB3DMOT' >> ~/.bashrc
echo 'export PYTHONPATH=${PYTHONPATH}:~/Xinshuo_PyToolbox' >> ~/.bashrc
```

Comandos para ejecución de AB3DMOT

```
python main_ros.py
python visualization_ros.py
```

5. BEV_tracking

Las comunicaciones entre los módulos de detección y tracking han sido implementadas en base a mensajes *custom* desarrollados sobre ROS.

Para facilitar la implementación al usuario, el paquete BEV_tracking recoge dichos mensajes, pudiendo instalarse fácilmente mediante su incorporación a un directorio de trabajo.

El paquete de ROS BEV_tracking se encuentra publicado en el siguiente GitHub: https://github.com/JavierEgido/bevtracking

Comandos para instalación de BEV tracking

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
cd src
git clone https://github.com/JavierEgido/bevtracking
cd ..
catkin_make
```

Bibliografía

[1] A. Geiger, P. Lenz y R. Urtasun, «Are we ready for autonomous driving? the kitti vision benchmark suite,» *2012 IEEE Conference on Computer Vision and Pattern Recognition,* pp. 3354--3361, 2012.

[2] X. Weng and K. Kitani, "A Baseline for 3D Multi-Object Tracking," *arXiv,* 2019.

[3] A. Dosovtskiy, G. Ros, F. Codevilla, A. López y V. Koltun, «CARLA: An Open Urban Driving Simulator,» *arXiv,* 2017.

[4] L. Peña y T. Ausín, El valor de la movilidad humana, Plaza y Valdés, 2015-03.

[5] S. International, "J3016_201806. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," 2018.

[6] A. Bochkovskiy, C.-Y. Wang y H.-Y. Mark Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection,» *arXiv,* p. 2004.10934, 2020.

[7] Z. Qin, J. Wang y Y. Lu, «Triangulation Learning Network: From Monocular to Stereo 3D Object Detection,» *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2019.

[8] H. Konigshof, N. Salscheider y C. Stiller, «Realtime 3D Object Detection for Automated Driving Using Stereo Vision and Semantic Information,» *2019 IEEE Intelligent Transportation Systems Conference (ITSC),* vol. 10.1109, nº ITSC.2019.8917330, pp. 1405-1410, 2019.

[9] T. Takikawa, D. Acuna, V. Jampani y S. Fidler, «Gated-SCNN: Gated Shape CNNs for Semantic Segmentation,» *2019 IEEE/CVF International Conference on Computer Vision (ICCV),* nº 10.1109/iccv.2019.00533, 2019.

[10] T. Johansson y O. Wellenstam, LiDAR clustering and shape extraction for automotive applications, Gothenburg, Sweden: CHALMERS UNIVERSITY OF TECHNOLOGY, 2017.

[11] B. Yang, M. Liang y R. Urtasun, «HDNET: Exploiting HD Maps for 3D Object Detection,» *Proceedings of The 2nd Conference on Robot Learning,* pp. 146-155, 2018.

[12] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang and O. Beijbom, "PointPillars: Fast Encoders for Object Detection From Point Clouds," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 2019.

[13] B. Y. W. Luo y R. Urtasun, «Fast and Furious: Real Time End-to-End 3D Detection,

Tracking and Motion Forecasting with a Single Convolutional Net,» *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition,* nº 10.1109/CVPR.2018.00376, pp. 3569-3577, 2018.

[14] S. Shi, X. Wang y H. Li, «PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud,» *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* nº 10.1109/cvpr.2019.00086, 2019.

[15] A. Bewley, Z. Ge, L. Ott, F. Ramos y B. Upcroft, «Simple online and realtime tracking,» *2016 IEEE International Conference on Image Processing (ICIP),* nº 10.1109/icip.2016.7533003, 2016.

[16] B. Y. H. W. Kuhn, «The Hungarian method for the assignment problem,» *Naval Res. Logist. Quart,* pp. 83-97, 1955.

[17] N. Wojke, A. Bewley y D. Paulus, «Simple online and realtime tracking with a deep association metric,» *2017 IEEE International Conference on Image Processing (ICIP),* nº 10.1109/icip.2017.8296962, 2017.

[18] C. Gómez-Huélamo, J. del Egido, L. M. Bergasa, R. Barea, M. Ocaña, F. Arango y R. Gutiérrez, «Real-time bird's eye view multi-object tracking system basedon fast encoders for object detection,» *2020 IEEE Intelligent Transportation Systems Conference (ITSC),* 2020.

[19] Yan Yan, Y. Mao y B. Li, «SECOND: Sparsely Embedded Convolutional Detection,» *Sensors,* nº doi.org/10.3390/s18103337, 2018.

[20] S. Shi, Z. Wang, J. Shi, X. Wang y H. Li, «From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* nº 10.1109/tpami.2020.2977026, 2020.

[21] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang y H. Li, «PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection,» *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* nº 10.1109/cvpr42600.2020.01054, 2020.

[22] C. R. Qi, H. Su, M. Kaichun y L. J. Guibad, «PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,» *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* nº 10.1109/cvpr.2017.16, 2017.

[23] J. Ku, M. Mozifian, J. Lee, A. Harakeh y S. L. Waslander, «Joint 3D Proposal Generation and Object Detection from View Aggregation,» *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* nº

10.1109/iros.2018.8594049, 2018.

[24] Y. Zhou y O. Tuzel, «VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection,» *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition,* nº 10.1109/cvpr.2018.00472, 2018.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu y A. C. Berg, «SSD: Single Shot MultiBox Detector,» *Lecture Notes in Computer Science,* nº 10.1007/978-3-319-46448-0_2, p. 21–37, 2016.

[26] OpenMMLab, «OpenPCDet,» 2020. [En línea]. Available: https://github.com/open-mmlab/OpenPCDet.

[27] P. C. Mahalanobis, «On the generalized distance in statistics,» *Proceedings of the National Institute of Sciences (Calcutta),* vol. 2, pp. 49--55, 1936.

[28] L. Zheng, Z. Bie, Y. Sun, J. Wang, C. Su, S. Wang y Q. Tian, «MARS: A Video Benchmark for Large-Scale Person Re-Identification,» *Computer Vision - ECCV,* 2016.

[29] E. Baser, V. Balasubramanian, P. Bhattacharyya y K. Czarnecki, «FANTrack: 3D Multi-Object Tracking with Feature Association Network,» *2019 IEEE Intelligent Vehicles Symposium (IV),* nº 10.1109/ivs.2019.8813779, 2019.

[30] D. Frossard y R. Urtasun, «End-to-end Learning of Multi-sensor 3D Tracking by Detection,» *2018 IEEE International Conference on Robotics and Automation (ICRA),* nº 10.1109/icra.2018.8462884, 2018.

[31] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan y O. Beijbom, «nuScenes: A Multimodal Dataset for Autonomous Driving,» *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* nº 10.1109/cvpr42600.2020.01164, 2020.

[32] P. Sun, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine y et al, «Scalability in Perception for Autonomous Driving: Waymo Open Dataset,» *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* nº 10.1109/cvpr42600.2020.00252, 2020.

[33] Sanders y Andrew, An Introduction to Unreal Engine 4, USA: A. K. Peters, Ltd., 2016.

[34] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler y A. Ng, «ROS: an open-source Robot Operating System,» *ICRA,* vol. 3, nº 3.2, p. 5, 2009.

[35] Z. Pei, «Deep Sort with PyTorch,» 2020. [En línea]. Available:

https://github.com/ZQPei/deep_sort_pytorch.

[36] K. Bernardin y R. Stiefelhagen, «Evaluating multiple object tracking performance: The CLEAR MOT metrics,» *EURASIP Journal on Image and Video Processing,* nº 10.1155/2008/246309, 2008.

[37] NHTSA, Pre-Crash Scenario Typology for Crash Avoidance Research, 2007.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá