Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Algoritmos de detección de objetos 3D basados en LiDAR: comparación entre técnicas PCL clásicas y Deep Learning

Autor: Javier de la Peña García

Tutores: Luis Miguel Bergasa Pascual y Carlos Gómez Huélamo

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de Grado

Algoritmos de detección de objetos 3D basados en LiDAR: comparación entre técnicas PCL clásicas y Deep Learning

Autor: Javier de la Peña García

Directores: Luis Miguel Bergasa Pascual y Carlos Gómez Huélamo

Tribunal:

Presidente: Felipe Espinosa Zapata

Vocal 1º: Fernando Naranjo Vega

Vocal 2º: Rafael Barea Navarro

Calificación: .....

Fecha: .....

A nuestros alumnos pasados, presentes y futuros...

"Empieza haciendo lo necesario, luego haz lo posible y de pronto empezarás a hacer lo imposible."
Francisco de Asís

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Este trabajo es el fruto de muchas horas de trabajo, tanto de los autores últimos de los ficheros de la distribución como de todos los que en mayor o menor medida han participado en él a lo largo de su proceso de gestación.

Mención especial merece Manuel Ocaña, el autor de la primera versión de las plantillas de proyectos fin de carrera y tesis doctorales usadas en el Departamento de Electrónica de la Universidad de Alcalá, con contribuciones de Jesús Nuevo, Pedro Revenga, Fernando Herránz y Noelia Hernández.

En la versión actual, la mayor parte de las definiciones de estilos de partida proceden de la tesis doctoral de Roberto Barra-Chicote, con lo que gracias muy especiales para él.

También damos las gracias a Manuel Villaverde, David Casillas, Jesús Pablo Martínez, José Francisco Velasco Cerpa que nos han proporcionado secciones completas y ejemplos puntuales de sus proyectos fin de carrera.

Finalmente, hay incontables contribuyentes a esta plantilla, la mayoría encontrados gracias a la magia del buscador de Google. Hemos intentado referenciar los más importantes en los fuentes de la plantilla, aunque seguro que hemos omitido alguno. Desde aquí les damos las gracias a todos ellos por compartir su saber con el mundo.

Resumen

Este documento ha sido generado con una plantilla para memorias de trabajos fin de carrera, fin de máster, fin de grado y tesis doctorales. Está especialmente pensado para su uso en la Universidad de Alcalá, pero debería ser fácilmente extensible y adaptable a otros casos de uso. En su contenido se incluyen las instrucciones generales para usarlo, así como algunos ejemplos de elementos que pueden ser de utilidad. Si tenéis problemas, sugerencias o comentarios sobre el mismo, dirigidlas por favor a Javier de la Peña García <j.pena@edu.uah.es>.

Palabras clave: Plantillas de trabajos fin de carrera/máster/grado y tesis doctorales, LaTeX, soporte de español e inglés, generación automática.

Abstract

This document has been generated with a template for Bsc and Msc Thesis (trabajos fin de carrera, fin de máster, fin de grado) and PhD. Thesis, specially thought for its use in Universidad de Alcalá, although it should be easily extended and adapted for other use cases. In its content we include general instructions of use, and some example of elements than can be useful. If you have problemas, suggestions or comments on the template, please forward them to Javier de la Peña García <j.pena@edu.uah.es>.

Keywords: Bsc., Msc. and PhD. Thesis template, LaTeX, English/Spanish support, automatic generation.

Resumen extendido

Con un máximo de cuatro o cinco páginas. Se supone que sólo está definido como obligatorio para los TFGs y PFCs de UAH.

Índice general

Resumen ix

Abstract xi

Resumen extendido xiii

Índice general xv

Índice de figuras xix

Índice de tablas xxi

Índice de listados de código fuente xxiii

Índice de algoritmos xxv

Lista de acrónimos xxviii

1 Introducción 1

1.1 Sistemas de conducción autónomos . . . 1

1.2 Sistemas de percepción . . . 2

1.2.1 Principales sensores para la percepción en vehículos autónomos . . . 3

1.2.1.1 Cámara . . . 3

1.2.1.2 Radar . . . 4

1.2.1.3 LiDAR . . . 4

1.2.2 Sistemas de detección . . . 5

1.2.3 Sistemas de seguimiento . . . 6

1.2.4 Fusión sensorial . . . 7

1.3 Deep Learning . . . 8

2 Propuesta de trabajo 9

3 Sistemas clásicos de percepción con LiDAR 11

3.1 Voxelización . . . 11

3.2 RANSAC-3D . . . 12

3.3 KD-tree . . . 14

3.4 Filtrado previo y posterior a la detección . . . 18

4 Sistemas de percepción con LiDAR basados en Deep Learning 19

4.1 Principales datasets . . . 19

4.1.1 Kitti . . . 19

4.1.1.1 Análisis de la estructura del GT y las PCLs . . . 19

4.1.2 Waymo . . . 19

4.1.3 nuScenes . . . 19

4.1.3.1 nuScenes-devkit . . . 19

4.1.4 Comparativa entre los diferentes datasets . . . 19

4.2 Estado del arte en detección utilizando LiDAR . . . 19

4.2.1 PointPillars . . . 19

4.2.2 SECOND . . . 19

4.2.3 PointRCNN . . . 19

4.2.4 PV-RCNN . . . 19

4.2.5 CBGS . . . 19

4.3 OpenPCDet . . . 19

5 Desarrollo realizado 21

5.1 Estado del proyecto T4AC . . . 21

5.1.1 ROS . . . 21

5.1.2 Docker . . . 21

5.1.3 Estructura del proyecto . . . 21

5.2 Implementación en CARLA . . . 21

5.2.1 CARLA . . . 21

5.2.2 Funcionamiento del LiDAR en CARLA . . . 21

5.2.3 Implementación del sistema clásico utilizando LiDAR . . . 21

5.2.4 Implementación del sistema basado en Deep Learning utilizando LiDAR . . . 21

5.3 Fusión sensorial . . . 21

5.4 Vehículo del proyecto T4AC . . . 21

5.5 Implementación sobre el vehículo T4AC . . . 21

6 AD DevKit **23**

6.1 Estado del arte en evaluación de vehículos autónomos 23

6.2 Obtención del ground truth 23

6.3 Evaluación de los modelos 23

7 Resultados obtenidos **25**

7.1 Análisis cuantitativo en Kitti 25

7.2 Análisis cuantitativo en nuScenes 25

7.3 Análisis cualitativo del modelo clásico en CARLA 25

7.4 Análisis cualitativo de CBGS en CARLA 25

7.4.1 Comparativa con PointPillars en CARLA 25

7.5 Análisis cuantitativo de CBGS en CARLA 25

7.6 Análisis cualitativo de CBGS sobre el vehículo T4AC 25

7.6.1 Comparativa con PointPillars sobre el vehículo T4AC 25

8 Conclusiones **27**

8.1 Modelos estudiados 27

8.2 Comparativas adicionales 27

8.2.1 Ajuste de modelos basados en Kitti a nuScenes 27

8.2.2 Número de PCL de entrada en modelos evaluados sobre nuScenes 27

8.2.3 Tamaño del voxel en modelos basados en redes neuronales 27

8.3 Futuros trabajos 27

Bibliografía **29**

A Herramientas y recursos **31**

Índice de figuras

1.1 Arquitectura de un sistema de conducción autonóma. 1
1.2 Niveles de autonomia. 2
1.3 Cámara utilizada en vehículos autónomos. 3
1.4 Uso de cámara con lluvia. 3
1.5 Radar utilizado en vehículos autónomos. 4
1.6 LiDAR utilizado en vehículos autónomos. 5
1.7 Detecciones 2D utilizando cámara. 5
1.8 Detecciones 3D utilizando LiDAR. 6
1.9 Tracking como vista de pájaro sobre una nube de puntos. 7
1.10 Fusión sensorial utilizando cámara y LiDAR. 7
1.11 Convolutional Neural Network. 8

3.1 Entorno 3D voxelizado. 11
3.2 Aplicación de RANSAC para detección de outliers. 12
3.3 Aplicación de RANSAC-3D. 14
3.4 Árbol binario ordenado. 15
3.5 Espacio bidimensional dividido por un KD-tree. 16
3.6 Estructura de un KD-tree de dos dimensiones. 16

Índice de tablas

7.1 Rendimiento medio de CBGS PointPillars Multihead en nuScenes. 25
7.2 Análisis por clase de CBGS PointPillars Multihead en nuScenes. 26
7.3 Análisis por clase de CBGS PointPillars Multihead en CARLA. 26

Índice de listados de código fuente

Índice de algoritmos

3.1 Algoritmo RANSAC 13
3.2 Búsqueda en árbol binario ordenado 15
3.3 Inserción en KD-tree 16
3.4 Cluster por distancia en KD-tree 17

Lista de acrónimos

AD DevKit	Autonomous Driving Development Kit.
ADAS	Advanced Driver Assistance System.
ADS	Automated Driving System.
CARLA	Car Learning to Act.
CNN	Convolutional Neural Network.
DL	Deep Learning.
EKF	Extended Kalman Filter.
FC	Fully Connected.
FPS	Frames per Second.
GNSS	Global Navigation Satellite System.
GPS	Global Positioning System.
IMU	Inertial Measurement Unit.
KF	Kalman Filter.
KIT	Karlsruhe Institute of Technology.
KNN	K Nearest Neighbors.
LiDAR	Light Detection and Ranging.
Radar	Radio Detection and Ranging.
RANSAC	Random Sampling and Consensus.
Sonar	Sound Navigation and Ranging.
UKF	Unscented Kalman Filter.
YOLO	You Only Look Once.

Capítulo 1

Introducción

No te conformes con el mundo que has heredado. Nunca se ha resuelto un desafío sin personas que pensasen diferente.

Tim Cook

Para el pleno entendimiento del trabajo desarrollado se explica el funcionamiento de los sistemas de conducción autónoma, para centrarse en la parte de percepción del vehículo y comprender que gracias al Deep Learning se consigue el estado del arte en este campo.

1.1 Sistemas de conducción autónomos

En los últimos años gracias a una mejora en los sensores, en la capacidad de computo principalmente por la aceleración por hardware, la visión por computador, el Deep Learning (DL) 1.3 y el desarrollo de técnicas de comunicación, ha propiciado que nos encontremos en una carrera por la creación de sistemas de conducción autónomos. Empresas de sectores de la automoción y la tecnológicas como ArgoAI, Audi, Baidu, Cruise, Mercedes-Benz, Tesla, Uber o Waymo entre otras, invierten enormes cantidades de dinero para el desarrollo de estas tecnologías [1].

Para la obtención de sistemas de conducción autónoma es necesario tener un buen entendimiento del entorno y hacer uso de un buen control en tiempo real, para ello son utilizados sensores que puedan aportar información al vehículo como son cámaras, LiDAR, Radar, IMU, GPS o hasta Sonar.

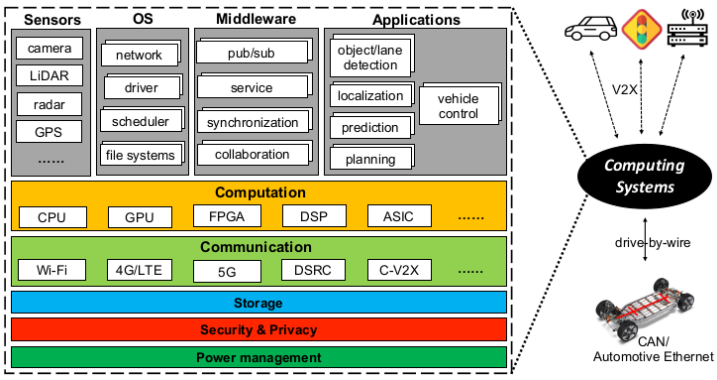

Figura 1.1: Arquitectura de un sistema de conducción autonóma.

En adición a los sensores también es necesario la utilización de sistemas de localización tanto global como local, mapeado del entorno, toma de decisiones y control del vehículo.

La evolución continua de estos sistemas trata de ofrecer un mayor nivel de seguridad al volante con Advanced Driver Assistance System (ADAS), para que en un futuro puedan ser remplazados por Automated Driving System (ADS).

Para analizar el avance de estos sistemas y para poder compararlos, se ha dividido según su nivel de autonomía, por lo que se tiene desde un nivel 0 a un nivel 5. El nivel 0 indica que el coche no tiene ningún tipo de autonomía, en el nivel 1 el vehiculo sigue siendo controlado por el conductor pero ciertas características de ayuda a la conducción son añadidas, en el siguiente nivel el vehículo es capaz de acelerar, frenar y hasta dirigir el vehículo pero con el conductor siempre atento, en el nivel 3 el conductor es necesario pero no es requerimiento la atención al entorno, pero debe de estar listo para tomar en control en todo momento, el nivel 4 permite un nivel de autonomía donde el vehículo no requiere de atención pero unicamente en ciertos escenarios y el último nivel es el que habilita la conducción autónoma completa [2].

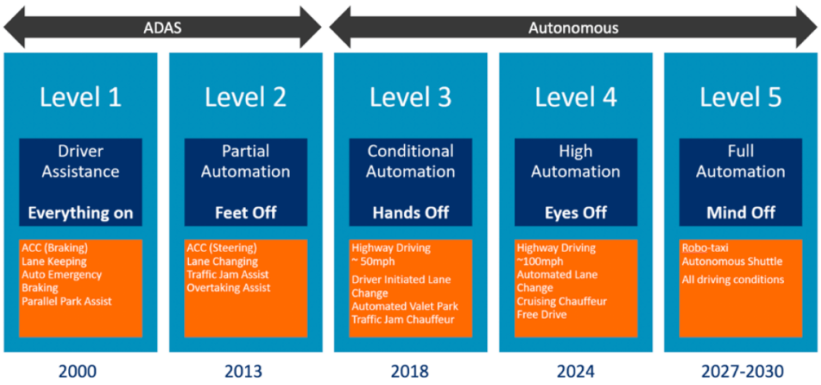

Figura 1.2: Niveles de autonomia.

El desarrollo de este tipo de sistemas no es una tarea sencilla, múltiples empresas involucradas en el desarrollo de vehículos autónomos pretendían tener vehículos en el nivel 4 de autonomía en poco tiempo, pero como se ha visto esto no es posible, actualmente nos encontramos en el mercado con sistemas que se encuentran entre el nivel 2 y el 3, por lo que aún queda un largo camino antes de llegar a conducción autónoma completa.

1.2 Sistemas de percepción

Los sistemas de ADS/ADAS requieren de un entendimiento del entorno para poder funcionar correctamente, para ello es necesario añadir diversos sensores a los largo del vehículo que nos permitan obtener la mayor información del exterior posible. A partir de este conocimiento es posible la toma de decisiones y la planificación, por lo que en este apartado se va a explicar de que manera se puede configurar un sistema de percepción, cuales son los principales sensores y que información se puede obtener de cada uno de ellos.

1.2.1 Principales sensores para la percepción en vehículos autónomos

Para la creación un sistema de percepción robusto es necesario el uso de diversos tipos de sensores que ofrezcan una información de relevancia de manera diferente al resto, por ello se utilizan sensores como cámaras, Radar, LiDAR, sensores de ultrasonidos, GPS, GNSS, IMU etc.

Estos ofrecen información de localización, velocidad, distancia de objetos en el entorno, e incluso información del propio vehículo, como su propia localización, la velocidad lineal y angular que este tiene.

También es necesario tener en cuenta que no todos los sensores funcionan de la misma manera en distintos escenario por lo que en situaciones donde un sensor es incapaz de obtener buenos datos otro sensor puede suplir esta carencia, por lo que la redundancia de sensores aporta otro nivel de seguridad al vehículo ya no solo un nivel mayor de detección del entorno.

1.2.1.1 Cámara

Uno de los sensores más utilizados es la cámara, este es el más extendido debido a la gran riqueza de información que ofrece del entorno. Actualmente se pueden encontrar cámaras que generen imágenes a una gran resolución y a una alta tasa de FPS por un precio bastante asequible, este es uno de los sensores más baratos.

Figura 1.3: Cámara utilizada en vehículos autónomos.

El problema de este sensor recae en el computo que es necesario para obtener información a partir de las imágenes, ya que estas no son más que píxeles en escala de grises o con un sistema de colores como el RGB. Por ello no solo es necesario tener el cuenta el coste del sensor, sino que también hay que aumentar la capacidad de computo del ordenador de abordo para que pueda analizar en tiempo real las imágenes.

Por último es necesario conocer las limitaciones de la cámara, esta funciona de forma correcta en situaciones de buena luminosidad y sin reflejos, por lo que en situaciones con lluvia 1.4, niebla, durante la noche y otros escenarios climatológicos adversos, no es capaz de obtener toda la información que esta obtendría en situaciones más favorables, lo cual hace que otros sensores sean usados en estas condiciones adversas para lidiar con estar limitaciones.

Figura 1.4: Uso de cámara con lluvia.

Aún conociendo las desventajas de estos sensores la gran mayoría de enfoques incluyen cámaras para la obtención de los objetos del entorno tanto en 2D como en 3D, pudiendo utilizar para lo segundo un sistema de cámaras estéreo que obtienen también información de la profundidad.

1.2.1.2 Radar

Los Radar son utilizados en múltiples aplicaciones como la previsión meteorológica, la astronomía, las comunicaciones, la navegación oceánica y la conducción autónoma entre otras.

Este sensor emite ondas de radio, las cuales son reflejadas devuelta a este, lo cual da una información de donde se hayan los objetos en el espacio tridimensional, lo que implica la distancia a estos junto con los dos ángulos necesarios, además gracias al efecto Doppler se puede inferir la velocidad de los objetos a partir de un fenómeno que hace variar la frecuencia de la onda enviada si hay algún tipo de movimiento local relativo respecto del propio Radar [3, 4].

Figura 1.5: Radar utilizado en vehículos autónomos.

Como se ha visto este sensor al contrario que la cámara obtiene directamente información utilizable para el entendimiento del entorno de forma directa, el problema radica en la escasa cantidad de datos que provee. Aunque se obtenga información de localización 3D y de velocidad, la cantidad de la nube de puntos producida es muy pequeña, por lo que es necesario de otros sensores para obtener una información completa del entorno.

Por otra parte, una de las principales ventajas del Radar es que se puede utilizar en cualquier situación meteorológica, unicamente podría verse afectado por lluvias muy intensas. Por lo que es un sensor muy completo y una gran adición para obtener información adicional de posición 3D y velocidad a un precio inferior a un LiDAR y por ello es adoptado por gran cantidad de sistemas ADAS en conjunto con sistemas de cámaras 360 alrededor del vehículo.

1.2.1.3 LiDAR

De formo similar al Radar, los sistemas LiDAR basan su funcionamiento en el escaneo del entorno a partir de el envío de láseres y el cálculo del tiempo desde su envío hasta su retorno. Con esta información de distancia y el ángulo de inclinación del haz que envió esa señal, se construye una nube de puntos que consta de valores x, y, z de posición y otro valor que es el coeficiente de reflectividad del rayo de luz con el objeto incidido [3].

Actualmente los LiDAR más utilizados son de 64 canales, lo cual indica que se tienen 64 láseres funcionando al mismo tiempo lo que da una gran resolución del entorno, además la nube de puntos generada es de hasta 120 metros alrededor del vehículo lo cual permite detectar objetos a una distancia

considerable y saber de manera casi perfecta su distancia en un entorno tridimensional gracias a los alrededor de 2.000.000 de puntos que se generan por segundo del entorno [5].

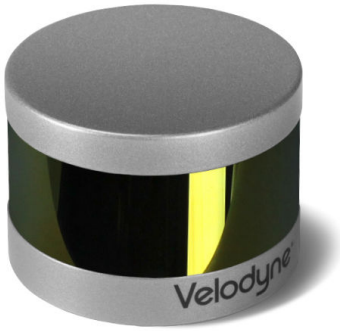

Figura 1.6: LiDAR utilizado en vehículos autónomos.

Los sistemas LiDAR tienen la ventaja de ser un sensor que aporta mucha información del entorno, pero tienen el mismo problema que las cámaras, en condiciones de lluvia, nieve, granizo o niebla, la efectividad de este sensor decae aunque se trata de minimizar ajustando la longitud de onda del láser utilizado [6].

Aún siendo un sensor muy útil, que puede aumentar el nivel de redundancia del sistema además de la seguridad, múltiples compañías como Tesla tratan de evitar su uso utilizando unicamente cámaras y Radar, esto es debido a que un LiDAR suele costar entre 8.000 y 100.000 dólares si se requiere de un resolución similar al estado del arte entre 16 y 128 haces [1].

1.2.2 Sistemas de detección

Unicamente con un sistema de sensores no es posible la comprensión del entorno, también es necesario de un procesamiento de los datos, mientras que la cámara no da ninguna información de forma directa, el LiDAR y el Radar son capaces de obtener la posición de obstáculos alrededor del vehículo, y además el Radar es capaz de inferir la velocidad de los objetos sin necesidad de un seguimiento.

Principalmente en los sistemas de detección para conducción autónoma se trata de obtener las posiciones de los diferentes objetos de interés del entorno. Estas detecciones pueden ser tanto en 2D como en 3D, pero el problema radica en como obtener un rectángulo u ortoedro que identifique donde se encuentran dicho objetos del entorno.

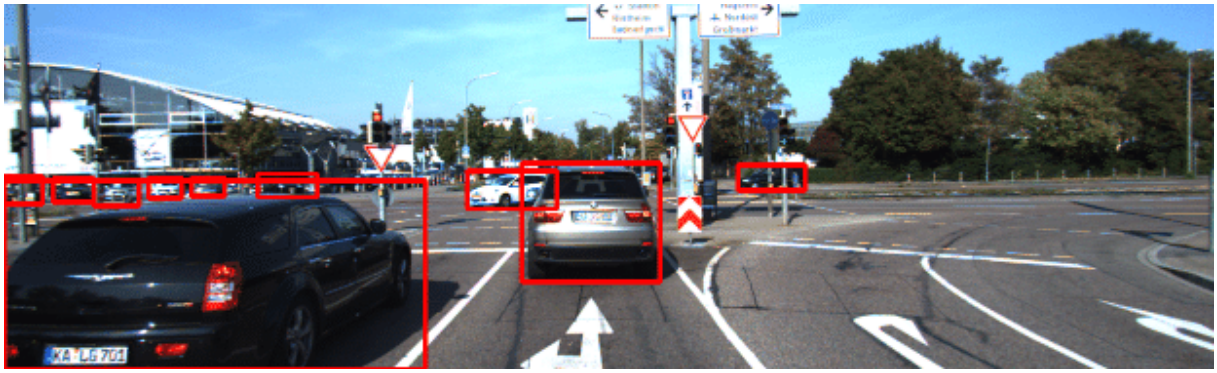

Figura 1.7: Detecciones 2D utilizando cámara.

Una de las formas de detección del entorno es a partir de cámaras, esto puede ser conseguido con un sistema de una cámara o de un sistema multicámara que abarque los 360 grados alrededor del coche, con esto instalado en el coche se puede generar rectángulos sobre las imágenes de los objetos del entorno como coches, peatones, bicicletas, motocicletas... Con esto se obtendría un listado de objetos detectados en 2D, lo cual es obtenible con un modelo basado en redes neuronales como YOLO [7] que es capaz de hacerlo en tiempo real, y que funciona tal y como se ve en 1.7.

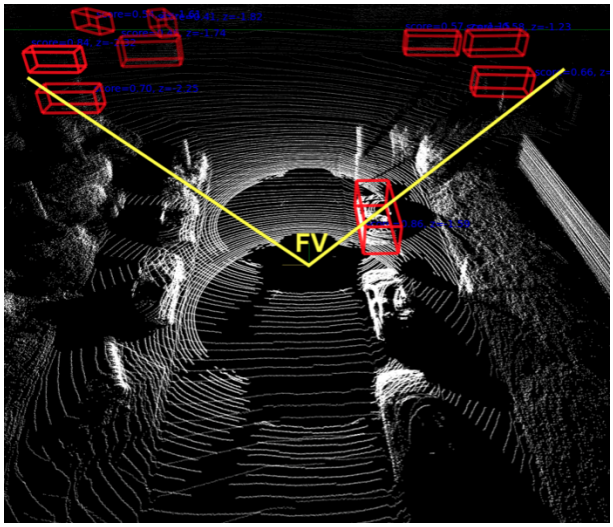

Figura 1.8: Detecciones 3D utilizando LiDAR.

Unicamente con detecciones 2D no se puede obtener la distancia a los vehículos, como mucho una aproximación a partir del tamaño de la bounding box 2D, aparte de la dirección en la que se encuentran respecto del coche. Por lo que termina trabajando con detecciones 3D, las cuales pueden tener como paso intermedio una detección 2D o ser obtenidas directamente con un sistema estéreo de cámaras o a partir de las nubes de nubes del Radar o del LiDAR.

Las técnicas que realizan detecciones en un sistema tridimensional tienen un coste computacional mayor al trabajar con una dimensión añadida, por lo que es necesaria la utilización de técnicas que permitan realizar estas detecciones en tiempo real, como es el caso de PointPillars [8], modelo que se explicará más adelante en 4.2.1, y que como se ve en 1.8 es capaz de realizar las detecciones en un entorno tridimensional utilizando unicamente el LiDAR.

1.2.3 Sistemas de seguimiento

Mientras que en los sistemas de detección se suele utilizar un único estado discretizado del entorno percibido por los sensores, el seguimiento o tracking utiliza múltiples estados para el reconocimiento de los objetos en múltiples escenas, con su posterior asociación y el calculo de la trayectoria de estos.

En este campo se pueden tomar diversos acercamientos al problema, utilizando técnicas clásicas como Kalman Filter (KF), Extended Kalman Filter (EKF) o Unscented Kalman Filter (UKF), modelos neuronales como en [9] o modelos end-to-end que incorporan detección y tracking en un mismo modelo neuronal como es el caso de PointTrackNet [10] o modelos que realizan tracking de forma implícita como CBGS [11] que calculan la velocidad de los objetos sin devolver identificadores de estos 4.2.5.

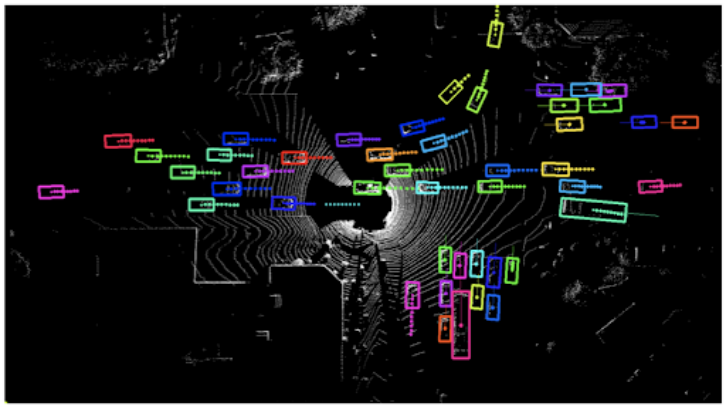

Figura 1.9: Tracking como vista de pájaro sobre una nube de puntos.

Los sistemas de detección suelen trabajar a partir de modelos en dos dimensiones que trabajan en vista de pájaro lo cual elimina lidiar con movimientos verticales, en estos se detecta la posición actual, velocidad lineal, velocidad angular y se puede definir el movimiento todo lo complejo que se desee. Tras esto también se predice los futuros estados de los objetos, manteniendo un identificador asociado a estos a los luargo de los frames analizados.

1.2.4 Fusión sensorial

Los sistemas de fusión sensorial son necesario para aumentar la precisión en los sistemas de detección y tracking, como se ha visto en 1.2.1, los sensores ofrecen información diferente al resto, por lo que el aumento de precisión es producido por el aumento en la cantidad de datos proveniente de todos los sensores. La complejidad de estas técnicas radica en el uso eficiente de todos los sensores.

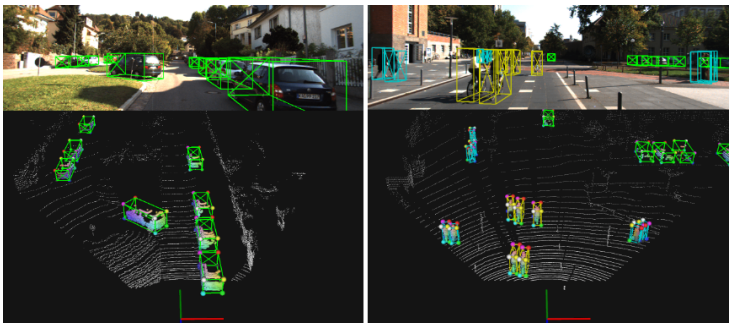

Figura 1.10: Fusión sensorial utilizando cámara y LiDAR.

Para la realización de un sistema de fusión sensorial se pueden tomar diversos acercamientos, en una early fusion se tratan todos los datos en crudo de los sensores para obtener un mejor sistema de percepción, si se trabaja con sensores tratados pero no con las detecciones finales se trata de una middle fusion, en el caso de utilizar las detecciones finales estaríamos ante una late fusion.

Entre las técnicas más utilizas para sistemas de fusión sensorial encontramos los KF, EKF y UKF [12], pero también se comienzan a utilizar técnicas más complejas que radican en el uso de redes neuronales para la fusión.

1.3 Deep Learning

Gracias al aumento en capacidad de computo, la mejora en el procesamiento de grandes volúmenes de datos y el desarrollo de nuevos algoritmos que aprendan con estos datos, se ha producido un estallido en el uso de técnicas basadas en redes neuronales. Desde 1958 con la creación del Perceptrón se conoce de este tipo de técnicas, pero a principios del siglo XXI es cuando realmente se ve su potencial.

Las redes neuronales profundas o DL son aquellas redes neuronales con múltiples capas intermedias que permiten la extracción del conocimiento. Estas son utilizadas en ámbitos como el análisis de datos tabulares, modelos del lenguaje o sistemas de visión que es donde se centrará este estudio.

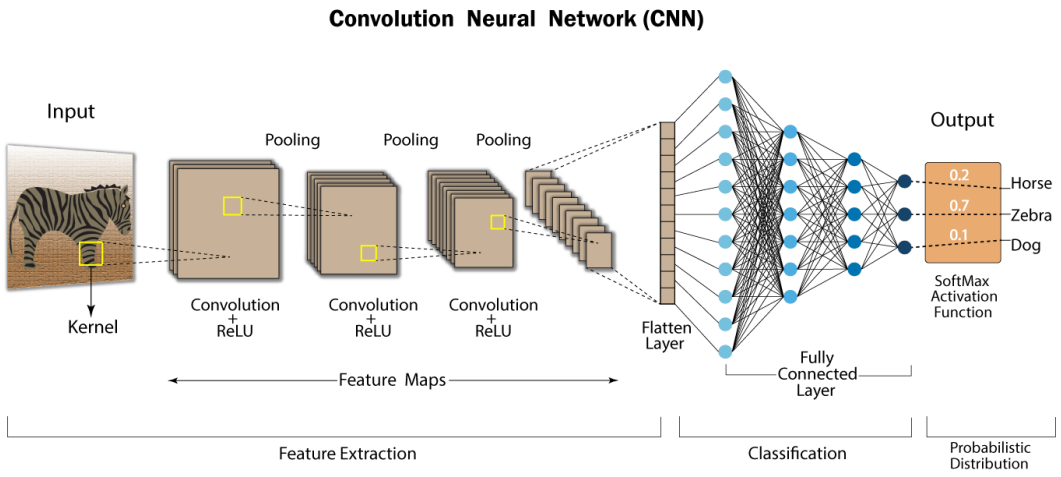

Figura 1.11: Convolutional Neural Network.

En el campo de la visión artificial las Redes Neuronales Convolucionales o Convolutional Neural Network (CNN) han conseguido una mejora importante en la precisión de los modelos. Estas redes son utilizadas principalmente para su uso con cámaras ya que su estructura tridimensional funciona de forma muy buena con las CNN. Un modelo típico para uso con imágenes sería 1.11.

Para los sistemas de detección con LiDAR basados Deep Learning son utilizadas principalmente las CNN junto con capas Fully Connected (FC), ya que son las que obtienen un mejor rendimiento como se verá en el capítulo 4.

Capítulo 2

Propuesta de trabajo

La educación científica de los jóvenes es al menos tan importante, quizá incluso más, que la propia investigación.

Glenn Theodore Seaborg

Este trabajo se encuentra dentro del proyecto Tech4AgeCars perteneciente al grupo RobeSafe. En este se pretende realizar un estudio de diferentes técnicas de detección utilizando LiDAR, tanto con métodos clásicos como basados en Deep Learning para el análisis en datasets reales, en el simulador Car Learning to Act (CARLA) y en el coche del grupo RobeSafe.

En la arquitectura del proyecto se tiene implementado un procesamiento basado en PointPillars para una ejecución en tiempo real sobre una plataforma NVIDIA Jetson AGX Xavier [13], por lo que se pretende remplazar esta subtarea de detección utilizando LiDAR, dentro de la capa de percepción.

Con la implementación de la detección realizada, se pretende realizar una fusión sensorial entre cámara y LiDAR junto con un compañero del grupo RobeSafe que se encuentra realizando un TFG de detección 3D utilizando cámara [14].

Tras una fusión y sin la posibilidad de analizar el sistema de percepción a crear, se trabajará por último en un proyecto junto con el Karlsruhe Institute of Technology (KIT) llamado Autonomous Driving Development Kit (AD DevKit) que tratará de analizar un Automated Driving System, en concreto se trabajará en el apartado de percepción del kit de desarrollo, todo ello esperando que este proyecto no solo sea útil para los compañeros del proyecto sino para cualquier desarrollador que utilice el simulador CARLA.

Capítulo 3

Sistemas clásicos de percepción con LiDAR

El placer más noble es el júbilo de comprender.

Leonardo da Vinci

Mientras que se tienen múltiples tipos de técnicas de percepción tanto clásicas como basadas en DL, el uso de técnicas clásicas utilizando unicamente LiDAR no abundan, por lo que se presentan las técnicas estudiadas e implementadas en el simulador CARLA que permiten la detección de los objetos del entorno.

3.1 Voxelización

Las nubes de puntos generadas por el LiDAR pueden ser de hasta 1.300.000 puntos por segundo en un LiDAR de 64 haces [5] lo que implicaría el análisis de una gran cantidad de datos en tiempo real lo que puede no ser muy viable ya que se tiene una capacidad de computo limitada en un vehículo.

Para ello se utiliza la voxelización, esta no solo es utilizada en sistemas de percepción, sino que también es utilizada en imágenes volumétricas de ámbito médico, para la representación del terreno o en el pipeline gráfico de un ordenador. Esta técnica trata de reducir la cantidad de datos en memoria a la vez que reduce el computo al reducir la resolución de la escena. Por lo que se puede entender como un proceso de discretización del entorno.

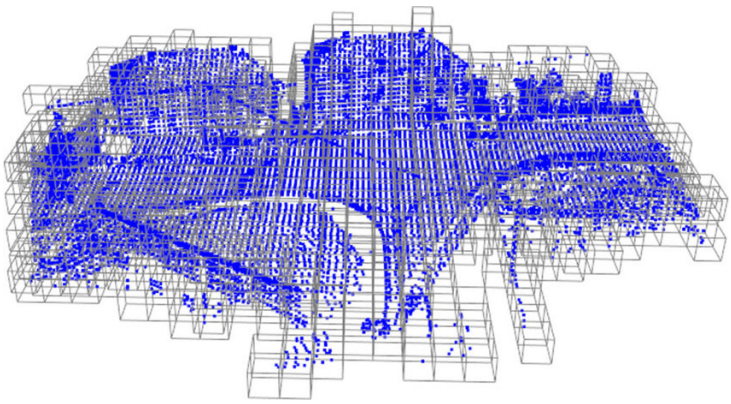

Figura 3.1: Entorno 3D voxelizado.

Trabajando con nubes de puntos, la voxelización sigue los siguientes pasos:

1. Definición del tamaño del vóxel, lo que sería un vector tridimensional.
2. A partir del tamaño del vóxel se divide la escena en un conjunto de ortoedros u vóxeles.
3. Si se encuentra un punto del Light Detection and Ranging (LiDAR) dentro de un vóxel este de activa

Esta técnica como se verá en el capítulo 4, también se utiliza en diversos modelos basados en DL, esto se hace para trabajar de forma similar a lo que sería la estructura de una imagen que se encuentra compuesta por píxeles en vez de por vóxeles.

3.2 RANSAC-3D

Para la detección de los objetos del entorno no es necesaria la información de los puntos que inciden en el suelo, por lo que una de las técnicas utilizadas para la selección del plano perteneciente al suelo es Random Sampling and Consensus (RANSAC)-3D.

El algoritmo RANSAC [15] tiene una funcionalidad similar a la regresión linear, ambos algoritmos a partir de un conjunto de datos hayan la relación lineal entre dos características. La creación de este algoritmo tenía como finalidad el ajuste de datos experimentales, el uso en el análisis de escenas y generación automática de mapas.

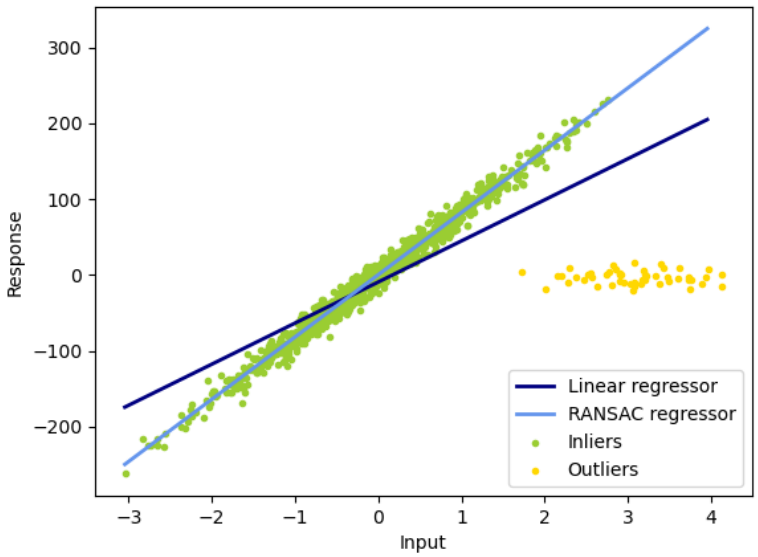

Figura 3.2: Aplicación de RANSAC para detección de outliers.

La idea principal del algoritmo RANSAC es la generación de rectas a partir de 2 o más puntos para aceptar como la mejor recta aquella que contenga más puntos entre un límite seleccionado, y esto es repetido un número arbitrario de veces. Esta recta será la que contenga los puntos asumidos como normales o inliers, y el resto de puntos son asumidos como anómalos o outliers. El algoritmo completo sería el siguiente 3.1.

Input

data	Conjunto de observaciones
model	Modelo que explica las observaciones
n	Mínimo número de puntos necesarios para estimar un modelo
k	Número de iteraciones del algoritmo
t	Valor límite que indica que puntos se encuentran bien estimados
d	Número de puntos cercanos que asegura que el modelo sea válido

Output

bestFit	Parámetros del modelo que ajustan de mejor manera a los datos

$iterations \leftarrow 0$
$bestFit \leftarrow$ null
$bestError \leftarrow \infty$
while $iterations < k$ **do**
  $maybeInliers \leftarrow n$ puntos seleccionados aleatoriamente
  $maybeModel \leftarrow$ modelo que se ajusta a $maybeInliers$
  $alsoInliers \leftarrow$ set vacío
  for *cada punto que no se encuentre en* $maybeInliers$ **do**
    if *error de ajustar el punto a* $maybeModel < t$ **then**
      añadir punto a $alsoInliers$
    end
  end
  if *número de puntos en* $alsoInliers > d$ **then**
    $betterModel \leftarrow$ parámetros del modelo sobre el que han sido ajustados los puntos de $maybeInliers$ y $alsoInliers$
    $thisErr \leftarrow$ medida de como de bien han sido ajustado los puntos;
    if $thisErr < bestErr$ **then**
      $bestFit \leftarrow betterModel$
      $bestErr \leftarrow thisErr$
    end
  end
  $iterations \leftarrow iterations+ 1$;
end

Algoritmo 3.1: Algoritmo RANSAC

En el caso de las nubes de puntos que devuelve el LiDAR, se trabaja en un entorno tridimensional, por lo que no funciona de la misma manera dicho algoritmo, se utiliza una variación, RANSAC-3D como se ve en 3.3, que en vez de trabajar con datos en 2D se trabajan en 3D por lo que en vez de ajustar un modelo lineal se ajusta como un plano, por lo que como mínimo se necesitan tres puntos para generar un posible modelo ya que es el mínimo número de puntos para generar un plano, el resto funciona de forma similar definiendo el límite de distancia, iteraciones...

Como se explicó, los resultados suelen ser similares a una regresión linear en un entorno bidimensional, pero en este caso no sería del todo cierto, ya que el plano que abarca más puntos suele ser en la mayoría de los casos el correspondiente al suelo. Esto implica una modificación de la regresión linear a las tres dimensiones, lo sería una regresión ajustada a un plano, esta generaría en la mayoría de las situaciones un plano que se encontraría por encima del suelo, ya que se trataría de minimizar una métrica de error al plano (distancia euclídea, manhattan, minkowski, hamming...), por lo que los objetos de la escena conseguirían levantar el plano para minimizar el error de este a los puntos correspondientes a los objetos.

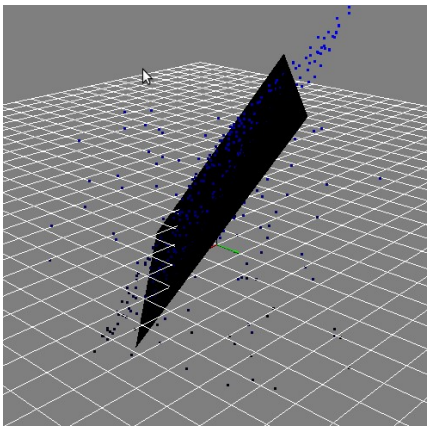

Figura 3.3: Aplicación de RANSAC-3D.

Al tener el resto de puntos por encima del suelo, nos encontraría ante una situación en la que el algoritmo RANSAC-3D ajustaría como inliers los puntos correspondientes al suelo y el resto se detectarían como outliers, por lo que es este el algoritmo seleccionado al ajustarse de la mejor manera a la tarea necesitada.

3.3 KD-tree

Tras la eliminación del suelo en la nube de puntos podemos encontrarnos con que los diferentes objetos del entorno se encuentran separados, ya que el suelo era el elemento unificador de la mayoría de puntos de la escena. Teniendo esto, es necesaria de una técnica que sea capaz de agrupar los puntos más cercanos de para que se agrupen por distancia, ya que si se hace comprobando cada punto con el resto se obtendría una complejidad de $O(n^2)$.

Teniendo en cuenta el coste computacional de los algoritmos de clustering y al trabajar con tantos puntos, alrededor de 1.000.000 por segundo y sabiendo que un LiDAR suele trabajar a 10 Hz, es muy recomendable aplicar una voxelización si no se aplicó previamente en la eliminación de los puntos incidentes en el suelo.

Para el clustering, se podría utiliza el algoritmo K Nearest Neighbors (KNN), pero esto produciría clústeres no válidos al encontrarse objetos con pocos vóxeles o con demasiados lo que produciría clústeres incompletos y otros mal formados sin no se tiene un cuenta una distancia máxima entre vóxeles.

El KD-tree [16] es una estructura de datos que con un eficiente uso de memoria, es capaz de hacer búsquedas en un entorno K dimensional con una complejidad media de $O(\log n)$, esto lo convierte en una gran estructura para trabajar con datos en un entorno tridimensional, como es el caso de las nubes de puntos o de vóxeles. Un KD-tree tiene una estructura similar a un árbol binario, la eficiencia de la estructura radica en la ordenación del mismo, donde en cada altura del árbol se ordena según una dimensión iterativamente.

Antes de analizar en profundidad la estructura KD-tree, es necesario comprender los árboles binarios, tanto su uso, como su utilidad. Los árboles binarios son una estructura de datos donde cada nodo tiene otros dos nodos hijos, referidos como hijo izquierdo e hijo derecho. La utilidad de la estructura radica en la forma en la que se pueden guardar los datos, mientras que para buscar un valor en una lista, es necesario iterar por todos ellos o hasta que se encuentre con una complejidad máxima de $O(n)$, un KD-tree tiene una complejidad máxima es de $O(\log_2 n)$.

Input

tree	Árbol binario ordenado
key	Clave del nodo buscado

Output

node	Nodo buscado

```
node ← null;
currentNode ← nodo raíz de tree
while currentNode ≠ null do
    if clave de currentNode = key then
        node ← currentNode
        break;
    end
    if clave de currentNode < key then
        currentNode ← hijo derecho de currentNode
    else
        currentNode ← hijo izquierdo de currentNode
    end
end
```

Algoritmo 3.2: Búsqueda en árbol binario ordenado

En el caso del árbol de la figura 3.4 para buscar el número 7:

1. Se empieza por el nodo con valor 8
2. Al ser $7 < 8$ se pasa al hijo de la izquierda
3. Como $7 > 3$ se salta al hijo de la derecha
4. Teniendo el nodo con valor 6, siendo menor que 7 se coge el hijo de la derecha
5. Por último se llegó al nodo con valor 7 requerido

Teniendo 9 nodos solo ha sido necesario analizar 4 nodos que la peor situación con este árbol.

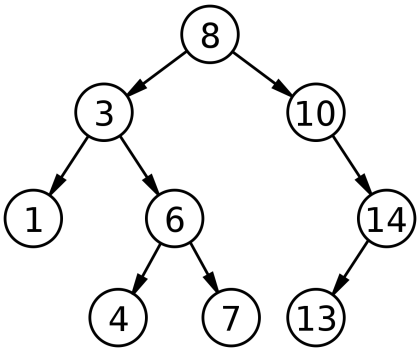

Figura 3.4: Árbol binario ordenado.

Al contrario que los árboles binarios, un KD-tree es capaz de un número K de dimensiones, por lo que hay una diferencia principal que es la rotación entre la dimensión sobre la que se ordena en cada altura del árbol. Esto produce que la forma de inserción 3.3 y búsqueda sea modificada.

Input

tree	KD-tree
node	Nodo a introducir
k	Número de dimensiones del árbol

Output

tree	KD-tree con el nodo introducido

currentNode ← nodo raíz de *tree*
depth ← 0
while *currentNode* ≠ *null* **do**
  x ← *depth* mod *k*
  if *valor de currentNode en la dimensión x* < *valor de node en la dimensión x* **then**
    currentNode ← hijo derecho de *currentNode*
  else
    currentNode ← hijo izquierdo de *currentNode*
  end
  depth ← *depth*+ 1
end
currentNode ← *node*

Algoritmo 3.3: Inserción en KD-tree

Lo que produce esta forma de guardar los datos en el árbol, es que según se aumenta la profundidad en el árbol, la región de los nodos hijos es cada vez menor, lo que permite una más sencilla agrupación y estudio de los datos por regiones en un entorno K dimensional. Como se ve en la figura 3.5 el espació bidimensional va siendo dividido por regiones, esto es gracias a que cada nodo divide en dos el espacio sobre el que se encuentran sus hijos, lo cual es una perfecta manera de agrupar los puntos en clústeres utilizando esta estructura, tal y como se detalla en el algoritmo 3.4

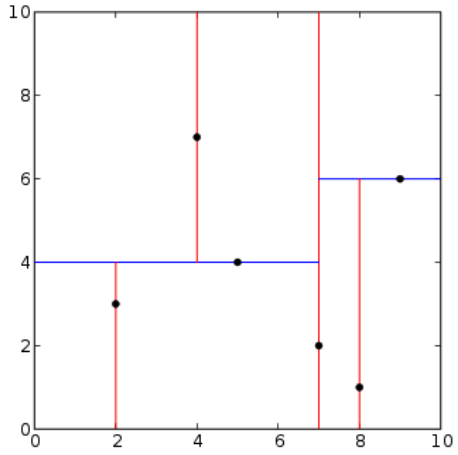

Figura 3.5: Espacio bidimensional dividido por un KD-tree.

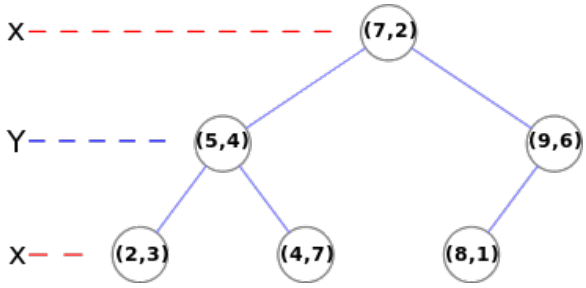

Figura 3.6: Estructura de un KD-tree de dos dimensiones.

Input

points	Nube de puntos del LiDAR
id_node	Id del punto sobre el que se va a comenzar el clúster
node	Nodo sobre el que buscar un clúster
processed	Vector de booleanos de tamaño igual al número de puntos
tree	KD-tree
distance	Distancia máxima a los puntos del clúster
k	Número de dimensiones del árbol

Output

cluster	Conjunto de los puntos perteneciente al clúster

Function `proximity` (*points, id_node, node, cluster, processed, tree, distance, k*)**:**
  processed[*id_node*] ← true
  añadir a *cluster points*[*id_node*]
  indexList ← `search` (*points*[*id_node*], *tree, distance, k*) **for** *index en indexList* **do**
    if *processed*[*index*] = *false* **then**
      `proximity` (*points, index, cluster, processed, tree, distance, k*)
    end
  end
Function `search` (*node, tree, distance, k*)**:**
  indexList ← lista vacía
  `searchNodes` (*node, tree, 0, distance, indexList, k*)
  return *indexList*
Function `searchNodes` (*node, tree, distance, depth, indexList, k*)**:**
  if *tree* ≠ *null* **then**
    if *distancia entre el nodo raíz de tree y node* < *distance* **then**
      añadir índice del nodo raíz de *tree* a *indexList*
    end
  end
  $x \leftarrow depth \bmod k$
  if *valor de node en la dimensión* x − *distance* < *valor del nodo raíz de tree en la dimensión* x **then**
    `searchNodes` (*node, árbol izquierdo de tree, depth+1, distance, indexList, k*)
  end
  if *valor de node en la dimensión* x + *distance* > *valor del nodo raíz de tree en la dimensión* x **then**
    `searchNodes` (*node, árbol derecho de tree, depth+1, distance, indexList, k*)
  end

Algoritmo 3.4: Cluster por distancia en KD-tree

Gracias a la estructura KD-tree, se puede reducir la cantidad de nodos o puntos analizados, ya que cada punto no tiene que ser estudiado con el resto sino que solo se estudian los puntos que están una región cercana dentro del radio máximo de distancia definido. Lo que produce una complejidad de $O(n)$ para la construcción de la estructura más la complejidad $O(n * \log n)$ de la función de clustering por distancia, por lo que en total se tendría una mejora de complejidad de $O(n^2)$ a $O(n * \log n)$.

La eficiencia de esta estructura en ciertas tareas, ha producido que a pesar de ser una técnica del año 1975, se siga estudiando para su utilización junto a KNN [17], aumentar su rendimiento con datos preordenados [18] o la paralelización de su construcción y técnicas como KNN [19].

3.4 Filtrado previo y posterior a la detección

Tras la obtención de las detecciones por parte de los diversos algoritmos clásicos podemos encontrarnos ante diferentes problemas con dichas detecciones.

Estas pueden generar clústeres con pocos o demasiados puntos, lo que puede resultar en clústeres incorrectos. Aquellos con pocos puntos pueden identificar objetos lejanos u objetos que no son necesarios para el entendimiento de la escena, por otra parte, aquellas detecciones con muchos puntos pueden identificar camiones, vehículos de construcción o simplemente objetos muy cercanos, pero también es muy normal que las construcciones sean detectadas por lo que hay que filtrar tanto por un número máximo como mínimo de puntos para obtener mejores detecciones.

Otra práctica para el filtrado, es el ajuste a unos tamaños prefijados en todas las dimensiones, lo cual elimine aquellos objetos que no son similares a los vehículos que se desean detectar.

Estas técnicas de filtrado no solo se pueden utilizar tras la obtención de las detecciones, sino que la nube de puntos obtenida del LiDAR es posible filtrarla, para que así solo se trabaje con una región de interés, ya que a partir de cierta distancia las detecciones no van a ser muy precisas, para ello se puede filtrar por distancia al vehículo. Además, un filtrado que permita trabajar unicamente con la parte delantera y trasera del coche, aporta una reducción en el computo de los algoritmos, a la vez que se reducen las falsas detecciones.

Capítulo 4

Sistemas de percepción con LiDAR basados en Deep Learning

Si no conozco una cosa, la investigaré.

Louis Pasteur

4.1 Principales datasets

4.1.1 Kitti

4.1.1.1 Análisis de la estructura del GT y las PCLs

4.1.2 Waymo

4.1.3 nuScenes

4.1.3.1 nuScenes-devkit

4.1.4 Comparativa entre los diferentes datasets

4.2 Estado del arte en detección utilizando LiDAR

4.2.1 PointPillars

4.2.2 SECOND

4.2.3 PointRCNN

4.2.4 PV-RCNN

4.2.5 CBGS

4.3 OpenPCDet

Capítulo 5

Desarrollo realizado

La persistencia es muy importante. No debes renunciar
al menos que te veas obligado a renunciar.

Elon Musk

5.1 Estado del proyecto T4AC

5.1.1 ROS

5.1.2 Docker

5.1.3 Estructura del proyecto

5.2 Implementación en CARLA

5.2.1 CARLA

5.2.2 Funcionamiento del LiDAR en CARLA

5.2.3 Implementación del sistema clásico utilizando LiDAR

5.2.4 Implementación del sistema basado en Deep Learning utilizando LiDAR

5.3 Fusión sensorial

5.4 Vehículo del proyecto T4AC

5.5 Implementación sobre el vehículo T4AC

Capítulo 6

AD DevKit

La inspiración existe, pero tiene que encontrarte trabajando.

Picasso

6.1 Estado del arte en evaluación de vehículos autónomos

6.2 Obtención del ground truth

6.3 Evaluación de los modelos

Capítulo 7

Resultados obtenidos

Ninguna investigación humana puede ser llamada
ciencia real si no puede demostrarse matemáticamente.

Leonardo da Vinci

7.1 Análisis cuantitativo en Kitti

7.2 Análisis cuantitativo en nuScenes

7.3 Análisis cualitativo del modelo clásico en CARLA

7.4 Análisis cualitativo de CBGS en CARLA

7.4.1 Comparativa con PointPillars en CARLA

7.5 Análisis cuantitativo de CBGS en CARLA

7.6 Análisis cualitativo de CBGS sobre el vehículo T4AC

7.6.1 Comparativa con PointPillars sobre el vehículo T4AC

Métrica	**Resultado**
mAP	0.4474
mATE	0.3379
mASE	0.2598
mAOE	0.3156
mAVE	0.2886
mAAE	0.2025
NDS	0.5832

Tabla 7.1: Rendimiento medio de CBGS PointPillars Multihead en nuScenes.

Tipo de objeto	AP	ATE	ASE	AOE	AVE	AAE
car	0.812	0.189	0.154	0.123	0.664	0.269
truck	0.500	0.354	0.189	0.093	0.415	0.277
bus	0.634	0.367	0.183	0.048	0.869	0.380
trailer	0.352	0.606	0.208	0.396	0.281	0.183
construction_vehicle	0.121	0.761	0.453	0.785	0.123	0.332
pedestrian	0.723	0.167	0.277	0.394	0.440	0.270
motorcycle	0.300	0.229	0.243	0.454	0.988	0.324
bicycle	0.064	0.189	0.273	0.506	0.494	0.093
traffic_cone	0.472	0.182	0.328	nan	nan	nan
barrier	0.499	0.341	0.288	0.071	nan	nan

Tabla 7.2: Análisis por clase de CBGS PointPillars Multihead en nuScenes.

Tipo de objeto	AP	IoU	AVE
Unknown	0.0	0.0	0.0
Unknown_Small	0.0	0.0	0.0
Unknown_Medium	0.0	0.0	0.0
Unknown_Big	0.0	0.0	0.0
Pedestrian	0.0	0.0	0.0
Bike	0.0	0.0	0.0
Car	0.731	0.491	0.715
Truck	0.0	0.0	0.0
Motorcycle	0.0	0.0	0.0
Other_Vehicle	0.0	0.0	0.0
Barrier	0.0	0.0	0.0
Sign	0.0	0.0	0.0

Tabla 7.3: Análisis por clase de CBGS PointPillars Multihead en CARLA.

Capítulo 8

Conclusiones

La verdadera felicidad radica en la finalización del trabajo utilizando tu propio cerebro y habilidades.

Soichiro Honda

8.1 Modelos estudiados

8.2 Comparativas adicionales

8.2.1 Ajuste de modelos basados en Kitti a nuScenes

8.2.2 Número de PCL de entrada en modelos evaluados sobre nuScenes

8.2.3 Tamaño del voxel en modelos basados en redes neuronales

8.3 Futuros trabajos

Bibliografía

[1] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021.

[2] "Automated vehicles for safety," Tech. Rep., 2018. [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

[3] "How self-driving cars work: Sensor systems," Tech. Rep., 2021. [Online]. Available: https://www.udacity.com/blog/2021/03/how-self-driving-cars-work-sensor-systems.html

[4] Y. Zhang, J. Ran, X. Chen, K. Fang, and H. Chen, "Observation of the inverse, zero and normal doppler effect in configurable transmission lines," in *2015 IEEE 4th Asia-Pacific Conference on Antennas and Propagation (APCAP)*, 2015, pp. 229–230.

[5] *HDL-64E, High Definition Real-Time 3D Lidar*, Velodyne Lidar. [Online]. Available: https://velodynelidar.com/products/hdl-64e/

[6] A. M. Wallace, A. Halimi, and G. S. Buller, "Full waveform lidar for adverse weather conditions," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7064–7077, 2020.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[8] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 689–12 697.

[9] F. Lotfi, V. Ajallooeian, and H. D. Taghirad, "Robust object tracking based on recurrent neural networks," in *2018 6th RSI International Conference on Robotics and Mechatronics (IcRoM)*, 2018, pp. 507–511.

[10] S. Wang, Y. Sun, C. Liu, and M. Liu, "Pointtracknet: An end-to-end network for 3-d object detection and tracking from point clouds," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3206–3212, 2020.

[11] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-balanced grouping and sampling for point cloud 3d object detection," 2019.

[12] T.-L. Kim, J.-S. Lee, and T.-H. Park, "Fusing lidar, radar, and camera using extended kalman filter for estimating the forward position of vehicles," in *2019 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2019, pp. 374–379.

[13] J. del Egido Sierra, "Detección del entorno 360º de un vehículo autónomo mediante lidar aplicando técnicas deep learning," Trabajo de Fin de Master, UAH Politécnica, 2020.

[14] M. Antunes, "Sistema de visión estereo aplicado a la detección y seguimiento de objetos en conducción autonoma," Trabajo de Fin de Grado, UAH Politécnica, 2021.

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[16] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[17] W. Hou, D. Li, C. Xu, H. Zhang, and T. Li, "An advanced k nearest neighbor classification algorithm based on kd-tree," in *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, 2018, pp. 902–905.

[18] Y. Cao, X. Zhang, B. Duan, W. Zhao, and H. Wang, "An improved method to build the kd tree based on presorted results," in *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, 2020, pp. 71–75.

[19] L. Hu, S. Nooshabadi, and M. Ahmadi, "Massively parallel kd-tree construction and nearest neighbor search algorithms," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2752–2755.

[20] "Información sobre gnu/linux en wikipedia," http://es.wikipedia.org/wiki/GNU/Linux [Último acceso 1/noviembre/2013].

[21] "Página de la aplicación emacs," http://savannah.gnu.org/projects/emacs/ [Último acceso 1/noviembre/2013].

[22] "Página de la aplicación kdevelop," http://www.kdevelop.org [Último acceso 1/noviembre/2013].

[23] L. Lamport, *LaTeX: A Document Preparation System, 2nd edition.* Addison Wesley Professional, 1994.

[24] "Página de la aplicación octave," http://www.octave.org [Último acceso 1/noviembre/2013].

[25] "Página de la aplicación cvs," http://savannah.nongnu.org/projects/cvs/ [Último acceso 1/noviembre/2013].

[26] "Página de la aplicación gcc," http://savannah.gnu.org/projects/gcc/ [Último acceso 1/noviembre/2013].

[27] "Página de la aplicación make," http://savannah.gnu.org/projects/make/ [Último acceso 1/noviembre/2013].

Apéndice A

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo GNU/Linux [20]
- Entorno de desarrollo Emacs [21]
- Entorno de desarrollo KDevelop [22]
- Procesador de textos LaTeX[23]
- Lenguaje de procesamiento matemático Octave [24]
- Control de versiones CVS [25]
- Compilador C/C++ gcc [26]
- Gestor de compilaciones make [27]

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá