

TRABAJO FIN DE GRADO INGENIERÍA Y SISTEMAS DE DATOS

TÍTULO: Sistema de percepción basado en LiDAR para detección y clasificación de conos en vehículos autónomos de competición

AUTOR/A: Eduardo Esteban Cases

TUTOR/A: David Luengo García

DEPARTAMENTO: Departamento de Ingeniería Audiovisual y Comunicaciones

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Javier Malo Gómez

TUTOR/A: David Luengo García

SECRETARIO/A: César Díaz Martín

Fecha de lectura: 29/09/2025

Calificación:

El Secretario/La Secretaria,

Agradecimientos

Quiero agradecer a mi familia por todo el apoyo que me dan y por estar siempre a mi lado. Su confianza ha sido fundamental para poder afrontar este trabajo.

A mis amigos y compañeros de carrera, por acompañarme en todo momento y aportar la motivación y la energía necesarias dentro y fuera del ámbito universitario.

A todos los miembros de UPM Racing, un equipo que representa mucho más que un proyecto universitario y que para mí se ha convertido en una familia de la que me siento orgulloso de haber formado parte.

En especial a mi división Driverless, con quienes he tenido la oportunidad de vivir una de las experiencias más intensas y enriquecedoras de mi vida universitaria, desde modificar un carrito de herramientas para convertirlo en un vehículo autónomo hasta sobrevivir a base de meal deals en el box del circuito de Silverstone mientras un Fórmula 1 rodaba a escasos metros.

El compromiso y la pasión de cada uno de sus miembros han hecho posible alcanzar metas que parecían inalcanzables al inicio del camino.

Finalmente, me gustaría agradecer la labor de mi tutor David, quien no solo me ha guiado en este proyecto, sino que también ha acompañado mis cuatro años de carrera como profesor. Sus enseñanzas han marcado de manera decisiva mi formación, y sus consejos han ayudado a enfocar este proyecto de la mejor manera posible.

Resumen

El presente Trabajo Fin de Grado, titulado *Sistema de percepción basado en LiDAR para detección y clasificación de conos en vehículos autónomos de competición*, aborda el diseño e implementación de un sistema de percepción basado exclusivamente en tecnología LiDAR. El objetivo es dotar a un vehículo de la competición *Formula Student UK* de un subsistema robusto y eficiente que garantice la detección y clasificación de conos en tiempo real, eliminando la dependencia de cámaras u otros sensores más frágiles y sensibles a condiciones ambientales adversas.

El desarrollo del sistema se ha visto condicionado por requisitos tecnológicos, económicos y normativos. Desde el punto de vista tecnológico, el procesado debía completarse en menos de 100 ms para cumplir con la frecuencia de publicación de un sensor Ouster OS1-32 a 10 Hz, capaz de generar nubes de puntos de más de 65 000 elementos. En lo económico, se priorizó el uso de herramientas libres como *ROS 2 Humble*, *Python*, *scikit-learn* y *XGBoost*, minimizando costes de licencias. Asimismo, se respetaron las normativas de *Formula Student Driverless*, asegurando la trazabilidad y la seguridad funcional del sistema.

La metodología se estructuró en una *pipeline* modular formada por cuatro etapas principales: filtrado inicial de la nube de puntos mediante restricciones geométricas; segmentación del plano del suelo con *RANSAC*; *clustering* de objetos con *DBSCAN*; y clasificación de conos mediante un modelo *XGBoost* entrenado sobre vectores de reflectividad. La implementación en *ROS 2* permitió comunicación asíncrona entre nodos, parametrización dinámica y validación tanto en simulación como en pruebas reales en la pista de ensayos del INSIA.

Los resultados experimentales muestran un salto cualitativo respecto a un sistema anterior existente: el tiempo máximo por *frame* se redujo de 604 ms a menos de 93 ms, ajustándose a la cadencia del sensor; los falsos positivos disminuyeron en más de un 99,99%; y la precisión en la clasificación de color alcanzó un 99,8 %. Estas mejoras se comprobaron en condiciones reales de pista, demostrando fiabilidad incluso bajo iluminación variable y presencia de ruido en la nube de puntos.

En conclusión, se presenta un sistema de percepción autónomo, robusto y eficiente, capaz de funcionar únicamente con LiDAR en tiempo real. Su implementación refuerza la competitividad del equipo UPM Racing en *Formula Student* y aporta una solución extrapolable a escenarios más amplios de conducción autónoma, donde la robustez y la simplicidad de la arquitectura son factores clave.

Abstract

This Bachelor Thesis, entitled *LiDAR-based perception system for cone detection and classification in autonomous racing vehicles*, addresses the design and implementation of a perception system based exclusively on LiDAR technology. The objective is to equip a vehicle competing in *Formula Student UK* with a robust and efficient subsystem that ensures real-time cone detection and classification, eliminating the dependence on cameras or other fragile sensors sensitive to adverse environmental conditions.

The development of the system was constrained by technological, economic, and regulatory requirements. From a technological perspective, the processing had to be completed in less than 100 ms to comply with the publication frequency of an Ouster OS1-32 LiDAR at 10 Hz, capable of generating point clouds with more than 65,000 elements. On the economic side, the use of open-source tools such as *ROS 2 Humble*, *Python*, *scikit-learn*, and *XGBoost* was prioritized, minimizing licensing costs. Furthermore, the design adhered to the regulations of *Formula Student Driverless*, ensuring traceability and functional safety.

The methodology was structured into a modular *pipeline* composed of four main stages: initial point cloud filtering through geometric constraints; ground plane segmentation using *RANSAC*; object clustering with *DBSCAN*; and cone classification using an *XGBoost* model trained on reflectivity vectors. The implementation in *ROS 2* enabled asynchronous communication between nodes, dynamic parameter tuning, and validation both in simulation and in real-world tests at the INSIA proving ground.

The experimental results demonstrate a qualitative leap compared to an existing previous system: the maximum processing time per *frame* was reduced from 604 ms to less than 93 ms, in line with the sensor's frequency; false positives decreased by more than 99.99%; and color classification accuracy reached 99.8%. These improvements were validated in real track conditions, proving reliability even under variable lighting and noisy point clouds.

In conclusion, this work presents an autonomous, robust, and efficient perception system capable of operating solely with LiDAR in real-time. Its implementation strengthens UPM Racing's competitiveness in *Formula Student* and provides a solution transferable to broader autonomous driving scenarios, where robustness and architectural simplicity are key factors.

Índice

Agradecimientos	I
Resumen	III
Abstract	V
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del documento	2
2. Contexto	5
2.1. Formula Student	5
2.2. UPM Racing	6
2.3. División <i>Driverless</i>	7
2.4. ROS 2	12
2.4.1. Modelo de comunicación	12
2.4.2. Calidad de servicio y middleware	12
2.4.3. Herramientas de desarrollo y validación	12
2.4.4. Ventajas en el proyecto Driverless	13
2.5. Competición Formula Student UK	13
2.5.1. Pruebas Estáticas	14
2.5.2. Pruebas Dinámicas	17
3. Estado del arte	23
3.1. Percepción en vehículos autónomos	23
3.2. Comparativa entre sensores exteroceptivos	23
3.2.1. Cámaras	24
3.2.2. LiDAR	25
3.2.3. Radar	27
3.2.4. Elección del sensor LiDAR	28
3.3. Procesamiento de nubes de puntos	28
3.3.1. Eliminación de ruido	29

3.3.2.	Reducción de la densidad de puntos	30
3.3.3.	Segmentación estructural	31
3.4.	Clustering en nubes de puntos	33
3.4.1.	Euclidean Cluster Extraction	33
3.4.2.	K-means	34
3.4.3.	Density-Based Spatial Clustering of Applications with Noise (DBSCAN) . .	34
3.4.4.	Clustering jerárquico	35
3.4.5.	Comparativa entre métodos:	36
3.5.	Clasificación de objetos	36
3.5.1.	Filtros geométricos basados en reglas	36
3.5.2.	Métodos clásicos sobre descriptores	37
3.5.3.	Aprendizaje profundo sobre nubes de puntos	39
3.6.	Caso de estudio: sistema de percepción de TU Dresden (FSG 2024)	40
4.	Desarrollo del proyecto	43
4.1.	Introducción al desarrollo	43
4.1.1.	Contexto en Fórmula Student	43
4.2.	Adquisición de datos	44
4.2.1.	Sensor LiDAR	44
4.2.2.	Formato de los datos	45
4.3.	Punto de partida	47
4.3.1.	Estado inicial del sistema de percepción	47
4.3.2.	Objetivos del nuevo desarrollo	48
4.4.	Solución propuesta	49
4.4.1.	Arquitectura	49
4.4.2.	Filtrado inicial de la nube de puntos	51
4.4.3.	Segmentación de la nube de puntos	52
4.4.4.	<i>Clustering</i> en la nube de puntos	56
4.4.5.	Clasificación de color basada en datos de reflectividad	62
5.	Resultados	67
5.1.	Metodología de pruebas	67
5.1.1.	Pruebas cualitativas	68

<i>ÍNDICE</i>	IX
5.1.2. Pruebas cuantitativas	69
5.2. Eficiencia en la reducción de puntos	70
5.3. Detección de conos	71
5.4. Clasificación de color	76
5.4.1. Comparativa de modelos candidatos	76
5.4.2. Rendimiento del clasificador XGBoost	77
5.4.3. Comparación final con Random Forest	77
5.4.4. Pruebas en entornos reales	79
5.5. Rendimiento temporal	82
5.5.1. Tiempos globales de ejecución	84
5.6. Cumplimiento de objetivos	87
5.6.1. Limitaciones del sistema	88
6. Presupuesto	89
6.1. Costes de hardware	89
6.2. Costes de software	89
6.3. Costes de personal	90
6.4. Resumen del presupuesto	90
7. Impacto del proyecto	93
7.1. Impacto social	93
7.2. Impacto en salud y seguridad	93
7.3. Impacto ambiental	93
7.4. Impacto económico	93
7.5. Impacto tecnológico e industrial	94
7.6. Contribución a los Objetivos de Desarrollo Sostenible	94
8. Conclusiones y Lineas Futuras	95
8.1. Conclusiones	95
8.2. Líneas futuras	96
Bibliografía	99
Anexos	102

X

ÍNDICE

A.

103

A.1. Información recogida en las pruebas 103

Índice de tablas

2.1. Distribución de puntuaciones en las pruebas estáticas (FS-AI).	14
2.2. Distribución de puntuaciones en las pruebas dinámicas (FS-AI).	17
4.1. Avance aproximado del vehículo en 0.1 s para distintas velocidades.	46
5.1. Comparación del número de conos correctamente detectados, falsos positivos y precisión en los dos sistemas (450 <i>frames</i>).	73
5.2. Distribución de detecciones de conos reales por intervalos de distancia	75
5.3. Espacio de búsqueda y mejores hiperparámetros por modelo (selección por F1 en CV). 76	
5.4. Comparación de modelos candidatos para la clasificación de color (validación cruzada). 76	
5.5. Matriz de confusión en el conjunto de test (XGBoost).	77
5.6. Comparativa de rendimiento entre XGBoost y <i>Random Forest</i> en el conjunto completo. 78	
5.7. Matriz de confusión con el modelo final XGBoost (conjunto completo).	78
5.8. Matriz de confusión con el modelo <i>Random Forest</i> (conjunto completo).	78
5.9. Resultados de la clasificación de color en las pruebas en la pista del INSIA, incluyendo la precisión considerando como fallo tanto los errores como los <i>unknowns</i> . . .	79
5.10. Rendimiento medio de la segmentación RANSAC en función del número de iteraciones. 82	
5.11. Tiempos medios de ejecución por etapa para conos amarillos [ms].	84
5.12. Tiempos medios de ejecución por etapa para conos azules [ms].	84
5.13. Peor caso (entre amarillos y azules) para el sistema original.	85
5.14. Peor caso (entre amarillos y azules) para el sistema con XGB	86
5.15. Avance lineal aproximado del vehículo durante el tiempo máximo de procesado de cada sistema.	86
6.1. Costes de <i>hardware</i> empleados en el desarrollo.	89
6.2. Costes de <i>software</i> empleados en el desarrollo (anualizados).	90
6.3. Costes de personal estimados.	90
6.4. Presupuesto global del proyecto.	90
A.1. Campos registrados durante las pruebas cuantitativas del sistema de percepción. .	103

Índice de figuras

2.1. Conos utilizados en la Fórmula Student Driverless (Normativa de la competición FSG).	8
2.2. Disposición de los sistemas de Driverless (Imagen del <i>Autonomous Design Event</i> (Sección 2.5.1.3) de <i>UPM Racing</i> en FSUK 2025).	9
2.3. Prueba Skidpad (Sección 2.5.2.2) en el simulador de EUFS.	10
2.4. Prueba en entorno real con el Mitsubishi i-MiEV modificado del INSIA.	11
2.5. ADS-DV con los sensores de <i>UPM Racing</i> instalados (FSUK 2024)	14
2.6. Niveles de autonomía de un vehículo definidos por la SAE (<i>SAE International</i> 2016)	15
2.7. Disposición de la prueba Skidpad (<i>FSUK Rules</i> 2025).	19
2.8. Disposición de la prueba <i>Acceleration</i> (<i>FSG Handbook</i> 2025).	20
2.9. Disposición de la prueba <i>Trackdrive</i> (<i>FSG Competition Handbook</i> 2018)	22
3.1. Ejemplo de configuración de cámaras estereoscópicas (OpenCV).	25
3.2. Esquema de funcionamiento de un sensor LiDAR basado en tiempo de vuelo (ToF) [22].	26
3.3. Ejemplo de Voxel Grid Filtering en una nube de puntos [27]. La imagen (a) muestra la nube de puntos original y la imagen (b), la nube procesada.	31
3.4. Ejemplo de modelo lineal ajustado por RANSAC (Wikipedia).	32
3.5. Ejemplo de <i>dataset</i> agrupado por DBSCAN con $m = 4$ (Wikipedia).	35
4.1. Sensor Ouster OS1-32 (ouster.com)	44
4.2. Comparación de los modos de resolución disponibles en el LiDAR Ouster OS1-32. .	45
4.3. Esquema de comunicación entre nodos en el sistema de percepción inicial (captura de <code>rqt_graph</code>).	47
4.4. Arquitectura propuesta basada en una única API.	50
4.5. Resultado del filtrado inicial aplicado a la nube de puntos del LiDAR.	52
4.6. Ejemplo de puntos fantasma (<i>ghost points</i>) bajo el suelo debido a reflexiones múltiples en una superficie reflectante.	54
4.7. Resultado del autonivelado en la nube de puntos.	56
4.8. Resultado de RANSAC aplicado a la nube de puntos.	56
4.9. Resultado de la agrupación: nube original tras la segmentación (arriba) y agrupamiento de puntos en <i>clusters</i> (abajo).	58
4.10. Ejemplo de separación de <i>cluster</i> con dos conos cercanos.	59

4.11. Recuperación de puntos eliminados en segmentación: <i>clusters</i> originales tras RAN-SAC (arriba) y <i>clusters</i> enriquecidos con la base recuperada (abajo).	60
4.12. Resultado de la etapa de <i>clustering</i> : conos detectados en la nube de puntos.	62
4.13. Ejemplo de malla obtenida con la proyección (y, z) . Se observa cómo la forma queda deformada cuando el cono no es visto de frente.	63
4.14. Ejemplos de mallas de reflectividad: cono azul (izquierda) y cono amarillo (derecha).	64
4.15. Comparación de la nube de puntos original y después del procesamiento completo.	65
5.1. Pista de ensayos del INSIA.	67
5.2. Carro de pruebas adaptado para la instalación de los sensores.	68
5.3. Escenario para extraer datos de los resultados del sistema.	69
5.4. Comparación del nuevo sistema y el original en la evolución del número medio de puntos y conos a lo largo de las etapas del <i>pipeline</i>	70
5.5. Evolución temporal del número de puntos tras el prefiltrado y la segmentación. Comparativa entre el sistema original y el desarrollado.	71
5.6. Evolución temporal del número de conos detectados por <i>frame</i> en cada sistema.	72
5.7. Evolución temporal del número de conos detectados, separando correctos y falsos positivos.	72
5.8. Distribución del número de conos reales detectados por <i>frame</i> en cada sistema.	73
5.9. Acumulado espacial de todas las detecciones.	74
5.10. Ejemplo de falsos positivos en el sistema original.	75
5.11. Distribución de nº de conos mal clasificados por <i>frame</i>	80
5.12. Porcentaje de aciertos, fallos y <i>Unknowns</i> en función de la distancia de detección.	81
5.13. Influencia del número de iteraciones de RANSAC en el tiempo de ejecución y en la clasificación de puntos como <i>inliers</i> o <i>outliers</i>	83
5.14. Tiempo acumulado de ejecución por etapa del <i>pipeline</i> para conos amarillos (izquierda) y azules (derecha).	84
5.15. Evolución temporal del tiempo de procesamiento acumulado en cada etapa del sistema original (izquierda) y del sistema desarrollado con <i>XGBoost</i> (derecha).	85

Lista de acrónimos

Acrónimo	Descripción
ABS	<i>Anti-lock Braking System</i>
ADAS	<i>Advanced Driver Assistance Systems</i>
ADS	<i>Autonomous Driving System</i>
ADS-DV	<i>Autonomous Driving System – Driverless Vehicle</i> (vehículo de pruebas FSUK)
API	<i>Application Programming Interface</i>
AS	<i>Autonomous System</i> (estados AS Emergency / AS Finished en FS-AI)
AUC	<i>Area Under the Curve</i>
BMS	<i>Battery Management System</i>
CFD	<i>Computational Fluid Dynamics</i>
CI/CD	<i>Continuous Integration / Continuous Deployment</i>
CMOS	<i>Complementary Metal–Oxide–Semiconductor</i>
CPU	<i>Central Processing Unit</i>
CUDA	<i>Compute Unified Device Architecture</i> (NVIDIA)
CV	<i>Cross-Validation</i>
DBSCAN	<i>Density-Based Spatial Clustering of Applications with Noise</i>
DDS	<i>Data Distribution Service</i>
DDR4	<i>Double Data Rate 4</i>
EBS	<i>Emergency Brake System</i>
ECU	<i>Engine Control Unit</i>
EKF	<i>Extended Kalman Filter</i>
EUFS	<i>Edinburgh University Formula Student</i> (simulador)
FEM	<i>Finite Element Method</i>
FOV	<i>Field of View</i>
FS	<i>Formula Student</i>
FSUK	<i>Formula Student United Kingdom</i>
GNSS	<i>Global Navigation Satellite System</i>
GPU	<i>Graphics Processing Unit</i>
GPS	<i>Global Positioning System</i>
HIL	<i>Hardware-In-the-Loop</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IMU	<i>Inertial Measurement Unit</i>
INS	<i>Inertial Navigation System</i>

Acrónimo	Descripción
INSIA	Instituto Universitario de Investigación del Automóvil (UPM)
IP (IP68/IP69K)	<i>Ingress Protection</i> (grado de protección)
ISO	<i>International Organization for Standardization</i>
KD-Tree	<i>K-Dimensional Tree</i> (estructura de datos)
LDWS	<i>Lane Departure Warning System</i>
LiDAR	<i>Light Detection And Ranging</i>
LogReg	<i>Logistic Regression</i> (Regresión Logística)
MPC	<i>Model Predictive Control</i>
PCA	<i>Principal Component Analysis</i>
PCB	<i>Printed Circuit Board</i>
PR-AUC	<i>Precision–Recall Area Under the Curve</i>
QoS	<i>Quality of Service</i>
RAM	<i>Random Access Memory</i>
RANSAC	<i>RANdom SAMple Consensus</i>
RBF	<i>Radial Basis Function</i> (kernel)
RES	<i>Remote Emergency Stop</i>
RF	<i>Random Forest</i>
RGB	<i>Red, Green, Blue</i>
ROC-AUC	<i>Receiver Operating Characteristic – Area Under the Curve</i>
ROS 2	<i>Robot Operating System 2</i>
RRT	<i>Rapidly-exploring Random Trees</i>
RTK	<i>Real-Time Kinematic</i>
RViz	Visualizador 3D de ROS
SAE	<i>Society of Automotive Engineers</i>
SIL	<i>Software-In-the-Loop</i>
SLAM	<i>Simultaneous Localization And Mapping</i>
SOTIF	<i>Safety Of The Intended Functionality</i> (ISO/PAS 21448)
SVM	<i>Support Vector Machine</i>
TFG	Trabajo Fin de Grado
ToF	<i>Time of Flight</i>
UPM	Universidad Politécnica de Madrid
V2I	<i>Vehicle-to-Infrastructure</i>
V2V	<i>Vehicle-to-Vehicle</i>
VCU	<i>Vehicle Control Unit</i>
XGB	Abreviatura usada para <i>XGBoost</i> (<i>Extreme Gradient Boosting</i>)

Acrónimo	Descripción
YOLO	<i>You Only Look Once</i> (detección en visión)

Capítulo 1

Introducción

El presente Trabajo Fin de Grado aborda el desarrollo de un sistema de percepción para un vehículo autónomo de *Formula Student*. El objetivo principal es diseñar un *pipeline*, es decir, una cadena de etapas de procesamiento basada exclusivamente en datos de un sensor LiDAR, capaz de detectar y clasificar los conos que delimitan el circuito de manera robusta, eficiente y en tiempo real.

El sistema desarrollado se ha integrado en el *stack Driverless* del vehículo y se ha validado tanto en simulación como en condiciones reales de pista. Los avances en la simplicidad del despliegue, el aumento de la distancia de detección, la reducción de falsos positivos y la mayor precisión en la clasificación de color de los conos contribuyeron de manera directa al rendimiento competitivo del equipo *UPM Racing*, de la Universidad Politécnica de Madrid, que alcanzó un destacado 4^o y 6^o puesto de 20 y 26 equipos participantes respectivamente en las dos últimas ediciones celebradas en el circuito de Silverstone, quedando en primer lugar entre los equipos españoles de ambas ediciones.

1.1. Motivación

La conducción autónoma se ha consolidado en los últimos años como uno de los ámbitos de investigación más relevantes dentro del sector de la automoción. Su potencial para mejorar la seguridad vial, incrementar la eficiencia del transporte y reducir el impacto medioambiental ha motivado tanto a la industria como al ámbito académico a invertir de forma intensiva en esta tecnología. Su desarrollo se sustenta en tres pilares fundamentales: percepción del entorno, planificación de trayectorias y control del vehículo.

Dentro de estos pilares, la percepción ocupa un lugar central, pues constituye la base de la que parten los demás módulos. En la modalidad *Driverless* de *Formula Student*, esta tarea se traduce en la necesidad de detectar con fiabilidad los conos que delimitan el circuito, ya que son la única referencia disponible para que el vehículo pueda interpretar el trazado y tomar decisiones de navegación. La precisión y la robustez del sistema de percepción resultan, por tanto, determinantes no solo para completar el recorrido, sino también para garantizar un rendimiento competitivo frente al resto de equipos.

En sus primeras versiones, el sistema de percepción del vehículo desarrollado por el equipo *UPM Racing* se planteó como un esquema multisensorial, en el que las cámaras aportaban la clasificación por color y el LiDAR la localización espacial de los conos. Sin embargo, la experiencia en pista demostró que las cámaras resultaban sensibles a las condiciones de iluminación y propensas a fallos de *hardware*, lo que en diversas ocasiones dejaron al vehículo sin capacidad de operar de forma autónoma.

Ante esta situación, surgió la necesidad de eliminar la dependencia de una arquitectura multisensorial frágil y desarrollar un sistema de percepción capaz de operar únicamente con la información del LiDAR. Este planteamiento no solo perseguía garantizar el funcionamiento autónomo del vehículo, sino también simplificar la arquitectura y reducir la posibilidad de fallos críticos durante la competición.

La motivación principal de este Trabajo Fin de Grado es, por tanto, dotar al vehículo de un

sistema de percepción eficiente y fiable, capaz de detectar los conos y clasificar su color en tiempo real y de funcionar en condiciones de competición sin depender de otros sensores complementarios.

1.2. Objetivos

Los objetivos generales del proyecto se pueden resumir en tres metas principales:

- **Aumentar el alcance de detección:** lograr que el sistema sea capaz de identificar conos a mayores distancias, manteniendo la fiabilidad incluso cuando la densidad de puntos disminuye.
- **Reducir los falsos positivos:** disminuir la probabilidad de que estructuras del entorno (vegetación, bordes de pista o ruido del sensor) sean clasificadas erróneamente como conos.
- **Mejorar la clasificación de color:** incrementar la precisión en la identificación de conos azules, amarillos y naranjas (los principales conos utilizados en la competición), aspecto fundamental para que el módulo de planificación pueda distinguir correctamente los límites izquierdo y derecho del circuito.

Para alcanzar estos objetivos principales se definieron metas técnicas adicionales, orientadas a la implementación práctica del sistema:

- Reestructurar el *pipeline* de percepción en un sistema modular, con nodos claramente separados y parametrizables, facilitando el ajuste de algoritmos sin necesidad de recompilación.
- Optimizar la clasificación de color, pasando de reglas heurísticas básicas a métodos más generalizables, con capacidad de adaptación a distintos escenarios.
- Integrar herramientas de validación y depuración (Véase la sección 2.4.3) que faciliten el análisis del sistema tanto en RViz como a través de registros reproducibles (*rosviz*).

Por otra parte, existe un objetivo aparte, que consiste en favorecer la mantenibilidad y la rápida transferencia de conocimiento, estructurando el código de manera clara y documentada. Esto permite que nuevos miembros del equipo puedan comprender el funcionamiento del sistema en su totalidad en poco tiempo y comenzar a experimentar con modificaciones o mejoras sin una curva de aprendizaje excesiva, mitigando así el impacto de la rotación natural de estudiantes en el proyecto.

1.3. Estructura del documento

El documento se organiza en los siguientes capítulos:

- **Capítulo 2. Contexto:** describe la competición *Formula Student*, el equipo *UPM Racing* y su división *Driverless*, así como el marco tecnológico de referencia empleado en el proyecto.
- **Capítulo 3. Estado del arte:** revisa los principales sensores exteroceptivos, técnicas de procesamiento de nubes de puntos, algoritmos de *clustering* y métodos de clasificación aplicados en percepción para vehículos autónomos.

- **Capítulo 4. Desarrollo del proyecto:** expone el punto de partida del sistema, los objetivos específicos y la implementación del *pipeline* de percepción propuesta, detallando cada una de sus etapas.
- **Capítulo 5. Resultados:** presenta la metodología de validación, así como los resultados cualitativos y cuantitativos obtenidos en detección y clasificación de conos, junto con el análisis de rendimiento temporal.
- **Capítulo 6. Presupuesto:** recoge la estimación de costes de *hardware*, *software* y personal, así como el resumen económico global del proyecto.
- **Capítulo 7. Impacto del proyecto:** analiza las implicaciones sociales, de salud y seguridad, ambientales, económicas, tecnológicas e industriales, además de la contribución a los Objetivos de Desarrollo Sostenible.
- **Capítulo 8. Conclusiones y trabajos futuros:** sintetiza las aportaciones principales del trabajo, evalúa el grado de cumplimiento de los objetivos y plantea posibles líneas de mejora y desarrollo posterior.

Capítulo 2

Contexto

Desde la invención del motor de combustión interna en el siglo XIX, los automóviles han sido un pilar fundamental en la transformación de la sociedad moderna. En 1886, Carl Benz patentó su “Vehículo propulsado por un motor de combustión” [1], dando inicio a una era de innovación en el transporte personal y comercial. A lo largo del siglo XX, la industria automotriz evolucionó rápidamente, introduciendo avances como la producción en masa, perfeccionada por Henry Ford [2], y numerosos sistemas de asistencia a la conducción (*Advanced Driver Assistance Systems* (ADAS)), desde el Sistema Antibloqueo de Frenos (*Anti-lock Braking System* (ABS), 1928) [3] hasta avances más recientes, como el sistema de aviso de salida de carril (*Lane Departure Warning System* (LDWS), 1989) [4]. Estos desarrollos han llevado a una mejora continua en la eficiencia, seguridad y accesibilidad de los vehículos. Hoy en día, los automóviles no solo representan un medio de transporte esencial, sino que también simbolizan el progreso tecnológico.

En este creciente contexto, el desarrollo de vehículos autónomos se ha convertido en uno de los asuntos más relevantes de la actualidad. Estos vehículos cuentan con el potencial de revolucionar la movilidad, mejorar la seguridad vial y reducir el impacto ambiental. Por estos motivos, tanto la industria como la investigación académica tienen centrada su atención en mejorar su diseño. Dentro de este marco, este proyecto tiene el propósito de diseñar y desarrollar un sistema de percepción basado en tecnología LiDAR para un vehículo autónomo de *Formula Student*.

2.1. Formula Student

Formula Student (FS), también conocida como *Formula SAE* (*Society of Automotive Engineers*) [5], es una de las competiciones universitarias de ingeniería más reconocidas a nivel internacional. Su origen se remonta a 1981 en los Estados Unidos bajo la iniciativa de la SAE, y desde entonces se ha expandido globalmente, contando con ediciones en Europa, Asia, América y Oceanía. En esta competición, equipos multidisciplinarios de estudiantes diseñan, fabrican y compiten con un prototipo de monoplace.

El objetivo principal de *Formula Student* no es únicamente la competición en sí misma, sino el proceso formativo que conlleva. Los estudiantes se enfrentan a problemas reales de ingeniería que abarcan desde el diseño mecánico y eléctrico hasta la gestión de proyectos, finanzas y logística. De esta forma, se busca fomentar competencias técnicas y transversales que los acercan al entorno profesional de la automoción y la ingeniería de sistemas.

La evaluación de los equipos se realiza a través de pruebas estáticas y dinámicas. Las **pruebas estáticas** incluyen la defensa del diseño del vehículo, el análisis de costes y la presentación de un plan de negocio. En estas pruebas los participantes deben poder justificar sus decisiones ante un comité de jueces especializados del sector. Las **pruebas dinámicas**, por su parte, ponen a prueba el rendimiento del prototipo en condiciones reales de pista, incluyendo aceleración, resistencia, maniobrabilidad y eficiencia. Estas fases garantizan una visión integral de las capacidades técnicas y organizativas del equipo.

La magnitud y el alcance de la competición han crecido de forma notable desde su creación. En la primera edición de 1981, celebrada en la *University of Texas* (Austin, TX), se registraron únicamente 6 equipos, de los cuales sólo 4 participaron efectivamente, con la presencia de 40

estudiantes en total. En 1998 la competición tomó lugar por primera vez en Europa, con *Formula Student United Kingdom* (FSUK), en el circuito de pruebas de MIRA (*Motor Industry Research Association*), en Warwickshire, Reino Unido. En contraste, en 2018 la *Formula SAE* en el *Michigan International Speedway* (Brooklyn, MI) alcanzó los 120 equipos registrados (de los cuales 117 participaron) y reunió a más de 2500 estudiantes. Esta evolución refleja no solo el interés académico por la competición, sino también la relevancia que ha adquirido como plataforma de innovación y formación.

En la actualidad, *Formula Student* se ha convertido en un entorno de investigación y desarrollo puntero. Cada vez más, la atención se centra en los vehículos de propulsión 100 % eléctrica, así como en la exploración de otras alternativas sostenibles de movilidad. De este modo, la competición no solo contribuye a la formación práctica de los ingenieros del futuro, sino que también se alinea con la transición tecnológica y energética que atraviesa la industria automovilística.

2.2. UPM Racing

Fundado en 2003, *UPM Racing* [6], la propuesta de los alumnos de la Universidad Politécnica de Madrid [7] para participar en *Formula Student*, fue el primer representante español de la disciplina. En sus primeros años, el equipo compitió con monoplasas de combustión interna. En 2012 se abordó la transición a la propulsión eléctrica con el *UPM 01e*; y, desde 2017, con el *03e*, todos los vehículos del equipo son 100 % eléctricos.

En 2017, *Formula Student Germany* (FSG) [8] trajo una nueva categoría a la competición: por primera vez, los equipos fueron desafiados a participar con vehículos autónomos capaces de completar circuitos sin intervención humana. Con el fin de participar en esta nueva categoría, en 2019 en *UPM Racing* se fundó la división *Driverless*.

Actualmente, *UPM Racing* está compuesto por más de 60 estudiantes organizados en distintas divisiones especializadas:

- **Aerodinámica y Refrigeración:**

Se encarga del diseño, análisis y fabricación del conjunto aerodinámico y del sistema de refrigeración del monoplaza, optimizando los alerones y reduciendo la masa del *pack* aerodinámico mediante simulaciones CFD (*Computational Fluid Dynamics*) y FEM (*Finite Element Method*), mientras que en Refrigeración se evalúa la fiabilidad de los componentes, se calculan pérdidas de presión y se dimensionan radiadores mediante análisis de transferencia de calor; además, junto con Chasis, participan en la fabricación del monocasco y del *pack* aerodinámico en fibra de carbono.

- **Automotive Mechanics:**

Esta división integra las áreas de Transmisión y Suspensión, encargándose del diseño y montaje del tren de potencia, incluyendo engranajes, buje y mangueta, así como del desarrollo, fabricación y ensayo de los sistemas de doble horquilla, dirección y frenos, utilizando simulaciones de esfuerzo y optimización topológica combinadas con la producción de componentes en taller.

- **Chasis:**

La división de Chasis se ocupa del diseño y fabricación del monocasco para garantizar integridad estructural y protección del piloto, al tiempo que asegura la ergonomía mediante el diseño del asiento, volante y pedalera; todas las decisiones de diseño se validan mediante simulaciones y ensayos mecánicos que verifican las propiedades de los materiales empleados.

- **Dinámica:**

Esta división optimiza el comportamiento dinámico del vehículo mediante modelos multi-cuerpo en Simulink, calculando cargas en la suspensión, optimizando el *set-up* y gestionando el reparto de par mediante sistemas HIL (*hardware-In-the-Loop*); adicionalmente, en colaboración con *Automotive Mechanics*, supervisa el diseño, preparación y mantenimiento de frenos, dirección, neumáticos y suspensión.

- **Driverless:**

Desarrolla el *software* para la conducción autónoma, utilizando C++, Python y ROS sobre Linux con control de versiones mediante GIT. Se estructura en **Percepción**, que emplea un sensor LiDAR y cámaras estereoscópicas con redes neuronales para la detección de conos y obstáculos, y *Dinámica*, que abarca SLAM (*Simultaneous Localization and Mapping*, *Path-Planning* y *Control* para generar mapas virtuales, trayectorias y comandos de actuación para el monoplaza.

- **High Voltage:**

Esta división gestiona la batería de alta tensión, inversores, motores y cableado, incluyendo montaje, mantenimiento, desarrollo del BMS (*Battery Management System*) propio y programación de la ECU (*Engine Control Unit*) para asegurar la seguridad eléctrica, control de sistemas y puesta a punto del vehículo, colaborando con *Low Voltage* en el diseño 3D e implementación del cableado y sus componentes.

- **Low Voltage:**

Se encarga del diseño y mantenimiento de la electrónica, la batería de baja tensión y el cableado del vehículo, fabricando y probando PCB (*Printed Circuit Board*), programando microcontroladores, dimensionando la batería y trabajando junto con *High Voltage* en el diseño 3D del cableado y su implementación.

- **Marketing, Business & Cost:**

Esta división gestiona las pruebas de *Cost & Manufacturing*, el desarrollo del *Business Plan* y las actividades de *marketing*, analizando costes y procesos de fabricación, elaborando y defendiendo planes de negocio ante potenciales inversores, y gestionando la visibilidad del proyecto mediante búsqueda de patrocinadores y presencia en redes sociales.

Esta estructura fomenta el trabajo multidisciplinar y la integración de conocimientos, permitiendo a los estudiantes abordar de forma coordinada todos los aspectos del diseño, fabricación y validación de un vehículo de competición. Además, el equipo se caracteriza por una rotación natural de sus integrantes cada uno o dos años (en raras ocasiones tres), a medida que los alumnos finalizan sus estudios. Este relevo constante hace especialmente relevante mantener una organización clara y una documentación rigurosa, de modo que los nuevos miembros puedan adaptarse rápidamente y continuar con el desarrollo del proyecto sin pérdida de conocimiento acumulado.

2.3. División *Driverless*

La división *Driverless* estructura su desarrollo en dos grandes *stacks*: **Percepción** y **Dinámica**.

Percepción: El objetivo principal de este *stack* es extraer información fiable del entorno a partir de distintos sensores. En *Formula Student Driverless*, esta tarea se centra en la detección de conos, que son la única referencia disponible para delimitar el trazado del circuito. Existen cuatro tipos reglamentarios de conos:

- **azules** para marcar el límite izquierdo del circuito
- **amarillos** para el derecho

- **naranjas grandes** que indican el inicio del recorrido
- **naranjas pequeños** para señalar cambios específicos de trazado y zonas especiales

Estos cuatro tipos de conos se muestran en la Figura 2.1. Una percepción robusta debe localizar su posición y clasificar correctamente su color, ya que de ello depende la planificación posterior.



Figura 2.1: Conos utilizados en la Fórmula Student Driverless (Normativa de la competición FSG).

Este *stack* se compone de los siguientes módulos:

- **Cámaras:** Mediante cámaras estereoscópicas y el uso de modelos de visión artificial, como YOLOv9 [9], este sistema detecta los conos que delimitan el circuito, clasificando su color y estimando su posición aproximada en el espacio.
- **LiDAR:** Basado en un sensor LiDAR, procesa nubes de puntos en 3D para identificar y localizar los conos de forma precisa. Se emplean algoritmos de *clustering* para agrupar puntos correspondientes a un mismo objeto y extraer sus coordenadas en el entorno.
- **Fusión:** Con el fin de combinar las ventajas de cámaras y LiDAR, se ha desarrollado un método de calibración apoyado en un sistema de ecuaciones no lineales. Esto permite integrar color y posición de cada cono en una representación unificada, aumentando la robustez del sistema frente a ruidos o pérdidas parciales de detección.

Dinámica: Este segundo *stack* transforma la información proporcionada por percepción en mapas, trayectorias y órdenes de control que permiten al vehículo completar un circuito sin intervención humana.

Los sistemas que integran este *stack* son:

- **SLAM (*Simultaneous Localization and Mapping*):**

Permite al vehículo construir un mapa global del circuito al mismo tiempo que estima su posición dentro de él. Se han explorado técnicas como FastSLAM y filtros de Kalman extendido (EKF), combinadas con datos de GPS e IMU [10].

- **Path Planning:**

Una vez generado el mapa, este sistema calcula la trazada óptima que el vehículo debe seguir. Se utilizan métodos como la triangulación de Delaunay para definir el espacio de navegación, junto con algoritmos de exploración como RRT (*Rapidly-exploring Random Trees*) para hallar trayectorias viables.

■ Control:

Convierte la trayectoria planificada en acciones concretas sobre los actuadores del monoplaza (dirección, aceleración y frenado). Se emplean enfoques como el controlador Stanley, orientado al guiado lateral, y técnicas de *Model Predictive Control* (MPC) para optimizar la estabilidad y precisión en el seguimiento de la trayectoria.

Los sensores en los que se apoya *Driverless* para conformar la base de la percepción y la estimación de estado del vehículo son los siguientes:

- **LiDAR:** Ouster OS1-32 [11], proporciona nubes de puntos tridimensionales de alta resolución.
- **Cámaras:** Stereolabs ZED 2i [12], cámaras estereoscópicas capaces de generar imágenes RGB y estimar profundidad.
- **IMU y GPS:** XSENS MTi-680 RTK [13], encargada de medir aceleraciones y velocidades angulares con gran precisión. Además, integra un receptor GPS de alta precisión para la localización absoluta del vehículo.

La arquitectura del sistema se articula en torno al modelo de publicación y suscripción de ROS 2 [14], donde la comunicación entre módulos se lleva a cabo mediante *topics*. Los nodos de procesamiento asociados a las cámaras y al LiDAR publican la información extraída de los datos sensoriales hacia el módulo de fusión, que integra ambas fuentes y transmite los resultados al sistema de *SLAM*. Este último genera un mapa del circuito y localiza en él tanto el vehículo como los conos, combinando además la información de la IMU y el GPS para mejorar la estimación de la posición. A partir de esta representación global, el módulo de *path planning* calcula una trayectoria óptima en función de la disposición de los conos y de la localización estimada del coche. Finalmente, el módulo de control utiliza tanto la trayectoria planificada como la posición del vehículo para generar las señales adecuadas hacia los actuadores, de forma que el monoplaza sea capaz de seguir la trayectoria definida. La Figura 2.2 ilustra de manera esquemática esta arquitectura jerárquica de percepción, planificación y control.

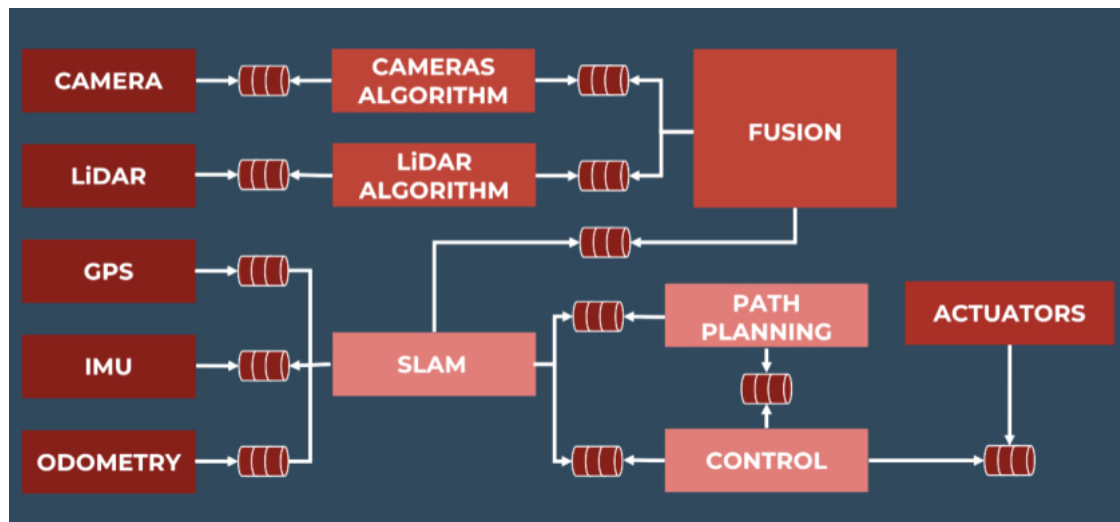


Figura 2.2: Disposición de los sistemas de Driverless (Imagen del *Autonomous Design Event* (Sección 2.5.1.3) de *UPM Racing* en FSUK 2025).

Con el fin de validar el correcto funcionamiento de la arquitectura, se emplea un simulador desarrollado por estudiantes del equipo de *Formula Student* de la Universidad de Edimburgo,

EUFS [15], que permite reproducir escenarios de competición de manera controlada y segura. Este entorno virtual constituye una herramienta de gran valor, ya que posibilita ensayar la interacción entre todos los módulos del sistema sin necesidad de recurrir inicialmente a un vehículo físico, lo cual supone una reducción significativa de costes, tiempos de preparación y riesgos asociados a pruebas en pista.

Paralelamente, el equipo se encuentra desarrollando un simulador propio, diseñado para adaptarse de manera más precisa a las necesidades específicas del proyecto y a los requisitos particulares de *Formula Student Driverless*. Este desarrollo interno no sólo busca replicar lo ya disponible en simuladores externos, sino también incorporar funcionalidades adicionales, tales como la posibilidad de personalizar con detalle los parámetros de los sensores, ajustar las condiciones ambientales o recrear situaciones límite difíciles de reproducir en entornos reales.

El uso de estos simuladores permite comprobar de manera sencilla que los distintos nodos de la arquitectura se comunican de forma adecuada y que la información fluye correctamente a lo largo de toda la cadena de percepción, planificación y control, como se puede apreciar en la simulación de una de las pruebas de la competición en la Figura 2.3. De este modo, se valida que los algoritmos de fusión de sensores, localización y planificación generen resultados coherentes en un entorno en el que es posible observar y analizar cada etapa con un nivel de detalle elevado.

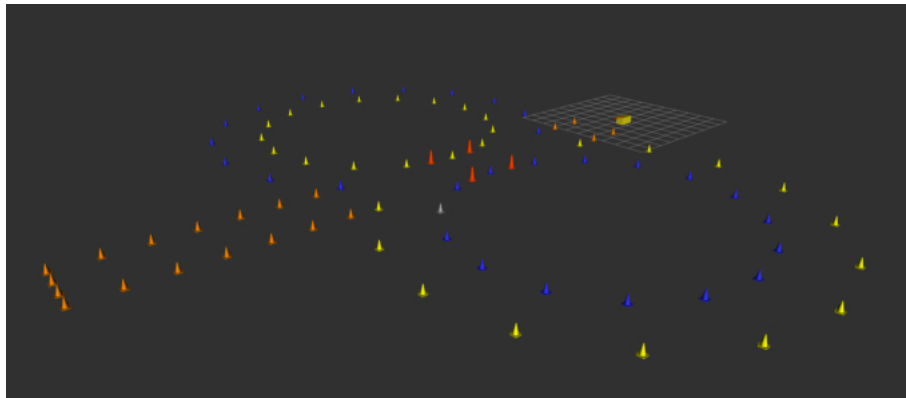


Figura 2.3: Prueba Skidpad (Sección 2.5.2.2) en el simulador de EUFS.

Un aspecto especialmente relevante es que estos entornos virtuales permiten evaluar la robustez del sistema frente a condiciones de fallo parcial o total de uno de los sensores. Por ejemplo, es posible simular la pérdida de señal de una cámara o la introducción de ruido en la nube de puntos LiDAR, verificando si la arquitectura es capaz de mantener un funcionamiento seguro mediante redundancia de información y estrategias de contingencia. Este tipo de pruebas resulta de gran utilidad para anticipar posibles comportamientos no deseados en situaciones reales, incrementando la fiabilidad global del sistema.

Asimismo, los simuladores ofrecen la capacidad de incorporar diferentes circuitos, diseñados de forma similar a los utilizados en las competiciones, lo que brinda la oportunidad de evaluar el rendimiento del vehículo frente a múltiples configuraciones y escenarios. Gracias a ello, se pueden analizar situaciones como cambios bruscos de trayectoria, maniobras de evasión, o la detección de conos en condiciones de visibilidad reducida. Estas pruebas virtuales resultan especialmente útiles para anticipar el comportamiento que tendría el monoplaza en condiciones reales de conducción autónoma, permitiendo depurar errores y optimizar el *software* antes de su implementación en un vehículo físico.

Además de la validación en entornos virtuales, resulta fundamental llevar a cabo pruebas en condiciones reales que permitan evaluar el comportamiento de los distintos sistemas en un escenario físico. No obstante, en la actualidad el equipo aún no dispone de los medios necesarios para

integrar actuadores en el propio monoplaza de *Formula Student*, lo cual imposibilita realizar ensayos completos directamente sobre el vehículo de competición. Con el objetivo de solventar esta limitación, se cuenta con la colaboración del Instituto Universitario de Investigación del Automóvil (INSIA), centro adscrito a la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid.

El INSIA pone a disposición del equipo un vehículo Mitsubishi i-MiEV específicamente modificado para tareas de investigación en el ámbito de la conducción autónoma. Dicho vehículo incorpora los actuadores necesarios para gobernar de manera automática elementos críticos como la dirección, el acelerador y el sistema de frenado. Sobre esta plataforma se instalan tanto los sensores del equipo (cámaras, LiDAR e IMU principalmente) como el ordenador de a bordo que ejecuta la arquitectura *software* previamente descrita.

Como se aprecia en el ejemplo de la Figura 2.4, las pruebas se llevan a cabo en la pista de ensayos del propio INSIA, donde es posible configurar circuitos con conos de características similares a los empleados en las competiciones de *Formula Student Driverless*. Este entorno controlado proporciona un banco de pruebas idóneo para verificar la integración de los distintos subsistemas y analizar cómo el vehículo percibe el entorno, genera un mapa del mismo y planifica trayectorias. Del mismo modo, estas pruebas permiten comprobar que el módulo de control es capaz de transmitir órdenes de manera fiable a los actuadores, logrando así reproducir en un vehículo real los comportamientos esperados en simulación.



Figura 2.4: Prueba en entorno real con el Mitsubishi i-MiEV modificado del INSIA.

De esta forma, la utilización del Mitsubishi i-MiEV no sólo posibilita validar experimentalmente la robustez de la arquitectura de percepción, planificación y control, sino que también constituye una etapa intermedia imprescindible antes de trasladar todo el sistema al monoplaza de *Formula Student*. Gracias a este proceso de verificación, se reduce el riesgo de fallos en competición y se garantiza un desarrollo más seguro y progresivo de la tecnología autónoma implementada por el equipo.

2.4. ROS 2

Como ya se ha mencionado, en *Driverless* se utiliza ROS 2 (*Robot Operating System 2*) como marco de desarrollo principal para toda la arquitectura del vehículo autónomo. ROS 2 no es un sistema operativo en sentido estricto, sino un conjunto de librerías y herramientas que permiten desarrollar sistemas robóticos complejos de manera modular, distribuida y escalable.

2.4.1. Modelo de comunicación

La comunicación entre módulos en ROS 2 se organiza en torno a nodos independientes que intercambian información mediante distintos mecanismos:

- **Topics:** canales basados en el paradigma *publish/subscribe*, empleados en todo el *stack*. Por ejemplo, un nodo de percepción publica la posición de los conos que detecta, mientras que un nodo de localización se suscribe a esa información para utilizarla en su mapa.
- **Servicios:** interacciones síncronas de tipo *request/response*, útiles para operaciones concretas como cambiar configuraciones o solicitar el estado de un subsistema.
- **Acciones:** pensadas para tareas de mayor duración que requieren seguimiento.

El uso de parámetros dinámicos permite ajustar el comportamiento de los nodos durante la ejecución, sin necesidad de recompilar, lo que acelera la puesta a punto del sistema en pruebas reales.

2.4.2. Calidad de servicio y middleware

ROS 2 se apoya en DDS (*Data Distribution Service*) como *middleware* de comunicación, lo que le dota de capacidades avanzadas como:

- Configuración de políticas de **Calidad de Servicio (QoS)**, fundamentales para equilibrar fiabilidad y latencia en función del tipo de dato (por ejemplo, mensajes críticos de control frente a flujos de sensores de alta frecuencia).
- **Ejecución distribuida:** los nodos pueden correr en diferentes máquinas dentro de la red del vehículo, favoreciendo la distribución de carga computacional.
- Compatibilidad con sistemas en **tiempo real**, un requisito esencial en un vehículo autónomo.

2.4.3. Herramientas de desarrollo y validación

ROS 2 incluye un conjunto de utilidades que son esenciales en el flujo de trabajo del equipo:

- **RViz:** herramienta de visualización 3D que permite representar datos de sensores, trayectorias planificadas o estados del vehículo en tiempo real.
- **rqt:** conjunto de interfaces gráficas para monitorizar tópicos, depurar comunicaciones o modificar parámetros durante la ejecución.
- **rosbag:** sistema de registro y reproducción de mensajes, que permite capturar sesiones completas y reutilizarlas en el laboratorio sin necesidad de repetir las pruebas en pista.

- **Gazebo / Ignition:** simulador físico para validar algoritmos en entornos virtuales antes de desplegarlos en el monoplaza real.

2.4.4. Ventajas en el proyecto Driverless

En el contexto de *Fórmula Student Driverless*, ROS 2 ofrece múltiples beneficios:

- **Modularidad:** cada división del *stack* (percepción, planificación, control, localización) puede trabajar de forma independiente sobre sus nodos, integrándose después en una arquitectura común.
- **Escalabilidad:** facilita la incorporación de nuevos sensores o algoritmos sin necesidad de modificar el resto del sistema.
- **Estandarización:** su uso generalizado en la comunidad académica e industrial acelera el desarrollo y la transferencia de conocimiento.
- **Depuración y validación:** las herramientas gráficas y de registro permiten comprobar el funcionamiento del sistema en tiempo real y reproducir condiciones de prueba de forma controlada.

De este modo, ROS 2 constituye la infraestructura de *software* que articula todos los módulos del vehículo, garantizando la comunicación eficiente entre ellos y permitiendo el desarrollo colaborativo de un sistema tan complejo como es el de un vehículo autónomo de competición.

2.5. Competición Formula Student UK

Como ya se ha mencionado, una de las principales limitaciones actuales en el desarrollo del sistema autónomo reside en la ausencia de actuadores en el monoplaza del equipo. Esto también afecta a las competiciones, ya que no existe forma de competir en esta categoría sin un vehículo debidamente preparado.

No obstante, la competición *Formula Student UK* (FSUK), celebrada anualmente en el circuito de Silverstone, proporciona una solución a esta limitación. En la categoría *Driverless*, los organizadores ponen a disposición de los equipos varios vehículos de pruebas (ADS-DV) (Figura 2.5) que ya incorporan actuadores, sobre los cuales cada equipo únicamente debe instalar su propio sistema autónomo, compuesto por la unidad de cómputo, los sensores y demás elementos necesarios. Esta modalidad permite que todos los equipos, independientemente de sus medios, puedan participar en igualdad de condiciones. No obstante, su principal desventaja es que estos vehículos únicamente están disponibles durante los días del evento, lo que limita las oportunidades de integración y validación previa en condiciones reales.



Figura 2.5: ADS-DV con los sensores de *UPM Racing* instalados (FSUK 2024)

2.5.1. Pruebas Estáticas

En la categoría *AI/Driverless* de *Formula Student*, las pruebas estáticas tienen como objetivo evaluar el proceso de ingeniería, la solidez metodológica y la viabilidad técnica y económica del proyecto, más allá del rendimiento en pista. En conjunto, estas pruebas suman **450 puntos** y se estructuran en cuatro eventos: *Business Plan Presentation* (100), *Real World AI* (100), *Engineering Design* (150) y *Simulation Development* (100). La Tabla 2.1 resume la distribución de la puntuación.

Tabla 2.1: Distribución de puntuaciones en las pruebas estáticas (FS-AI).

Evento	Puntos máximos
Business Plan Presentation	100
Real World AI	100
Engineering Design	150
Simulation Development	100
Total	450

2.5.1.1. Business Plan Presentation (100 puntos)

Este evento evalúa la capacidad del equipo para transformar su *stack* autónomo (*hardware* y *software*) en una propuesta de valor rentable y sostenible. El objetivo es presentar un modelo de negocio coherente que demuestre comprensión del mercado, encaje del producto (o sistema) con las necesidades del cliente y una estrategia creíble de implantación a nivel *SAE Level 4*. En la figura 2.6 se resumen los 6 niveles de conducción autónoma definidos por la SAE:

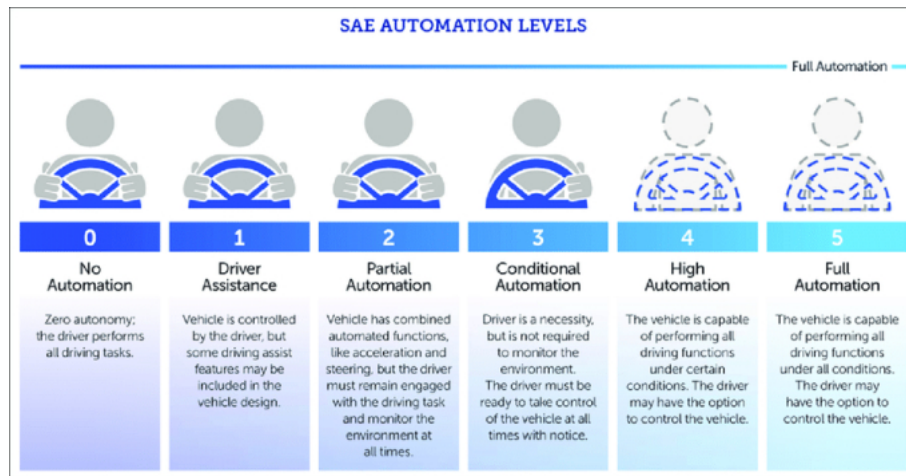


Figura 2.6: Niveles de autonomía de un vehículo definidos por la SAE (SAE International 2016)

Ámbitos de evaluación típicos:

- **Propuesta de valor y caso de uso:** problema que se resuelve, segmentos de clientes, diferenciación tecnológica (p. ej., robustez en condiciones reales y seguridad funcional).
- **Análisis de mercado:** tamaño y acceso al mercado objetivo, competencia directa/indirecta, barreras de entrada, posicionamiento y estrategia *go-to-market*.
- **Modelo económico:** fuentes de ingresos, estructura de costes, CAPEX/OPEX, proyecciones financieras y retorno esperado.
- **Estrategia regulatoria y de riesgos:** cumplimiento normativo, certificaciones, ciberseguridad, privacidad de datos, responsabilidad legal.
- **Plan de desarrollo y escalado:** hitos técnicos, *roadmap* de producto, alianzas industriales, estrategia de fabricación y soporte.
- **Calidad de la presentación:** claridad narrativa, evidencias cuantitativas, visuales profesionales y defensa en Q&A.

2.5.1.2. Real World AI (100 puntos)

Su finalidad es comprobar el entendimiento del equipo sobre los retos que afrontan los vehículos autónomos en entornos reales. Se valora la capacidad analítica y crítica para conectar la tecnología con el contexto operativo, regulatorio y social. La puntuación se reparte entre (i) el análisis de *temas clave* y (ii) un *estudio de caso* definido por la organización.

Temas clave (selección de al menos tres):

- **Interacción con la infraestructura y otros usuarios:** señales, marcas viales, peatones, vehículos, obstáculos y escenarios ambiguos.
- **Evaluación del *sensor suite*:** alternativas y compensaciones (LiDAR, cámaras, radar, GNSS/IMU), redundancia y degradación controlada.
- **Conectividad (V2V/V2I):** ventajas, límites operativos, ciberseguridad y resiliencia.

- **Seguridad funcional:** consideraciones ISO 26262, SOTIF (ISO/PAS 21448), gestión de riesgos y *fail-operational*.
- **Climatología adversa:** lluvia, niebla, nieve, baja iluminación; estrategias de *sensor fusion* y robustez.
- **Impacto socioeconómico y ético:** aceptación pública, empleo, responsabilidad y marco legal.

Estudio de caso (2025): análisis crítico de la preparación global para *SAE Level 4* en vías públicas, comparando al menos dos países (madurez tecnológica, legislación y preparación social), identificando brechas, riesgos y *trade-offs* realistas.

2.5.1.3. Autonomous Design (150 puntos)

Es la prueba estática con mayor peso. Evalúa el **proceso de ingeniería** seguido para concebir, diseñar, integrar y justificar el *Autonomous Driving System* (ADS) sobre el vehículo. No se valora únicamente el resultado, sino la **trazabilidad de decisiones** y la **madurez del sistema**.

Ámbitos de evaluación típicos:

- **Arquitectura del sistema:** particionado funcional, interfaces entre nodos (ROS 2), temporización, sincronización y gestión de *latency/jitter*.
- **Selección e integración de sensores:** racional técnico (*Field Of View* (FOV), alcance, resolución, tasa de datos), calibraciones (intrínsecas/extrínsecas) y estrategias de redundancia.
- **Percepción y localización:** *pipeline* de visión/LiDAR, filtrado y segmentación, fusión multi-sensor, SLAM/localización global, métricas de *tracking*.
- **Planificación y control:** generación de trayectorias, criterios de optimalidad (seguridad, suavidad, tiempo), controladores (p. ej., Stanley/MPC) y validación HIL/SIL.
- **Seguridad y fallback:** detección de fallos, modos degradados, *fail-safe/fail-operational*, gestión de riesgos y normativas aplicables.
- **Verificación y validación:** cobertura de escenarios, correlación sim-pista, *logging, replay, Key Performance Indicators (KPIs)* (precisión, latencia, robustez).
- **Calidad de ingeniería:** gestión de configuración, *Continuous Integration and Continuous Delivery*, CI/CD, documentación técnica, justificación de *trade-offs* y análisis de sensibilidad.

2.5.1.4. Simulation Development (100 puntos)

Valora la **comprensión y uso de la simulación** en el flujo de desarrollo del ADS. Se espera que el equipo replique en simulación, con rigor, los eventos dinámicos (*Acceleration, Skidpad, Autocross, Trackdrive*) utilizando modelos de vehículo, sensores y entorno apropiados. Se admiten herramientas *open-source* o de nivel industrial; se premia la fidelidad de los modelos y la solidez metodológica.

Ámbitos de evaluación típicos:

- **Flujo de simulación extremo a extremo:** escenarios, *mission control*, orquestación, *replay* y trazabilidad de resultados.

- **Modelado:** dinámicas del vehículo, fricción y neumáticos, sensores (modelos físicos/estocásticos), factores ambientales (clima, iluminación).
- **Integración con el *stack*:** percepción, localización, planificación y control operando en lazo; integración con interfaz de actuadores simulados.
- **Validación y correlación:** metodologías de correlación sim-real, factores de ruido, análisis de sensibilidad y cobertura de *corner cases*.
- **Robustez:** inyección de fallos (pérdida de GNSS, saturación de cámara, ruido en LiDAR), degradación controlada y comportamiento seguro.
- **Herramientas y visualización:** depuración, métricas cuantitativas, cuadros de mando y claridad en la comunicación de resultados.

En conjunto, las pruebas estáticas garantizan que el proyecto no sólo funcione, sino que sea **explicable, justificable y transferible**: con una arquitectura sólida, decisiones técnicas razonadas, evidencia de validación y una visión realista de implantación y sostenibilidad.

2.5.2. Pruebas Dinámicas

Las pruebas dinámicas ponen a prueba el conjunto: *hardware*, sensores, *software* y la integración entre todos los nodos. En FS-AI se evalúan cuatro misiones principales en pista: *Skidpad*, *Acceleration*, *Autocross/Sprint* y *Trackdrive*. A continuación se explica de forma directa y práctica qué supone cada una, y qué se exige antes de poder salir a pista.

Las pruebas dinámicas suman un total de **550** puntos, repartidos de la siguiente forma:

Tabla 2.2: Distribución de puntuaciones en las pruebas dinámicas (FS-AI).

Evento	Puntos máximos
Skidpad	100
Acceleration	100
Autocross	100
Trackdrive	250
Total	550

2.5.2.1. Inspecciones previas (Static Inspection A / Static Inspection B / Autonomous Demonstration)

Antes de acceder a las pruebas dinámicas, la normativa exige la superación de una secuencia de comprobaciones por parte del equipo de *scrutineering*: **Static Inspection A**, **Static Inspection B** y **Autonomous Demonstration Mission**.

- **Static Inspection A**
 - El equipo configurará el ADS-DV tal y como pretende utilizarlo en los eventos dinámicos, incluyendo el montaje del *hardware* de cómputo o ensamblajes de sensores previamente aprobados.

- Se desplegará el *software* en el vehículo.
- El vehículo será elevado del suelo; los oficiales podrán decidir si se retiran o no las ruedas.
- Se seleccionará la misión '*Static Inspection A*' y se ejecutará con la señal RES "Go".
- El vehículo debe barrer todo el rango de ángulos de dirección y finalizar en la posición centrada.
- El tren motriz debe acelerar progresivamente hasta alcanzar 200 rpm en 10 segundos y detenerse en menos de 5 segundos, tras lo cual el estado de la *Vehicle Control Unit* (VCU) debe fijarse en '*AS Finished*'.
- Finalmente, el sistema autónomo será reiniciado mediante un ciclo de apagado/encendido (*power-cycle*).

Esta primera inspección se centra en comprobar la correcta integración básica del sistema: que la dirección responda adecuadamente a consignas de *software*, que el tren motriz sea capaz de ejecutar rampas de velocidad bajo control y que la VCU gestione de manera fiable las transiciones de estado. Al hacerse con el vehículo levantado, se eliminan riesgos en pista y se verifica que los actuadores obedecen al *software* sin cargas externas.

■ Static Inspection B

- Tras la finalización de la inspección A, se seleccionará la misión '*Static Inspection B*' y se ejecutará con la señal RES "Go".
- El tren motriz debe acelerar hasta 50 rpm y, a continuación, activarse el sistema de freno de emergencia (EBS), lo que debe provocar que el estado de la VCU pase a '*AS Emergency*'.

La finalidad de esta prueba es verificar específicamente que el sistema de freno de emergencia (EBS) funciona correctamente y que, en caso de activarse, el vehículo entra en el estado seguro *AS Emergency*. Esto asegura que, ante un fallo o una situación crítica, los mecanismos de seguridad están operativos y el *software* responde como exige la normativa.

■ Autonomous Demonstration Mission

- Una vez validadas las otras dos inspecciones, el equipo será autorizado a pasar la última de las pruebas.
- El equipo cargará su *software* en el ordenador AI del vehículo y, si ha solicitado utilizar sensores adicionales, instalará el ensamblaje aprobado.
- El vehículo será colocado entre una serie de conos similares a los de la prueba de aceleración.
- Se seleccionará la misión '*Autonomous Demo*' y se ejecutará con la señal RES "Go".
- El vehículo debe barrer la dirección hacia la izquierda y hacia la derecha, regresando después a la posición recta.
- A continuación, debe acelerar durante 10 m hasta alcanzar al menos 15 km/h.
- Debe detenerse en un máximo de 10 m tras la aceleración.
- Posteriormente, debe volver a acelerar otros 10 m alcanzando de nuevo un mínimo de 15 km/h.
- El sistema debe activar el EBS.
- Durante toda la misión el vehículo debe permanecer dentro del espacio delimitado por los conos.

El propósito de la demostración autónoma es comprobar, ya con el vehículo en el suelo, que el sistema es capaz de ejecutar maniobras dinámicas básicas: controlar dirección, acelerar y detenerse de forma segura. Es una validación crítica porque, al haber rozamiento real entre las ruedas y el asfalto, se introduce una carga física que muchas veces no aparece en banco, lo que puede poner en evidencia problemas de par, control de velocidad o calibración de sensores. Superar esta misión demuestra que el ADS tiene la robustez mínima necesaria para acceder a pruebas en pista.

En caso de no superar cualquiera de estas fases, el vehículo no estará autorizado a participar en las pruebas dinámicas hasta que el equipo subsane las deficiencias y vuelva a ejecutar con éxito la inspección correspondiente.

Una vez superadas las inspecciones previas, los equipos pueden acceder a las pruebas dinámicas. Estas representan la parte más exigente de la competición, ya que evalúan directamente el rendimiento del sistema autónomo en pista. Cada prueba está diseñada para medir aspectos concretos del comportamiento del vehículo, desde la maniobrabilidad hasta la fiabilidad en condiciones prolongadas.

2.5.2.2. Skidpad (100 puntos)

- **Formato:** Como se muestra en la Figura 2.7, el trazado consiste en dos círculos de 15,25 m de diámetro, dispuestos en forma de “8”. El vehículo debe recorrer primero dos vueltas cronometradas en un sentido y después otras dos en el sentido opuesto. El tiempo oficial se calcula como la media de la primera vuelta cronometrada en cada dirección.

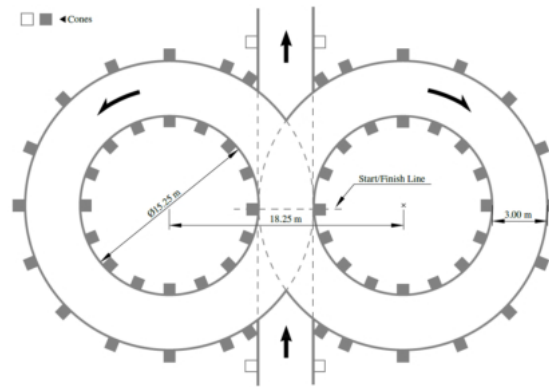


Figura 2.7: Disposición de la prueba Skidpad (*FSUK Rules 2025*).

- **Disposición:** Los conos marcan las trayectorias interiores y exteriores de cada círculo, con una separación precisa que evita atajos.
- **Finalización:** Al terminar la prueba, el vehículo debe salir por el carril de salida en la misma dirección de entrada y detenerse por completo dentro de los 25 m posteriores a la línea de cronometraje, permaneciendo dentro del carril marcado y entrando en el estado ‘*AS Finished*’.
- **Puntuación:** Todos los equipos que completen la prueba reciben 25 puntos iniciales. El resto (hasta un máximo de 100) se calcula comparando el tiempo del equipo (T_{team}) con el mejor tiempo de la jornada ($T_{\text{mín}}$), mediante la fórmula oficial:

$$\text{SCORE} = 25 + 75 \cdot \left(\frac{\left(\frac{T_{\text{máx}}}{T_{\text{team}}} \right)^2 - 1}{\left(\frac{T_{\text{máx}}}{T_{\text{mín}}} \right)^2 - 1} \right)$$

siendo $T_{\text{máx}} = 2 \cdot T_{\text{mín}}$. Estos últimos solo son otorgados si $T_{\text{team}} \leq T_{\text{máx}}$.

La finalidad de esta prueba es medir la capacidad del vehículo para mantener un comportamiento estable en curvas de radio constante, evaluando la adherencia lateral, la precisión de la dirección y la efectividad del control en maniobras prolongadas.

2.5.2.3. Acceleration (100 puntos)

- **Formato:** El vehículo parte desde parado y debe recorrer 75 m en línea recta, delimitados por conos a ambos lados de la pista como se muestra en la Figura 2.8. El cronometraje comienza al recibir la señal de inicio y se detiene en la línea de meta.

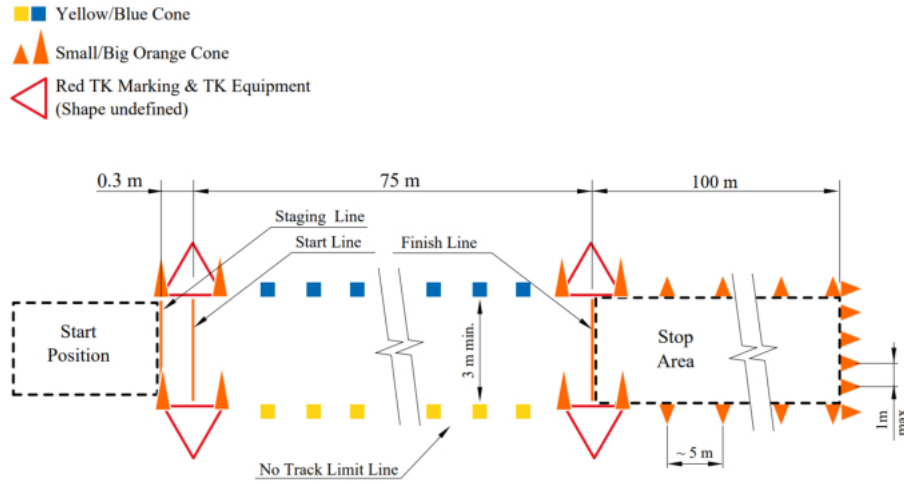


Figura 2.8: Disposición de la prueba *Acceleration* (FSG Handbook 2025).

- **Disposición:** El carril tiene una anchura de 3 m, ampliándose a 5 m en la zona de frenado. En caso de pérdida de trayectoria, los conos determinan la validez del intento.
- **Finalización:** Tras cruzar la meta, el vehículo debe detenerse por completo dentro de los 100 m posteriores, dentro del carril marcado, y pasar al estado '*AS Finished*'.
- **Puntuación:** Los equipos que completen al menos un intento válido de los dos intentos disponibles reciben 25 puntos. Los 75 puntos restantes dependen del tiempo corregido del equipo en comparación con el mejor tiempo del día:

$$\text{SCORE} = 25 + 75 \cdot \left(\frac{\frac{T_{\text{máx}}}{T_{\text{team}}} - 1}{\frac{T_{\text{máx}}}{T_{\text{mín}}} - 1} \right)$$

con $T_{\text{máx}} = 1,5 \cdot T_{\text{mín}}$ y solo son otorgados si $T_{\text{team}} \leq T_{\text{máx}}$.

Esta prueba evalúa la capacidad de aceleración pura, así como la sincronización entre el *software*, el sistema de propulsión y la gestión del par motor. Constituye un indicador directo del rendimiento del tren motriz.

2.5.2.4. Autocross / Sprint (100 puntos)

- **Formato:** El vehículo debe completar una vuelta a un circuito corto de aproximadamente 800–1000 m, diseñado con curvas cerradas, chicanes y tramos de aceleración. El objetivo es recorrerlo en el menor tiempo posible sin cometer penalizaciones.
- **Disposición:** Los conos definen tanto los límites laterales de la pista (3 m de ancho mínimo) como las chicanes y giros de radio variable. Un derribo implica penalización en tiempo.
- **Finalización:** Tras cruzar la meta, el vehículo debe detenerse por completo en un máximo de 30 m, permaneciendo dentro de la pista y entrando en el estado ‘AS Finished’.
- **Puntuación:** La puntuación sigue la misma estructura que en Acceleration: 25 puntos por completar un intento válido y 75 adicionales en función del tiempo corregido respecto al mejor equipo, que solo son otorgados si $T_{\text{team}} \leq T_{\text{máx}}$:

$$\text{SCORE} = 25 + 75 \cdot \left(\frac{\frac{T_{\text{máx}}}{T_{\text{team}}} - 1}{\frac{T_{\text{máx}}}{T_{\text{mín}}} - 1} \right)$$

con $T_{\text{máx}} = 1,45 \cdot T_{\text{mín}}$.

La finalidad del Autocross es evaluar la agilidad, la precisión del control lateral y la eficacia de los algoritmos de planificación de trayectorias. También sirve como predictor de rendimiento para la prueba de resistencia.

2.5.2.5. Trackdrive (250 puntos)

- **Formato:** Es la prueba de mayor duración. El vehículo debe completar 10 vueltas consecutivas en un circuito con el mismo formato que el del Autocross. No existe señal de última vuelta: el sistema debe ser capaz de contar las vueltas de forma autónoma.
- **Disposición:** Similar a la del Autocross, aunque con mayor longitud y variedad de curvas. La anchura mínima es de 3 m, delimitada por conos interiores y exteriores.
- **Finalización:** Tras completar las diez vueltas, el vehículo debe detenerse por completo en un máximo de 30 m tras la meta, permaneciendo en el carril y entrando en el estado AS Finished.
- **Puntuación:** Se otorgan hasta 100 puntos en función del número de vueltas completadas (10 puntos por vuelta). Además, se pueden obtener hasta 150 puntos adicionales en función del tiempo total corregido respecto al mejor de la jornada si $T_{\text{team}} \leq T_{\text{máx}}$:

$$\text{SCORE} = 10 \cdot N_{\text{laps}} + 150 \cdot \left(\frac{\frac{T_{\text{máx}}}{T_{\text{team}}} - 1}{\frac{T_{\text{máx}}}{T_{\text{mín}}} - 1} \right)$$

siendo N_{laps} el número de vueltas completadas (máximo 10).

En la Figura 2.9 se muestra un segmento del circuito de la prueba *Trackdrive*.

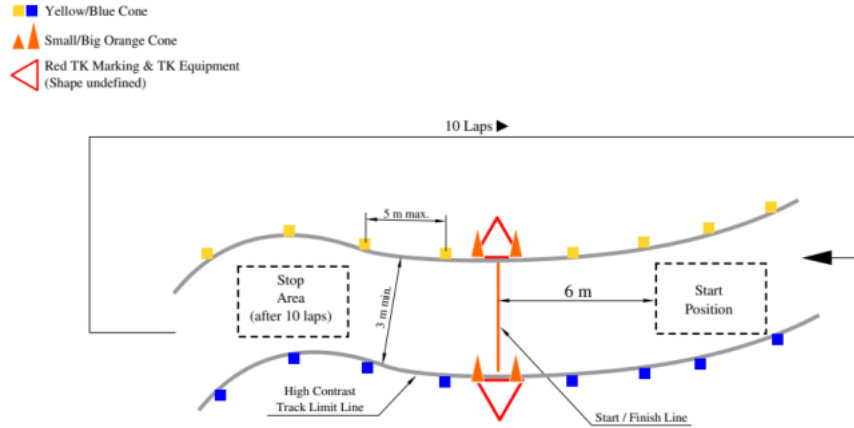


Figura 2.9: Disposición de la prueba *Trackdrive* (*FSG Competition Handbook 2018*)

La finalidad del *Trackdrive* es medir la fiabilidad del sistema autónomo en condiciones prolongadas de carrera, donde deben coordinarse de manera robusta los módulos de percepción, localización, planificación y control.

Las penalizaciones (conos caídos, salidas de pista, paradas inseguras, etc.) se añaden al tiempo cronometrado o pueden provocar *Did Not Finish* (DNF)/ *Disqualified* (DQ). Dado que las fórmulas usan T_{team} (tiempo *incluyendo* penalizaciones), cualquier sanción afecta directamente a la puntuación. Por eso, en competición hay que equilibrar la búsqueda de un buen tiempo con la seguridad y la robustez para evitar sanciones costosas.

Algunos comentarios finales respecto a las pruebas dinámicas:

- Las fórmulas normalizan cada resultado respecto al mejor tiempo del día ($T_{\text{mín}}$) y a un umbral superior ($T_{\text{máx}}$), lo que hace la puntuación comparable entre equipos y robusta frente a las condiciones del circuito.
- El hecho de que haya un bloque de puntos fijos por completar una ejecución válida (los 25 puntos en muchos eventos) incentiva asegurar al menos una ejecución correcta antes que arriesgar un intento que acabe en DNF.
- FSUK proporciona un *Practice Track* y vehículos con actuadores para que los equipos puedan comprobar en pista su *stack* antes de las pruebas oficiales; combinado con la simulación, esto es clave para depurar modos degradados y la integración final.
- Pasar las inspecciones previas y ensayar bien en simulación y *Practice Track* son tareas imprescindibles para maximizar la puntuación y reducir el riesgo de penalizaciones.

Capítulo 3

Estado del arte

3.1. Percepción en vehículos autónomos

La percepción es una de las capacidades más críticas en un sistema de conducción autónoma, ya que permite al vehículo interpretar su entorno para tomar decisiones seguras y eficientes. Este módulo se encarga de responder preguntas fundamentales como: ¿qué hay alrededor del vehículo?, ¿dónde están los obstáculos?, ¿cuál es la zona segura para circular?, y ¿qué elementos deben ser evitados o seguidos?

Para lograr esto, el sistema de percepción emplea sensores diversos que capturan información sobre el entorno, la cual es procesada para obtener representaciones estructuradas del mundo. Estas representaciones son consumidas por los módulos de planificación y control, que determinan las trayectorias y acciones que debe ejecutar el vehículo.

Entre las tareas clave de percepción se encuentran la detección y seguimiento de obstáculos, la segmentación del espacio navegable, la clasificación de objetos (peatones, vehículos, conos, señales, etc.), y la estimación de movimiento relativo. Una percepción robusta es, por tanto, imprescindible para garantizar el funcionamiento seguro y fiable del sistema completo.

La arquitectura de percepción de un vehículo autónomo [16] suele estar compuesta por un conjunto de módulos interconectados que colaboran para construir una representación coherente, precisa y en tiempo real del entorno circundante. El flujo de información se inicia en la capa sensorial, donde distintos tipos de sensores capturan datos complementarios sobre el entorno. Entre los más comunes se encuentran las cámaras, que proporcionan información visual detallada como el color, la textura o el estado de las señales de tráfico; los sensores LiDAR (*Light Detection and Ranging*), que generan mapas tridimensionales de alta resolución midiendo el tiempo de vuelo de pulsos láser; los radares, que son especialmente útiles en condiciones meteorológicas adversas y para medir velocidades relativas; los ultrasónicos, que permiten detectar objetos a corta distancia como bordillos o peatones cercanos; las IMU (*Inertial Measurement Unit*), que detectan aceleraciones y rotaciones del vehículo; y los sistemas de GPS de alta precisión, que aportan información geoespacial para la localización global.

Sin embargo, ninguno de estos sensores es capaz por sí solo de ofrecer una comprensión completa y confiable del entorno, debido a sus limitaciones individuales en aspectos como la resolución, la sensibilidad a condiciones externas o el alcance. Por este motivo, es habitual diseñar arquitecturas que implementen procesos de fusión sensorial, donde los datos de múltiples fuentes se combinan para formar una representación más robusta, consistente y precisa del entorno. Esta fusión puede realizarse en distintos niveles: a nivel de datos brutos, de características extraídas o de decisiones tomadas por distintos módulos. El objetivo final es mitigar los errores individuales de cada sensor y aprovechar sus fortalezas complementarias para mejorar la percepción general del sistema.

3.2. Comparativa entre sensores exteroceptivos

En los sistemas de percepción de vehículos autónomos, los sensores se pueden clasificar en función del tipo de información que proporcionan [17]. Los sensores propioceptivos, como las IMU o

los *encoders*, recogen datos internos del vehículo, como su aceleración, orientación o desplazamiento relativo. Por otro lado, los sensores de posicionamiento absoluto, como el GNSS (*Global Navigation Satellite System*), permiten estimar la posición del vehículo en un sistema de coordenadas global. Sin embargo, para que el vehículo pueda entender y reaccionar ante su entorno, es necesario un tercer tipo de sensores: los sensores exteroceptivos, que capturan información sobre el entorno externo, ya sea a nivel geométrico, visual o dinámico.

Dentro de esta última categoría se encuentran sensores como las cámaras, los LiDAR y los radares, fundamentales para tareas como la detección de obstáculos, la clasificación de objetos, la identificación del entorno navegable o la predicción del comportamiento de otros agentes. Cada uno de estos sensores ofrece ventajas particulares y presenta limitaciones, por lo que una arquitectura multisensor, que combine sus puntos fuertes, es habitual en los sistemas de percepción modernos.

3.2.1. Cámaras

Las cámaras [18] son uno de los sensores más utilizados en sistemas de percepción para vehículos autónomos, debido a su capacidad para capturar imágenes del entorno con alto nivel de detalle semántico. Proporcionan información visual, que es indispensable para tareas como el reconocimiento de señales de tráfico, lectura de semáforos, detección de peatones o interpretación de marcas viales.

El funcionamiento físico de una cámara digital se basa en la captación de la luz reflejada por los objetos del entorno, que es enfocada mediante un sistema óptico (lente) hacia una matriz fotosensible conocida como sensor [19]. Este sensor está compuesto por millones de celdas fotosensibles llamadas fotodiodos, que convierten la energía lumínica incidente en carga eléctrica proporcional a la intensidad de luz recibida. Los sensores más comunes son los de tipo CMOS (*Complementary Metal-Oxide Semiconductor*), que destacan por su bajo consumo y rápida lectura. Cada fotodiodo mide la intensidad luminosa en una única componente del espectro visible, generalmente filtrada mediante una matriz de Bayer (compuesta por filtros RGB), permitiendo reconstruir el color final de cada píxel mediante interpolación. El resultado es una imagen digital codificada en una matriz de valores discretos que representa una proyección bidimensional de la escena tridimensional.

Entre sus principales ventajas se encuentra su alta densidad de información visual, lo que permite detectar texturas, formas y colores con gran riqueza semántica. Son sensores compactos, ligeros y de bajo coste, lo que facilita su integración en vehículos y su despliegue en configuraciones múltiples. Además, ofrecen una alta resolución espacial y una tasa de adquisición elevada, lo que permite trabajar con flujos de vídeo en tiempo real. Su bajo consumo energético y su madurez tecnológica las convierten en una opción robusta y ampliamente adoptada tanto en la industria como en el entorno académico.

No obstante, presentan ciertas limitaciones importantes. Las cámaras dependen fuertemente de las condiciones de iluminación, lo que puede deteriorar su rendimiento en situaciones de contraluz, sombras fuertes, ambientes nocturnos o fenómenos meteorológicos adversos como niebla, lluvia o nieve. Además, al tratarse de sensores pasivos que capturan una proyección 2D de la escena, no proporcionan directamente información de profundidad, lo que supone un obstáculo en tareas de localización o detección precisa de obstáculos. El cálculo de la geometría tridimensional a partir de una sola imagen requiere modelos de reconstrucción complejos y con alta incertidumbre, y el procesamiento de imágenes con redes neuronales profundas implica una carga computacional elevada.

Esta última desventaja, la ausencia de información de profundidad, puede mitigarse utilizando una configuración estereoscópica [20]. En este enfoque se disponen dos cámaras paralelas separadas una distancia conocida (llamada “*baseline*”) y se capturan dos imágenes simultáneamente desde ángulos ligeramente distintos, tal y como se muestra en la Figura 3.1.

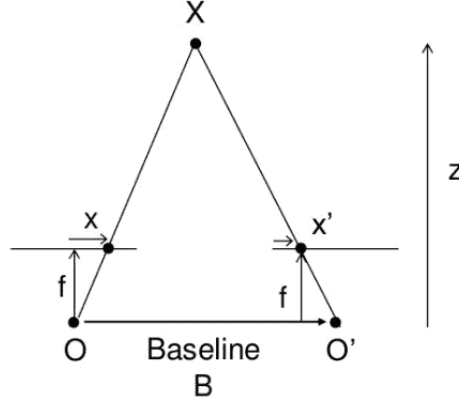


Figura 3.1: Ejemplo de configuración de cámaras estereoscópicas (OpenCV).

Mediante técnicas de correspondencia epipolar, se calcula la disparidad (la diferencia de posición entre un mismo punto observado en ambas imágenes) y, conociendo la distancia entre las cámaras B y la distancia focal f , se puede estimar la profundidad Z de un punto utilizando la fórmula trigonométrica fundamental de la visión estereo:

$$Z = \frac{f \cdot B}{d}, \quad (3.1)$$

donde d es la disparidad entre los dos píxeles correspondientes al mismo punto del mundo real. Este método permite recuperar la tercera dimensión y construir mapas de profundidad, aunque su precisión depende de la calibración de las cámaras, la resolución de imagen y la calidad de la correspondencia entre píxeles.

3.2.2. LiDAR

El sensor LiDAR (*Light Detection and Ranging*) [21] es un dispositivo activo que permite medir distancias precisas al entorno mediante pulsos láser. Se ha convertido en un componente fundamental en muchos sistemas de percepción para vehículos autónomos, gracias a su capacidad para generar representaciones tridimensionales del entorno con gran resolución espacial y precisión métrica. Su principal aportación frente a sensores ópticos pasivos como las cámaras es la obtención directa de geometría 3D, lo cual es clave para tareas de localización, detección de obstáculos y planificación de rutas.

El funcionamiento físico del LiDAR se basa en la medición del tiempo de vuelo (*Time of Flight* (ToF)) de un haz láser. El sensor emite un pulso láser en una dirección determinada y mide el tiempo que tarda en reflejarse en un objeto del entorno y volver al receptor, tal como se muestra en la Figura 3.2

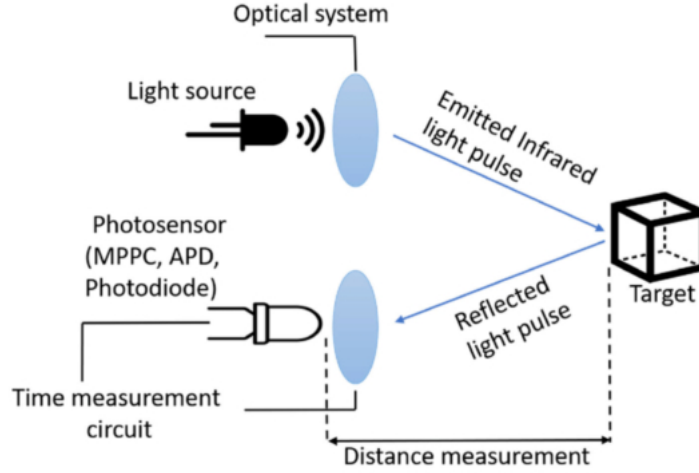


Figura 3.2: Esquema de funcionamiento de un sensor LiDAR basado en tiempo de vuelo (ToF) [22].

Dado que la velocidad de la luz c es conocida, la distancia d al objeto puede calcularse mediante la expresión:

$$d = \frac{c \cdot \Delta t}{2}, \quad (3.2)$$

donde Δt es el tiempo transcurrido entre la emisión y la recepción del pulso. El factor 2 se debe a que el haz recorre la distancia dos veces (ida y vuelta). En los sensores LiDAR modernos, como el Ouster OS1-32 utilizado en este proyecto, se emiten múltiples haces simultáneamente mediante una matriz rotatoria de emisores láser que giran en torno a un eje vertical. Cada emisor (o canal) está orientado a un ángulo fijo en el plano vertical, lo que permite construir una imagen tridimensional del entorno a partir de la acumulación de múltiples mediciones por segundo. El resultado es una nube de puntos (*point cloud*), donde cada punto está definido por sus coordenadas espaciales (x, y, z) y puede ir acompañado de información adicional como intensidad del retorno, canal de origen, *timestamp*, reflectividad o luz ambiente.

El LiDAR presenta varias ventajas fundamentales para su uso en vehículos autónomos. Proporciona mediciones de distancia muy precisas, independientemente de la iluminación ambiental, lo que lo hace robusto frente a condiciones nocturnas o de luz cambiante. La representación 3D del entorno facilita la detección de obstáculos, el modelado del terreno y la construcción de mapas con alta fidelidad. Su cobertura angular puede alcanzar los 360 grados horizontales, y al estar compuesto por múltiples canales verticales (32 en el caso del OS1), permite escanear tanto el suelo como estructuras elevadas. Además, es relativamente poco afectado por la textura o el color del objeto, ya que mide la distancia por reflexión, no por interpretación visual.

Sin embargo, también existen limitaciones. El rendimiento del LiDAR puede degradarse en presencia de condiciones meteorológicas adversas como niebla, lluvia o nieve, ya que las partículas en suspensión pueden reflejar los haces láser y generar ruido. Su capacidad para detectar materiales oscuros o altamente absorbentes es limitada, y las superficies especulares o transparentes pueden producir errores de medición. En comparación con las cámaras, los LiDAR no capturan información semántica como color o textura, lo que limita su utilidad para tareas como la clasificación de objetos sin apoyo de otros sensores. Además, son dispositivos más caros y voluminosos, con mayores requerimientos energéticos y de procesamiento.

3.2.3. Radar

El radar (*Radio Detection and Ranging*) es un sensor activo que utiliza ondas electromagnéticas en el espectro radioeléctrico para detectar y localizar objetos [23]. Se ha empleado tradicionalmente en aplicaciones militares y aeronáuticas, pero en los últimos años ha adquirido un papel destacado en sistemas avanzados de asistencia a la conducción (ADAS) y vehículos autónomos debido a su capacidad para detectar objetos a larga distancia con gran robustez frente a condiciones adversas. A diferencia de los sensores ópticos o LiDAR, el radar no depende de la luz visible ni de la reflectancia superficial, lo que lo convierte en un sensor muy fiable en contextos de baja visibilidad.

Desde el punto de vista físico, un radar automotriz funciona emitiendo una señal de radiofrecuencia mediante una antena direccional y analizando la señal reflejada por los objetos del entorno. Para estimar la distancia a un objeto, se mide el retardo temporal entre la emisión y la recepción, de manera similar al LiDAR. Sin embargo, lo más habitual en automoción es el uso de radar FMCW (*Frequency-Modulated Continuous Wave*), que no emite pulsos discretos, sino una señal continua cuya frecuencia varía linealmente con el tiempo. Al recibir el eco reflejado, se compara la frecuencia de la señal original con la de retorno, y esa diferencia (Δf) es proporcional al tiempo de vuelo, y por tanto a la distancia al objeto. Esta relación se expresa mediante la ecuación:

$$R = \frac{c_0 |\Delta f|}{2 \left(\frac{d(f)}{d(t)} \right)}, \quad (3.3)$$

donde R es la distancia al objeto en metros, $d(f)/d(t)$ es la desviación de frecuencia por unidad de tiempo y c_0 es la velocidad de la luz $= 3 \cdot 10^8 \frac{m}{s}$. Esta fórmula permite calcular con precisión la distancia al objetivo en función del retardo entre las señales.

Además, gracias al efecto Doppler, el radar puede estimar la velocidad relativa v de los objetos en movimiento, analizando el desplazamiento en frecuencia producido por su velocidad radial. Este desplazamiento genera una frecuencia Doppler f_d , y la velocidad se puede calcular mediante la expresión:

$$v = \frac{f_d \cdot \lambda}{2}, \quad (3.4)$$

donde λ es la longitud de onda de la señal emitida por el radar. De este modo, el radar no solo puede detectar la presencia de objetos, sino también medir su velocidad respecto al vehículo, lo que es clave para el seguimiento y la prevención de colisiones.

El radar automotriz de largo alcance típicamente opera en la banda de 77 GHz, lo que le permite mantener una buena resolución espacial sin comprometer el alcance. La señal emitida se puede modular en forma de múltiples haces simultáneos (mediante antenas *phased array* o MIMO), lo que permite obtener una matriz de detección en dos dimensiones: distancia y ángulo. Aunque la resolución angular del radar es inferior a la de los sensores ópticos, las nuevas generaciones de radares de alta definición están reduciendo esta brecha mediante mejoras en la arquitectura de las antenas y el procesamiento digital.

Entre las principales ventajas del radar se encuentra su robustez frente a condiciones ambientales adversas: su señal penetra con facilidad a través de niebla, lluvia, nieve y polvo, condiciones que deterioran notablemente el rendimiento de cámaras y LiDAR. También es muy eficaz para medir velocidades relativas, lo que lo convierte en el sensor ideal para tareas como el control de cruce adaptativo (ACC), detección de vehículos en movimiento y prevención de colisiones. Su consumo energético es moderado, su coste es inferior al del LiDAR, y su rango de detección puede superar fácilmente los 150 metros.

Sin embargo, el radar presenta también limitaciones importantes. Su resolución espacial es baja en comparación con cámaras o LiDAR, lo que dificulta la reconstrucción precisa de la forma

y contorno de los objetos detectados. Además, no proporciona información visual o semántica, como color o textura, y sufre en entornos con múltiples superficies metálicas cercanas, donde pueden producirse interferencias o reflexiones múltiples. En situaciones urbanas complejas, el radar tiende a producir retornos confusos o “fantasmas”, especialmente cuando las señales se reflejan en estructuras cercanas antes de llegar al objeto real. Por estas razones, el radar raramente se utiliza de forma aislada, y suele formar parte de sistemas de percepción multisensor que combinan sus fortalezas con las de otros dispositivos.

3.2.4. Elección del sensor LiDAR

En el contexto del equipo *UPM Racing*, la elección del sensor LiDAR responde tanto a criterios técnicos como a limitaciones prácticas. Aunque una arquitectura multisensor que incluya radar, cámaras y LiDAR es común en vehículos autónomos, los equipos de *Formula Student* no suelen incorporar radar en sus sistemas. Esto se debe a varias razones. Por un lado, las competiciones se desarrollan en circuitos cerrados y controlados, sin tráfico real, lo que reduce la necesidad de detectar objetos a larga distancia o en condiciones meteorológicas adversas, donde el radar ofrece ventajas. Además, la capacidad para detectar objetos en movimiento es menos relevante en un entorno donde los obstáculos suelen ser estáticos (como conos o barreras). Por otro lado, el coste y la complejidad de integración de estos sensores no se justifica en un entorno tan restringido, lo que lleva a la mayoría de equipos a optar únicamente por una arquitectura basada en cámaras y sensores LiDAR.

Por otra parte, las cámaras estereoscópicas Stereolabs ZED que utilizaba el equipo en temporadas anteriores presentaron fallos que dejaron al sistema sin capacidad de visión funcional. Aunque el vehículo proporcionado por la organización de FSUK incorpora una nueva y mejor versión, las ZED 2i de Stereolabs, esta limitación impidió alcanzar un nivel de madurez suficiente en el desarrollo de algoritmos de visión o fusión sensorial, lo que llevó a centrar los esfuerzos exclusivamente en el uso del sensor LiDAR.

El sensor LiDAR empleado es un Ouster OS1 de 32 canales, que proporciona datos 3D en tiempo real con una resolución angular adecuada y un rango de detección suficiente para el entorno de competición. Este sensor genera nubes de puntos densas que permiten detectar obstáculos, clasificar objetos como conos y estimar distancias con una precisión elevada, todo ello sin depender de la luz ambiente ni de colores visibles, lo que lo convierte en una solución robusta frente a distintas condiciones de iluminación.

3.3. Procesamiento de nubes de puntos

El primer paso en un sistema de percepción basado en LiDAR consiste en transformar la nube de puntos “en bruto” (*raw*) en una representación más limpia y manejable. Una nube de puntos es, en esencia, un conjunto masivo de coordenadas espaciales (x , y , z) que describen la forma y posición de los objetos presentes en una escena. El objetivo de esta etapa es reducir el volumen de datos, eliminar ruido y descartar estructuras irrelevantes como el suelo, de forma que se conserven únicamente los puntos que puedan pertenecer a objetos de interés.

Este preprocesamiento resulta esencial en el contexto de *Formula Student Driverless*, ya que la detección fiable de conos requiere trabajar con nubes depuradas y eficientes que permitan operar en tiempo real. Para lograrlo, en la literatura se proponen un conjunto de algoritmos destinados a filtrar, segmentar y simplificar la nube de puntos, los cuales constituyen la base sobre la que se desarrollan las siguientes fases del *pipeline* de percepción. Esta información “en bruto” presenta varios retos:

- **Gran volumen de datos:** Un sensor LiDAR puede llegar a generar cientos de miles de puntos por segundo, lo que implica un alto coste computacional si no se optimiza el procesamiento.
- **Ruido y artefactos:** Reflexiones no deseadas, interferencias ambientales o errores de medición que introducen datos erróneos.
- **Información redundante:** Muchas zonas del entorno generan múltiples puntos que no aportan valor adicional.

El procesamiento de nubes de puntos suele estructurarse en un *pipeline* compuesto por varias etapas encadenadas:

1. **Filtrado y reducción de ruido**, mediante técnicas como SOR (*Statistical Outlier Removal*) o ROR (*Radius Outlier Removal*).
2. **Downsampling**, con algoritmos como *Voxel Grid* para disminuir el número de puntos.
3. **Segmentación estructural**, por ejemplo eliminando el suelo con RANSAC o separando superficies homogéneas mediante *Region Growing*.
4. **Clustering**, agrupando los puntos en objetos individuales (Véase la Sección 3.4).
5. **Extracción de características y clasificación**, donde cada cluster se describe con atributos geométricos o se procesa con técnicas de aprendizaje automático (Véase la Sección 3.5).

Este *pipeline* proporciona un marco de referencia estandarizado que guía el diseño de sistemas de percepción, y conecta directamente con los apartados siguientes dedicados al *clustering* y a la *clasificación*. A continuación, se describen algunos de los métodos más empleados en el preprocesamiento de nubes de puntos.

3.3.1. Eliminación de ruido

Dentro del preprocesamiento, una de las primeras tareas consiste en suprimir puntos espurios o mediciones erróneas. Para ello, se emplean técnicas de filtrado que analizan la distribución local de vecinos en torno a cada punto. A continuación se revisan los dos más habituales.

- ***Statistical Outlier Removal (SOR)*** [24]

El filtrado estadístico de *outliers* parte de la premisa de que, en una nube de puntos bien muestreada, la distancia media entre un punto y sus vecinos más cercanos sigue una distribución aproximadamente Gaussiana. Sea $\mathcal{P} = \{p_i \in \mathbb{R}^3 \mid i = 1, \dots, N\}$ la nube de puntos, para cada punto p_i se calcula la distancia media a sus k vecinos más próximos:

$$\bar{d}_i = \frac{1}{k} \sum_{j=1}^k \|p_i - \mathcal{N}_j(p_i)\|_2$$

donde $\mathcal{N}_j(p_i)$ denota el j -ésimo vecino más cercano de p_i y $\|\cdot\|_2$ es la norma Euclídea. Posteriormente, se obtiene la media global μ_d y la desviación estándar σ_d de todas las distancias medias $\bar{d}_i$. Los puntos que cumplen:

$$\bar{d}_i > \mu_d + \alpha \sigma_d$$

con α un factor de umbral (típicamente $\alpha \in [1, 3]$) se consideran *outliers* y se eliminan. Este método es robusto frente a ruido aleatorio y errores de medición dispersos, aunque su eficacia depende de que la distribución local de distancias sea relativamente homogénea.

En sensores LiDAR multicapa, la separación entre puntos es marcadamente *anisótropa*: la densidad azimutal es alta dentro de cada anillo mientras que la separación entre anillos (limitados en número, 32 en el caso del sensor utilizado en este proyecto) es mucho mayor.

Esta anisotropía puede sesgar los umbrales globales (μ_d, σ_d) y llevar a etiquetar como atípicos puntos válidos en regiones intrínsecamente más ralas (bordes de anillo o zonas con menor vecindad vertical). En la práctica, pueden emplearse variantes con normalización por anillo o por bandas de rango, o estimadores robustos (mediana/MAD (*Mean Absolute Deviation*)) para mitigar este efecto.

■ **Radius Outlier Removal (ROR)** [25]

En este enfoque, se define un radio r y un número mínimo de vecinos m . Para cada punto p_i , se calcula:

$$n_i = |\{p_j \in \mathcal{P} \mid \|p_i - p_j\|_2 < r\}|,$$

donde $|\cdot|$ denota la cardinalidad del conjunto.

Si $n_i < m$, el punto p_i se clasifica como ruido y se elimina. Este método es particularmente efectivo para filtrar ruido aislado, ya que puntos espurios suelen carecer de densidad local suficiente para superar el umbral.

Sin embargo, al usar un radio fijo r en sensores de muestreo *angular*, la separación entre puntos crece con la distancia al sensor, por lo que el número de vecinos dentro de r disminuye a medida que los objetos están más lejos.

Además, el número finito de anillos limita la vecindad vertical incluso a distancias cortas.

Como consecuencia, objetos lejanos o estrechos pueden no alcanzar el mínimo de vecinos m y ser descartados pese a ser válidos. Para paliarlo, se emplean radios dependientes de la distancia $r(r)$, radios *anisótropos* (diferentes en plano horizontal y vertical) o variantes aplicadas por anillo/capa.

3.3.2. Reducción de la densidad de puntos

Una vez eliminados los *outliers*, se busca disminuir la redundancia en la nube, es decir, reducir el número de puntos sin perder la estructura geométrica esencial. Esta etapa es clave, porque los sensores LiDAR generan cientos de miles de puntos por segundo, y procesarlos directamente puede resultar inviable en aplicaciones en tiempo real.

El método más extendido es el **Voxel Grid Filtering** [26]. La idea básica consiste en dividir el espacio tridimensional en una malla regular de celdas cúbicas (*vóxeles*) de lado ℓ . Cada voxel V_q agrupa un subconjunto de puntos $\mathcal{P}_q$, que se reemplazan por un único punto representativo, generalmente el centroide:

$$c_q = \frac{1}{|\mathcal{P}_q|} \sum_{p \in \mathcal{P}_q} p$$

El conjunto reducido de centroides $\{c_q\}$ constituye la nueva nube. De esta forma, se reduce el número de puntos de N a $N' \ll N$, preservando la forma global y disminuyendo el coste computacional. La Figura 3.3 contiene un ejemplo de esta transformación:

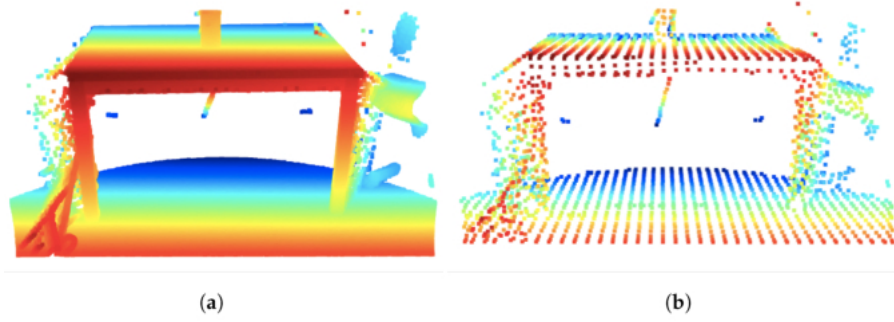


Figura 3.3: Ejemplo de Voxel Grid Filtering en una nube de puntos [27]. La imagen (a) muestra la nube de puntos original y la imagen (b), la nube procesada.

En la práctica, la elección del tamaño del voxel ℓ supone un compromiso entre eficiencia y precisión: valores pequeños conservan más detalle pero ofrecen poco ahorro, mientras que valores grandes simplifican demasiado la nube y pueden hacer desaparecer objetos pequeños como conos.

Además, como ya se ha mencionado, en sensores LiDAR multicapa la distribución de puntos no es homogénea. Por ello, una discretización cúbica puede resultar ineficiente, ya que elimina demasiada información vertical o introduce redundancias horizontales. De la misma forma que en el apartado anterior, se pueden realizar variantes con vóxeles adaptativos o discretizaciones anisótropas, así como estrategias específicas por anillo o por distancia.

3.3.3. Segmentación estructural

Finalmente, es necesario separar grandes estructuras del entorno que no son relevantes para la detección de conos, como el suelo o superficies continuas. En este ámbito destacan algoritmos como *RANSAC* y *Region Growing Segmentation*.

■ *Random Sample Consensus (RANSAC)* [28]:

RANSAC es un algoritmo robusto que busca ajustar un modelo geométrico $M(\theta)$ (por ejemplo, una línea o un plano) a un conjunto de datos que contiene ruido y valores atípicos (*outliers*). La idea central es que, aunque muchos puntos sean erróneos, basta con encontrar un subconjunto consistente que permita describir correctamente la estructura subyacente. El procedimiento general es el siguiente:

1. **Selección aleatoria:** se escoge un subconjunto mínimo de puntos $\mathcal{S} \subset \mathcal{P}$, suficiente para definir el modelo (por ejemplo, dos puntos para una línea o tres para un plano).
2. **Ajuste provisional:** se calcula el modelo $M(\theta)$ que pasa por los puntos de $\mathcal{S}$.
3. **Evaluación:** se mide cuántos puntos del conjunto total $\mathcal{P}$ encajan con el modelo, es decir, cuántos están a una distancia menor que un umbral τ . Estos puntos se consideran *inliers* y forman el conjunto:

$$\mathcal{I}(\theta) = \{p_i \in \mathcal{P} \mid \text{dist}(p_i, M(\theta)) < \tau\}$$

4. **Iteración:** se repite el proceso un número K de veces con distintos subconjuntos aleatorios $\mathcal{S}$, guardando el modelo con el mayor número de *inliers*.

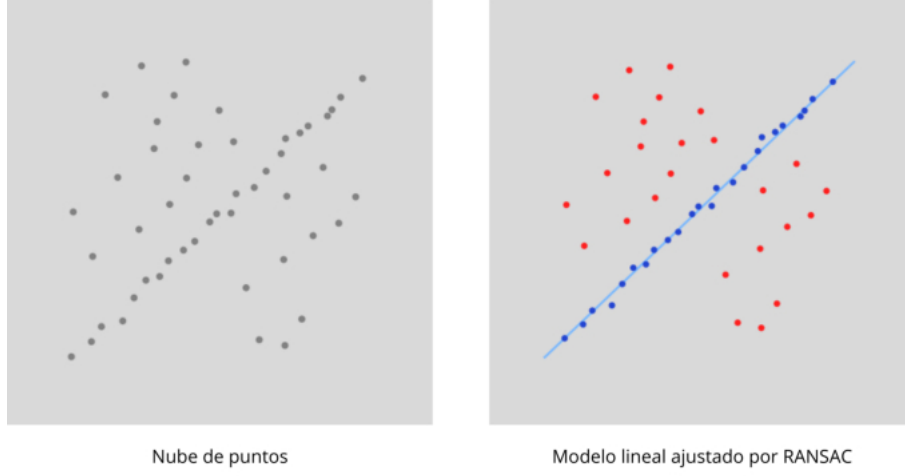


Figura 3.4: Ejemplo de modelo lineal ajustado por RANSAC (Wikipedia).

En aplicaciones con LiDAR, $M(\theta)$ suele ser un plano, lo que permite separar el suelo de los objetos. RANSAC es especialmente útil porque puede encontrar el plano correcto incluso cuando una parte considerable de los puntos corresponde a ruido o estructuras irrelevantes.

Entre sus ventajas está su simplicidad y robustez frente a *outliers*. Sin embargo, presenta limitaciones: el resultado depende de los parámetros K y τ , puede requerir mucho tiempo de cómputo si la nube es muy densa, y no siempre garantiza el modelo óptimo, sino uno suficientemente bueno.

■ *Region Growing Segmentation* [29]

Este método se basa en la idea de que los puntos que forman parte de la misma superficie (por ejemplo, el suelo o una pared) presentan propiedades similares en su geometría local. El algoritmo funciona de manera iterativa, empezando con un punto inicial (semilla) y expandiéndose poco a poco mediante la incorporación de puntos vecinos que cumplan criterios de similitud. El proceso puede dividirse en tres fases principales:

1. **Selección de una semilla:** se escoge un punto p_s de la nube. Puede elegirse de manera aleatoria o siguiendo algún criterio (por ejemplo, el punto más bajo para segmentar el suelo).
2. **Búsqueda de vecinos:** se identifican los puntos p_j cercanos a p_s , definidos como aquellos situados a una distancia inferior a un umbral $d_{\text{máx}}$.
3. **Criterio de similitud:** para cada vecino p_j se compara la orientación de la superficie local en torno a p_s y p_j . Dicha orientación se obtiene estimando la **normal** en cada punto:
 - Se toman sus k vecinos más cercanos
 - Se ajusta un plano local mediante Análisis de Componentes Principales (PCA), y la normal $\mathbf{n}$ se define como el vector perpendicular a ese plano.
 - Si la diferencia angular entre $\mathbf{n}_s$ y $\mathbf{n}_j$ ($\angle(\mathbf{n}_s, \mathbf{n}_j)$) es menor que un umbral $\phi_{\text{máx}}$, y además la distancia $\|p_s - p_j\|_2$ es inferior a $d_{\text{máx}}$, el punto se añade a la región.

De forma compacta, la condición de crecimiento puede expresarse como:

$$\angle(\mathbf{n}_s, \mathbf{n}_j) < \phi_{\text{máx}} \quad \wedge \quad \|p_s - p_j\|_2 < d_{\text{máx}}$$

Cada nuevo punto añadido pasa a ser una semilla a su vez, repitiéndose el proceso hasta que no se encuentren más vecinos que cumplan los criterios.

El resultado es un conjunto de regiones homogéneas que corresponden a superficies continuas dentro de la escena.

Este enfoque es intuitivo y efectivo para segmentar superficies planas y regulares, aunque presenta limitaciones: es sensible a la elección de los parámetros $\phi_{\text{máx}}$ y $d_{\text{máx}}$, puede fragmentar objetos si las condiciones son demasiado estrictas, o unir regiones distintas si son demasiado laxas.

Ambos métodos se pueden utilizar de forma complementaria, aunque presentan diferencias notables. RANSAC resulta muy adecuado para extraer modelos geométricos simples y dominantes en la escena, como planos o líneas, siendo por ello una opción muy empleada para segmentar el suelo en aplicaciones con LiDAR. Su fortaleza radica en la robustez frente a *outliers*, pero depende fuertemente de sus parámetros, el número de iteraciones K y el umbral de distancia τ , y puede resultar costoso en nubes densas.

Por otro lado, *Region Growing Segmentation* es más flexible para detectar superficies continuas con cierta complejidad geométrica. Permite agrupar puntos en regiones homogéneas sin necesidad de un modelo previo, pero requiere una estimación fiable de normales y una cuidadosa selección de los parámetros $\phi_{\text{máx}}$ y $d_{\text{máx}}$. En la práctica, típicamente RANSAC se emplea de forma preliminar para eliminar estructuras globales como el suelo, mientras que *Region Growing* se utiliza en fases posteriores para refinar la segmentación de superficies u objetos específicos.

Además, la literatura recoge otras aproximaciones, como algoritmos de **clustering** (por ejemplo DBSCAN) aplicados a la segmentación, técnicas basadas en proyecciones a coordenadas polares para separar el suelo de manera más eficiente, o métodos avanzados como la segmentación por **supervoxeles**, que agrupan puntos en regiones regulares aunque con un mayor coste computacional.

En el caso concreto de *Formula Student*, la segmentación del suelo mediante RANSAC es especialmente útil, ya que permite descartar gran parte de los puntos irrelevantes, pues las pruebas se realizan en circuitos planos, y concentrar el procesamiento en la detección de conos.

3.4. Clustering en nubes de puntos

Tras la etapa de filtrado y eliminación de ruido, es necesario agrupar los puntos en función de su pertenencia a objetos u obstáculos individuales. Este proceso, denominado *clustering*, permite segmentar la nube en subconjuntos coherentes que representan entidades físicas diferenciadas en el entorno.

En el contexto de este proyecto, el objetivo principal es identificar de manera fiable los conos que delimitan el circuito. Estos elementos son pequeños y tras la segmentación estructural, aparecen aislados en la nube, por lo que el algoritmo de *clustering* debe ser capaz de generar grupos compactos a partir de un número limitado de puntos, evitando a su vez confundir ruido residual con objetos reales.

A continuación, se describen algunos de los métodos más empleados, junto con sus ventajas y limitaciones.

3.4.1. Euclidean Cluster Extraction

El método más extendido en nubes de puntos es el *Euclidean Cluster Extraction* [30], que se basa únicamente en la proximidad espacial entre puntos. Dado un punto p_i , se identifican sus

vecinos $\mathcal{N}(p_i)$ mediante una búsqueda de radio r . Estos vecinos se añaden al mismo *cluster*, y a su vez se repite el proceso con ellos, haciendo crecer el grupo hasta que no se encuentran más puntos cercanos.

A partir de un punto semilla, se construye un *cluster* C_k como:

$$C_k = \{p_j \in \mathcal{P} \mid \|p_i - p_j\|_2 < r, \forall p_i \in C_k\}$$

El crecimiento continúa hasta que no se encuentran más vecinos dentro del radio r . De este modo, se generan varios *clusters* $\{C_1, C_2, \dots, C_M\}$ que representan objetos individuales.

Este método es conceptualmente simple y ofrece buenos resultados cuando los objetos están claramente separados. Sin embargo, su rendimiento depende en gran medida del valor de r : un radio demasiado pequeño fragmenta un mismo objeto en varios *clusters*, mientras que un radio demasiado grande puede fusionar elementos distintos.

3.4.2. K-means

El algoritmo *k-means* [31] es uno de los métodos de *clustering* no supervisado más empleados en el procesamiento de nubes de puntos debido a su simplicidad y eficiencia computacional.

Su objetivo es particionar un conjunto de puntos en k *clusters*, minimizando la varianza intra-*cluster*. Formalmente, el problema se plantea como la minimización de la siguiente función de coste:

$$J = \sum_{i=1}^k \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

donde C_i representa el conjunto de puntos pertenecientes al *cluster* i , y $\boldsymbol{\mu}_i$ es el centroide correspondiente.

El algoritmo sigue un proceso iterativo en dos pasos:

1. Asignación de cada punto al *cluster* cuyo centroide se encuentre más cercano (p.ej. en distancia Euclídea)
2. Actualización de los centroides como la media de los puntos asignados a cada *cluster*.

En el contexto de nubes de puntos, *k-means* se utiliza para segmentar la escena en regiones coherentes, aunque presenta limitaciones: (i) requiere conocer de antemano el número de *clusters* k , lo cual no siempre es evidente, (ii) tiende a favorecer agrupaciones esféricas cuando se utiliza la distancia Euclídea, lo que lo hace poco adecuado para distribuciones no convexas o con ruido significativo. En este proyecto no sería adecuado puesto que no se dispone de información acerca del número de conos que se deben ver.

3.4.3. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

El algoritmo DBSCAN [32] es especialmente adecuado para nubes de puntos, ya que permite detectar *clusters* de forma arbitraria y manejar el ruido de manera explícita. Dado un radio ε y un número mínimo de vecinos m , se definen tres tipos de puntos:

- **Puntos núcleo:** aquellos con al menos m vecinos en un radio ε .
- **Puntos frontera:** aquellos que no cumplen el criterio de núcleo, pero son vecinos de un punto núcleo.
- **Ruido:** puntos que no cumplen ninguno de los criterios anteriores.

El conjunto de un *cluster* se construye como la unión transitiva de puntos núcleo y sus vecinos alcanzables:

$$C_k = \bigcup \{p_j \in \mathcal{P} \mid \text{reach}(p_j, p_i, \varepsilon, m) = \text{true}\},$$

donde $\text{reach}(\cdot)$ indica si un punto es alcanzable desde otro a través de una secuencia de puntos núcleo.

En la Figura 3.5 se muestra un ejemplo de agrupación por DBSCAN, donde los puntos rojos son puntos núcleo; los amarillos, puntos frontera; y el azul, es considerado ruido.

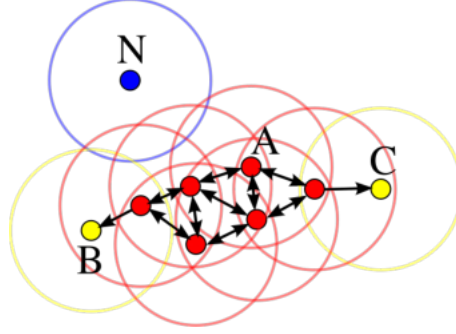


Figura 3.5: Ejemplo de *dataset* agrupado por DBSCAN con $m = 4$ (Wikipedia).

DBSCAN es robusto frente al ruido y valores atípicos. Además, no requiere especificar el número de *clusters* *a priori*, lo que lo convierte en una técnica idónea para entornos no estructurados.

Sus principales limitaciones son la sensibilidad a los parámetros ε y m , y el hecho de que la variación de densidad en la nube puede dificultar su rendimiento en escenas heterogéneas.

3.4.4. Clustering jerárquico

El *clustering jerárquico* [33] construye una estructura en forma de árbol (*dendrograma*) que agrupa los puntos de manera recursiva, lo que permite analizar la organización de los puntos a distintos niveles de granularidad. En la variante *agglomerative*, se parte de cada punto como un *cluster* individual y se fusionan iterativamente los dos *clusters* más cercanos según una medida de distancia $D(C_i, C_j)$:

$$D(C_i, C_j) = \min_{p_a \in C_i, p_b \in C_j} \|p_a - p_b\|_2$$

Este proceso continúa hasta alcanzar un número deseado de grupos o un umbral de distancia máximo.

El enfoque tampoco requiere especificar *a priori* el número de *clusters*. No obstante, su elevada complejidad computacional lo limita a contextos de procesamiento *offline* y lo hace

menos adecuado para aplicaciones en tiempo real, como este proyecto, donde la eficiencia es crítica.

3.4.5. Comparativa entre métodos:

En términos generales, en este caso concreto, donde el objetivo es detectar conos como objetos pequeños y separados, DBSCAN y *Euclidean Cluster Extraction* son las opciones más viables por su capacidad para segmentar estructuras compactas y manejar ruido de forma eficaz.

3.5. Clasificación de objetos

Una vez que los puntos han sido agrupados en *clusters*, es necesario determinar qué objetos corresponden realmente a conos y cuáles representan otras estructuras del entorno. Esto permite filtrar los falsos positivos y conservar únicamente aquellos elementos relevantes para la navegación en el circuito. Existen principalmente tres enfoques para abordar esta tarea:

1. Filtros sencillos por reglas geométricas
2. Clasificadores clásicos sobre descriptores del *cluster*
3. Técnicas basadas en aprendizaje profundo capaces de operar directamente sobre los datos de la nube de puntos.

3.5.1. Filtros geométricos basados en reglas

Este enfoque decide si un *cluster* tiene forma compatible con un cono aplicando umbrales sobre magnitudes geométricas y estadísticas sencillas, sin entrenamiento previo.

Dimensiones métricas y proporciones

Sea el *bounding box* alineado con los ejes del *cluster* C con dimensiones h (altura), w (anchura) y d (profundidad). Un cono presenta típicamente h mayor que w, d y una relación de esbeltez dentro de un rango acotado:

$$\text{slender}(C) = \frac{h}{\max(w, d)} \in [\tau_{\min}, \tau_{\max}].$$

Adicionalmente, el volumen del *bounding box*, $V_{\text{bbox}} = h w d$, y la ocupación

$$\rho(C) = \frac{N_p}{V_{\text{bbox}}},$$

que es la razón entre el número de puntos N_p y el volumen, ayudan a descartar estructuras demasiado difusas o demasiado grandes para ser un cono.

Orientación (verticalidad)

La dirección principal del *cluster* puede estimarse mediante PCA sobre sus puntos. Sea $\mathbf{v}_1$ el autovector asociado al mayor autovalor de la matriz de covarianza y $\mathbf{e}_z$ el eje vertical, la verticalidad se evalúa como

$$\theta(C) = \arccos(|\mathbf{v}_1 \cdot \mathbf{e}_z|),$$

aceptándose el *cluster* si $\theta(C) \leq \theta_{\text{máx}}$, lo que indica un cono aproximadamente erguido.

Forma a partir de autovalores

Con los autovalores $\lambda_1 \geq \lambda_2 \geq \lambda_3$ de la matriz de covarianza, se definen tres métricas estándar:

$$\text{linealidad } L = \frac{\lambda_1 - \lambda_2}{\lambda_1}, \quad \text{planaridad } P = \frac{\lambda_2 - \lambda_3}{\lambda_1}, \quad \text{esfericidad } S = \frac{\lambda_3}{\lambda_1}.$$

Los conos reales suelen mostrar L y P moderados y S no extremo (ni casi plano ni puramente lineal), lo que permite descartar superficies planas (suelo, paredes) o estructuras filiformes.

Conteo adaptativo de puntos

En sensores LiDAR de muestreo angular, el espaciado entre impactos crece con la distancia d :

$$\Delta s_h \approx d \Delta \theta, \quad \Delta s_v \approx d \Delta \phi,$$

donde $\Delta \theta$ y $\Delta \phi$ son las resoluciones angulares horizontal y vertical. Para evitar rechazar conos lejanos por baja densidad de puntos, los umbrales sobre N_p y los radios de vecindad se ajustan con d (p. ej., radios proporcionales a d o mínimos de puntos $N_{\text{mín}}(d)$ decrecientes con la distancia).

Ajuste a un modelo cónico

Otra estrategia es verificar la compatibilidad con un modelo geométrico. Sea un cono de ápice $\mathbf{a}$, eje unitario $\mathbf{u}$ y semiángulo α , para un punto $\mathbf{x}$ del *cluster*, la distancia angular al eje se define como

$$\delta(\mathbf{x}) = \arccos \left(\frac{|(\mathbf{x} - \mathbf{a}) \cdot \mathbf{u}|}{\|\mathbf{x} - \mathbf{a}\|_2} \right).$$

Un *cluster* es aceptado si un porcentaje alto de sus puntos cumple $|\delta(\mathbf{x}) - \alpha| \leq \varepsilon$. Este test puede robustecerse con RANSAC (Véase la Sección 3.3.3) para el ajuste de $(\mathbf{a}, \mathbf{u}, \alpha)$.

Estos criterios son interpretables, eficientes y adecuados para tiempo real; su eficacia depende de umbrales bien calibrados (idealmente, dependientes de distancia y capa de escaneo) y pueden fallar ante oclusiones o formas atípicas.

3.5.2. Métodos clásicos sobre descriptores

En lugar de decidir con reglas fijas, es posible extraer un vector de características de cada *cluster* y entrenar un clasificador supervisado cono/no-cono.

Extracción de características

Un vector típico puede incluir:

- Dimensiones (h, w, d)
- Relaciones $(h/w, h/d)$
- Volumen (V_{bbox})

- Ocupación (ρ)
- Autovalores ($\lambda_1, \lambda_2, \lambda_3$)
- Métricas (L, P, S)
- Número de puntos (N_p)
- Agregados de reflectividad (media, varianza, histogramas)

Clasificadores clásicos supervisados

Tras la extracción del vector de características de cada *cluster*, la clasificación cono/no-cono puede abordarse mediante clasificadores supervisados ligeros:

- **Máquina de vectores de soporte (SVM) [34]:**

El objetivo es determinar un hiperplano que maximice el margen entre clases. En su formulación con pérdidas (*hinge*) y regularización, debe resolverse el siguiente problema de optimización:

$$\min_{w, b, \{\xi_i\}} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad \text{s.a.} \quad y_i(w^\top \varphi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0,$$

donde C controla el compromiso entre complejidad del modelo y errores, y $\varphi(\cdot)$ puede ser implícita vía *kernels* (p.ej., lineal o RBF) para capturar fronteras no lineales. La selección de hiperparámetros (p.ej., C y, en RBF, γ) se realiza habitualmente mediante validación cruzada estratificada, que divide los datos en pliegues manteniendo la misma proporción de clases que en el conjunto original. Es recomendable estandarizar las variables de entrada para garantizar escalas comparables, es decir, transformar cada variable para que tenga media 0 y desviación estándar 1.

- **Random Forest [35]:**

Es un clasificador de conjunto que agrega T árboles de decisión entrenados sobre réplicas *bootstrap* del conjunto de datos e introduce aleatoriedad adicional al considerar sólo un subconjunto de variables en cada división. La predicción se obtiene por votación mayoritaria (o promediando probabilidades de hoja):

$$\hat{y}(x) = \arg \max_{c \in \mathcal{C}} \frac{1}{T} \sum_{t=1}^T \mathbf{1}[h_t(x) = c], \quad \hat{P}(y = c | x) = \frac{1}{T} \sum_{t=1}^T \hat{P}_t(y = c | x).$$

En cada nodo, la división se elige maximizando la reducción de impureza (p.ej., el índice de Gini [36]):

$$G(S) = 1 - \sum_c p_c^2, \quad \Delta G = G(S) - \frac{|S_L|}{|S|} G(S_L) - \frac{|S_R|}{|S|} G(S_R).$$

El error *out-of-bag* (OOB) [37] permite la estimación interna del rendimiento agregando sólo los árboles que no han visto cada muestra durante el entrenamiento. Los hiperparámetros más influyentes son el número de árboles T , el número de variables candidatas por nodo (m_{try}), la profundidad máxima y el tamaño mínimo de hoja; el desbalance puede tratarse con ponderación de clases. Es un método robusto a ruido y a *features* heterogéneas, con baja sensibilidad al reescalado, a costa de un mayor coste de inferencia que un árbol individual y de probabilidades no calibradas salvo ajuste posterior.

- **KNN** [38]:

Asigna la clase en función de la mayoría entre los k vecinos más próximos en el espacio de características:

$$\hat{y}(x) = \arg \max_c \sum_{i \in \mathcal{N}_k(x)} \mathbf{1}[y_i = c].$$

Su rendimiento depende de la métrica de distancia (Euclídea, Mahalanobis, etc.) y del valor de k , que se seleccionan por validación cruzada. Requiere normalizar/estandarizar las variables para evitar sesgos por escalas diferentes y puede resultar costoso en inferencia con conjuntos de entrenamiento grandes, si bien estructuras de búsqueda (*KD-Tree*, *Ball-Tree*) mitigan parcialmente este efecto en baja dimensión.

Consideraciones. Se suelen evaluar estos clasificadores mediante validación cruzada estratificada y métricas por clase (precisión, *recall*, F1 y matriz de confusión), además de *accuracy*. En LiDAR es habitual normalizar o diseñar características invariantes respecto a la distancia y la anisotropía por anillos, y emplear agregados de reflectividad robustos. Cuando hay desbalance o costes asimétricos, se recurre a ponderación de clases o ajuste del umbral de decisión.

3.5.3. Aprendizaje profundo sobre nubes de puntos

Estos modelos aprenden directamente a partir de la geometría del *cluster*, evitando el diseño manual de descriptores. Las aproximaciones más empleadas son:

- **Convolutional Neural Network 3D (CNN3D)** [39]:

La nube se discretiza en una rejilla $X \in \mathbb{R}^{H \times W \times D \times C}$ (ocupación, intensidad, etc.) y se aplican convoluciones volumétricas para capturar patrones espaciales jerárquicos. En este tipo de redes la relación entrada-salida de la capa ℓ -ésima es la siguiente:

$$h_{i,j,k}^{(\ell)} = \sigma \left(\sum_{u,v,w} W_{u,v,w}^{(\ell)} x_{i+u, j+v, k+w}^{(\ell-1)} + b^{(\ell)} \right).$$

Ofrecen alta capacidad, si bien el coste de memoria/cómputo crece con la resolución; variantes con convoluciones dispersas mitigan este efecto en nubes escasas.

- **PointNet y variantes** [40]:

Se trata la nube como un conjunto no ordenado $\mathcal{P} = \{p_i\}_{i=1}^N$; se aplican MLP (*Multi-Layer Perceptron*) punto a punto y una agregación simétrica (SYM) que garantiza invariancia al orden:

$$f(\mathcal{P}) = \gamma \left(\text{SYM}_i \{ h(p_i) \} \right) \quad (\text{p. ej., SYM} = \text{max-pooling}).$$

Extensiones jerárquicas (p. ej., PointNet++) introducen vecindarios locales a múltiples escalas para capturar contexto geométrico.

- **Modelos basados en grafos (GNN (*Graph Neural Network*)/ DGCNN (*Dynamic Graph CNN*))** [41]:

Se construye un grafo de vecindad (k -NN) y se actualizan representaciones mediante agregación de mensajes. En DGCNN, la operación *EdgeConv* sobre el nodo i es:

$$h_i^{(\ell+1)} = \text{AGG}_{j \in \mathcal{N}(i)} \phi(h_i^{(\ell)}, h_j^{(\ell)} - h_i^{(\ell)}),$$

donde AGG suele ser **max** o **sum** y ϕ es una MLP que aprende relaciones locales. En otras palabras, *EdgeConv* modela las interacciones locales entre un punto y sus vecinos considerando tanto sus características absolutas como relativas.

Discusión. Las CNN 3D proporcionan gran capacidad a costa de cuantización y uso de memoria; PointNet es eficiente e invariante al orden, con mejor desempeño al incorporar jerarquías locales; las GNN/DGCNN explotan la estructura de vecindad con alta precisión, a mayor coste computacional y de anotación. La elección depende del compromiso entre recursos, volumen de datos disponibles y requisitos de tiempo real.

3.6. Caso de estudio: sistema de percepción de TU Dresden (FSG 2024)

Además de los métodos y arquitecturas revisados en los apartados anteriores, resulta de interés considerar las soluciones adoptadas por equipos punteros de *Formula Student Driverless*, pues constituyen un entorno altamente competitivo donde se validan distintas arquitecturas de percepción.

Un ejemplo representativo es el equipo Elbflorace (TU Dresden), ganador de las pruebas *Acceleration* y *Driverless Acceleration* en la edición 2024 de Formula Student Germany [42], una de las competiciones más prestigiosas. En dichas pruebas, su vehículo autónomo alcanzó un tiempo de 3,294 s, superando incluso el registro con piloto humano (3,380 s). Este hito evidencia que un *stack* autónomo bien diseñado puede igualar o superar el rendimiento manual en condiciones de competición real.

Sus miembros presentaron su sistema autónomo en la conferencia *Beating the Human Driver* [43], impartida en la *FSG Academy*. Su sistema de percepción se basa en un conjunto de tres sensores LiDAR Livox HAP [44], cada uno con un campo de visión de $120^\circ \times 25^\circ$. La fusión de estos dispositivos proporciona aproximadamente 300° de cobertura.

El flujo de trabajo del equipo comienza con la acumulación de barridos parciales y la compensación del movimiento del vehículo:

1. Se combinan los datos de los distintos LiDAR en un marco común.
2. Se acumulan escaneos internos a 100 Hz para generar una nube de mayor resolución, con una frecuencia efectiva de 15 Hz en la prueba de aceleración y 10 Hz en *autocross*.
3. Se corrige el movimiento propio del vehículo durante el período de acumulación.

Una vez obtenida la nube densa, el *pipeline* aplica las siguientes etapas:

1. Eliminación de suelo y estructuras irrelevantes.
2. *Clustering* de puntos candidatos a conos.
3. Ajuste de cada *cluster* a una plantilla geométrica de cono para aumentar la precisión.
4. Filtrado final de falsos positivos.

Aunque no detallaron en su exposición el uso de cámaras, se conoce que emplean una *Basler acA4096-11gc* con óptica *Kowa LM6HC*, presumiblemente destinada a la clasificación por color y filtrado de falsos positivos.

Con esta estrategia, el sistema alcanza distancias de detección de entre 50 y 70 metros, lo que permite al vehículo anticipar gran parte del trazado desde la salida. El coste de cada sensor Livox HAP es de aproximadamente 1 400 euros, lo que convierte a esta arquitectura en una alternativa

más económica que soluciones basadas en LiDAR de 360 grados, además de ofrecer un campo de visión más ajustado a la tarea de detección de conos en circuito.

En comparación con el sistema desarrollado en este TFG, basado en un único LiDAR Ouster OS1-32, la propuesta de TU Dresden prioriza la redundancia angular y la densidad de puntos frente a la simplicidad y optimización temporal. Mientras que su enfoque busca maximizar cobertura y robustez, el presente trabajo apuesta por un diseño minimalista y eficiente, capaz de cumplir en tiempo real con las restricciones de *hardware* embarcado.

Capítulo 4

Desarrollo del proyecto

4.1. Introducción al desarrollo

El presente capítulo tiene como objetivo describir el proceso de desarrollo e implementación de un sistema de percepción basado únicamente en un sensor LiDAR para un vehículo autónomo de *Formula Student*. El propósito es mostrar de forma estructurada las etapas que permiten transformar la nube de puntos cruda, generada por el sensor, en una representación del entorno adecuada para la posterior planificación de la trayectoria.

El sistema diseñado no se limita a un análisis teórico, sino que ha sido implementado e integrado en un vehículo de competición. Esto implica que cada decisión técnica ha estado condicionada por restricciones reales de funcionamiento: limitaciones del propio sensor, limitaciones de tiempo de cómputo, condiciones variables del entorno en el circuito y necesidad de garantizar una detección fiable en todo momento, además de su compatibilidad con el resto de sistemas.

Así, en este capítulo se detallan tanto las técnicas empleadas como las decisiones de diseño adoptadas, exponiendo además las dificultades encontradas durante el proceso y las soluciones implementadas. El objetivo final es construir un *pipeline* robusto y eficiente que sea capaz de procesar los datos del LiDAR en tiempo real y aportar al sistema de conducción autónoma información precisa sobre la posición y el color de los conos.

4.1.1. Contexto en Fórmula Student

En la competición *Formula Student Driverless*, la percepción del entorno constituye el primero de los retos al que el vehículo se debe enfrentar. De este módulo depende el resto de sistemas, ya que el trazado de la pista está delimitado exclusivamente por conos de distintos colores y tamaños. La correcta detección y clasificación de estos conos es esencial para que el vehículo pueda generar un mapa navegable del circuito y ejecutar de forma segura las tareas de planificación y control.

En un principio, el sistema de percepción del equipo recurría a un esquema multisensorial que incluía tanto cámaras como el sensor LiDAR. Las cámaras se empleaban principalmente para la clasificación de conos por color, mientras que el LiDAR se destinaba a la localización espacial de los mismos. Sin embargo, durante una de las competiciones internacionales el sistema de cámaras sufrió fallos que lo dejaron inutilizable. Aunque el equipo cuenta ahora con otras cámaras, como ya se ha mencionado (Véase Sección 3.2.4) esta restricción frenó el avance necesario para alcanzar un nivel óptimo en el desarrollo de algoritmos de visión artificial o fusión sensorial. Este hecho puso de manifiesto la necesidad de contar con un *pipeline* capaz de funcionar de manera autónoma únicamente con la información proveniente del LiDAR.

Como resultado, se estableció como objetivo el desarrollo de un sistema de percepción completo basado exclusivamente en LiDAR. Esta aproximación presenta ventajas adicionales, ya que el sensor no se ve afectado por condiciones de iluminación, sombras o reflejos, lo que aumenta la robustez del sistema. En este capítulo se expone cómo se llevó a cabo dicho desarrollo e implementación, fruto del trabajo realizado en este Trabajo Final de Grado, desde la adquisición de la nube de puntos hasta la obtención de una lista de conos clasificados y localizados en el espacio.

4.2. Adquisición de datos

4.2.1. Sensor LiDAR

El sistema de percepción que se ha desarrollado en este proyecto se basa en un sensor *Ouster OS1-32* [11], mostrado en la Figura 4.1. Este sensor es un LiDAR de medio alcance diseñado específicamente para aplicaciones en robótica y automoción. Se trata de un dispositivo compacto (85 mm de diámetro y 58 mm de altura) y ligero (447 g en total y 377 g sin el disipador instalado), lo que facilita su integración en un monoplaza de *Formula Student* sin penalizar el diseño mecánico ni el reparto de masas.



Figura 4.1: Sensor Ouster OS1-32 (ouster.com)

En cuanto a prestaciones, el OS1-32 proporciona un rango máximo de detección de 120 metros bajo condiciones de alta iluminación (100 klx), con una precisión en la medida de distancias de hasta ± 3 cm en superficies lambertianas, que son aquellas que reflejan la luz de forma difusa e igual en todas las direcciones, sin brillos especulares. Esta capacidad es suficiente para detectar conos en un circuito de competición incluso a grandes distancias, facilitando en gran medida el trabajo del sistema de localización y mapeo del circuito.

El sensor dispone de 32 canales verticales que cubren un campo de visión de 45° en el plano vertical (de $+22.5^\circ$ a -22.5°), lo que supone una separación aproximada de 1.4° entre canales consecutivos. En el plano horizontal ofrece una cobertura completa de 360° , con resoluciones configurables de 512, 1024 o 2048 puntos por revolución. La máxima resolución de 2048 puntos por revolución únicamente está disponible cuando el sensor opera a 10 Hz, mientras que a 20 Hz la resolución se limita a 1024 puntos por revolución.

La separación angular entre medidas horizontales depende directamente de la resolución configurada:

- **512 puntos/rev:** $\sim 0.70^\circ$ entre puntos consecutivos.
- **1024 puntos/rev:** $\sim 0.35^\circ$.
- **2048 puntos/rev (solo a 10 Hz):** $\sim 0.175^\circ$.

De este modo, a mayor resolución se obtiene una nube de puntos más densa y detallada, lo que facilita la detección de objetos pequeños como los conos, aunque también incrementa el volumen de datos a procesar en tiempo real.

En la Figura 4.2 se muestran las diferentes configuraciones de resolución disponibles en el sensor OS1-32, que ilustran cómo varía la densidad de puntos en función del modo de operación. El cono y los neumáticos de la figura se encuentran aproximadamente a 3.4 m del sensor.

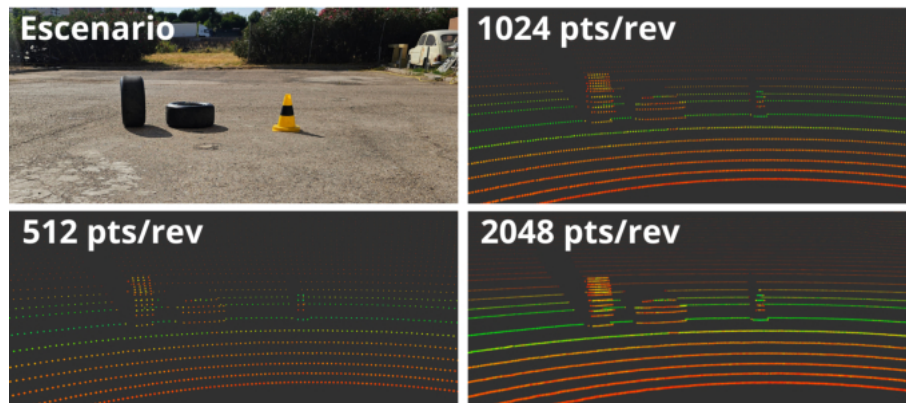


Figura 4.2: Comparación de los modos de resolución disponibles en el LiDAR Ouster OS1-32.

Un aspecto relevante es la precisión en el muestreo angular, especificada en $\pm 0,01^\circ$ tanto en vertical como en horizontal. Esto significa que la orientación de cada haz láser puede diferir en ese margen respecto a su valor nominal. En términos prácticos, a una distancia de 50 metros dicho error supone un desplazamiento transversal máximo de unos 8.7 mm, un valor despreciable para la tarea de delimitación del circuito, y que pone de manifiesto el nivel de exactitud del sensor.

Cada punto medido incluye, además de la distancia a la que se encuentra del sensor y el ángulo de azimut, información de reflectividad, intensidad en infrarrojo cercano, canal de origen y marca temporal con una resolución inferior a $1 \mu s$. Adicionalmente, el sensor incorpora una unidad inercial (IMU) que transmite datos de aceleración y velocidad angular a 100 Hz, útiles para la compensación de movimientos del vehículo, aunque no será utilizada en este proyecto.

Finalmente, el OS1-32 cuenta con un grado de protección IP68/IP69K, un rango de operación entre $-40^\circ C$ y $+60^\circ C$, y un consumo eléctrico de entre 14 y 20 W, características que garantizan su funcionamiento fiable en condiciones de competición exigentes.

4.2.2. Formato de los datos

Para la integración del OS1-32 en *ROS 2* se emplea el paquete `ros2_ouster_drivers` [45], que convierte los paquetes UDP del sensor en mensajes estándar y los publica en distintos *topics* mediante nodos *lifecycle* y *components*.

Tópicos publicados (principales):

- `/points` (`sensor_msgs/PointCloud2`): nube 3D generada a partir de una rotación completa del LiDAR.
- `/range_image` (`sensor_msgs/Image`): imagen de rango.
- `/intensity_image` (`sensor_msgs/Image`): imagen de intensidad.
- `/noise_image` (`sensor_msgs/Image`): imagen de ruido.
- `/reflectivity_image` (`sensor_msgs/Image`): imagen de reflectividad.
- `/scan` (`sensor_msgs/LaserScan`): proyección 2D del anillo más cercano a 0° de azimut.
- `/imu` (`sensor_msgs/Imu`): medidas de la IMU a la frecuencia de transmisión.

Contenido de la nube de puntos.

El procesado por defecto del driver almacena una rotación completa y publica un `PointCloud2` con los campos: `X`, `Y`, `Z`, `range`, `intensity`, `reflectivity`, `ring` y `noise`. Opcionalmente puede omitirse el punto $(0,0,0)$ con `pointcloud_filter_zero_points` (`False` por defecto).

Parámetros relevantes:

- `lidar_ip`, `computer_ip`, `lidar_mode` (por defecto 512x10), `imu_port` (7503), `lidar_port` (7502).
- `sensor_frame`, `laser_frame`, `imu_frame` (nombres de *frames* TF).
- `use_system_default_qos` (`False` por defecto) para facilitar la grabación con `rosviz`.
- `timestamp_mode`: `TIME_FROM_INTERNAL_OSC`, `TIME_FROM_SYNC_PULSE_IN`, `TIME_FROM_PTP_1588`, `TIME_FROM_ROS_RECEPTION`. Esta opción determina cómo se sellan temporalmente las medidas; en aplicaciones móviles se desaconseja `TIME_FROM_ROS_RECEPTION`.

En este proyecto se ha trabajado principalmente con la nube de puntos publicada en el *topic* `/points`, que concentra toda la información para el desarrollo del *pipeline* de percepción.

El sistema se ha diseñado para funcionar en el modo 2048x10, que proporciona la máxima resolución angular disponible en el OS1-32. La elección de este modo responde a la necesidad de detectar conos a distancias considerables, lo que requiere una nube de puntos densa incluso a costa de renunciar a la frecuencia de 20 Hz.

En la práctica, los huecos temporales entre dos rotaciones consecutivas (0.1 s a 10 Hz) son asumibles, ya que el vehículo no recorre distancias excesivas en ese intervalo. En la Tabla 4.1 se muestra el desplazamiento lineal aproximado para diferentes velocidades:

Velocidad	Avance en 0.1 s
10 km/h	0.28 m
30 km/h	0.83 m
60 km/h	1.67 m
100 km/h	2.78 m

Tabla 4.1: Avance aproximado del vehículo en 0.1 s para distintas velocidades.

Como puede observarse, incluso a velocidades muy superiores a las que se alcanzan en *Formula Student Driverless* (que suelen moverse entre 30 y 60 km/h), el desplazamiento en 0.1 s es reducido en comparación con el rango de detección del LiDAR, por lo que la pérdida de frecuencia no compromete la fiabilidad del sistema.

Por otra parte, Ouster proporciona una API que permite configurar ciertos aspectos del sensor mediante peticiones HTTP. Esta API se maneja habitualmente a través de comandos `curl`, lo que posibilita modificar parámetros sin necesidad de reiniciar el dispositivo ni recompilar el *software*. Entre las opciones disponibles se encuentran, por ejemplo, la limitación del ángulo de barrido o el paso a modo *standby*, que permite apagar el sensor de forma segura sin desconectarlo físicamente.

Dado que en *Formula Student* el sector trasero del LiDAR queda mayoritariamente ocluido por la propia estructura del vehículo (carrocería, neumáticos, etc.), la adquisición en 360° aporta un retorno útil marginal para la tarea de detección de conos. Por ello, se configuró el sensor para un barrido de 180° ($\pm 90^\circ$ respecto al eje longitudinal), reduciendo aproximadamente a la mitad el

volumen de datos desde el propio dispositivo sin pérdida de información relevante para planificación y control. Esta decisión no impide la detección en curvas de radio reducido, puesto que el campo de visión frontal de 180° preserva la cobertura lateral necesaria para capturar los conos de salida; las limitaciones en esos escenarios vienen determinadas por oclusión geométrica y no por el ángulo de barrido.

4.3. Punto de partida

4.3.1. Estado inicial del sistema de percepción

El sistema de percepción original se implementaba como una cadena de nodos en *ROS 2*, organizados en distintas etapas de filtrado y procesamiento de la nube de puntos. Cada nodo ejecuta una tarea concreta y comunica sus resultados al siguiente mediante *topics*. En la Figura 4.3 se muestra un esquema simplificado de esta arquitectura.



Figura 4.3: Esquema de comunicación entre nodos en el sistema de percepción inicial (captura de `rqt_graph`).

- **Nodo de entrada y prefiltrado** (`pointcloud_subscriber`): Suscribe la nube de puntos publicada por el sensor en el *topic* `/points` y aplica un primer recorte geométrico a las coordenadas x , y y z . El resultado se publica en `/prefiltered_points`.
- **Nodo de segmentación por plano** (`ransac_node`): Recibe la nube prefiltrada y ejecuta el algoritmo RANSAC para identificar y eliminar planos dominantes, en particular el suelo. Los puntos clasificados como parte del plano se publican en `/volume_inliers`, mientras que los restantes (posibles obstáculos) se publican en `/volume_outliers`.
- **Nodo de clustering** (`clustering`): Se suscribe a `/volume_outliers` y aplica el algoritmo *DBSCAN* para agrupar los puntos restantes en *clusters* candidatos a cono. Para cada *cluster* se realizan dos pasos adicionales:
 - **Estimación de posición:** se calcula un punto representativo del *cluster* mediante la media ponderada de las coordenadas de todos sus puntos, de modo que cada cono detectado queda definido por unas coordenadas en el espacio.
 - **Clasificación de color:** se aplican reglas heurísticas:
 - Los *conos naranjas* se identifican principalmente por su mayor altura en el eje z .
 - La distinción entre *azules* y *amarillos* se realiza mediante un cociente de intensidades (reflectividad) entre distintos anillos del LiDAR, aprovechando que ambos materiales presentan un comportamiento distinto frente al láser en esas bandas.

Los resultados se publican en dos formatos complementarios:

- `/clustered_points` (`PoseArray`), con la posición estimada de cada *cluster*, útil para depuración.
- `/LidarConeList` (`ConeArray`), con la lista de conos detectados, incluyendo su posición y el color asignado. Este es el mensaje consumido por otros módulos del *stack* (por ejemplo, localización y planificación).

4.3.2. Objetivos del nuevo desarrollo

Como se ha mencionado en la Introducción (Véase la Sección 1.2), los objetivos principales de este Trabajo Fin de Grado se centran en el rediseño y mejora del sistema LiDAR. Partiendo de la arquitectura descrita en la Sección anterior, estos objetivos se concretan en:

- **Aumentar el alcance de detección:** lograr que el sistema sea capaz de identificar conos a mayores distancias, manteniendo la fiabilidad incluso cuando la densidad de puntos disminuye.
- **Reducir falsos positivos:** disminuir la probabilidad de que estructuras del entorno (vegetación, bordes de pista, ruido del sensor) sean clasificadas erróneamente como conos.
- **Mejorar la clasificación de color:** incrementar la precisión en la identificación de conos azules, amarillos y naranjas, elemento fundamental para que el módulo de planificación del *stack* pueda distinguir correctamente los límites izquierdo y derecho del circuito.

Además de estos tres objetivos principales, fue necesario definir una serie de metas de carácter más técnico orientadas a la implementación práctica del sistema. Mientras que los primeros establecen el *qué* debía lograrse (ver más lejos, reducir falsos positivos y acertar en el color), los siguientes objetivos especifican el *cómo* debía alcanzarse, introduciendo mejoras en la estructura del *software*, en la eficiencia de los algoritmos y en las herramientas de validación empleadas durante el desarrollo:

- Reestructurar el *pipeline* de percepción en módulos claramente separados y parametrizables, facilitando el ajuste de algoritmos sin necesidad de recompilación.
- Mejorar la robustez y eficiencia de las etapas del *pipeline*, garantizando la ejecución en tiempo real en condiciones de competición.
- Integrar herramientas de validación y depuración que faciliten el análisis del sistema tanto en RViz como a través de registros reproducibles (*rosbag*).
- El código debe estar estructurado y documentado de forma clara, facilitando su interpretación por nuevos integrantes del equipo. Este aspecto es esencial, ya que permite reducir el tiempo de adaptación y asegurar la continuidad del desarrollo en futuras temporadas.

De esta forma, los objetivos definidos pueden clasificarse en dos niveles complementarios: por un lado, los **requisitos funcionales**, que establecen de manera explícita las capacidades que el sistema debe ofrecer (qué debe ser capaz de hacer el sistema de percepción); y, por otro lado, los **requisitos no funcionales**, que especifican las condiciones de diseño, eficiencia y robustez necesarias para garantizar que dichas capacidades se alcanzan en un entorno de competición real.

Requisitos funcionales

- Detectar conos a una distancia mínima de 15 metros con una tasa de acierto superior al 90 %.
- Clasificar correctamente conos azules, amarillos y naranjas con una precisión superior al 80 % incluso en condiciones de iluminación variable.
- Reducir la tasa de falsos positivos derivados de ruido del sensor o estructuras del entorno (vegetación, bordes de pista) a menos de un 10 %.
- Publicar la lista de conos detectados en un mensaje `ConeArray`, incluyendo su posición tridimensional y el color estimado.

- Mantener la consistencia de las detecciones entre iteraciones consecutivas, evitando pérdidas o duplicaciones de conos.
- Aceptar y procesar de forma robusta cualquier nube de puntos de entrada, incluyendo casos límite como nubes vacías, nubes sin conos o con objetos atípicos, sin provocar fallos en la ejecución.

Requisitos no funcionales

- Soportar nubes de puntos de hasta $32 \times 2048 = 65\,536$ puntos por rotación (configuración máxima del sensor Ouster OS1-32) sin pérdida de funcionalidad ni errores.
- El tiempo de procesamiento del *pipeline* no debe superar los 100 ms por nube de puntos, garantizando operación en tiempo real con el sensor en su configuración de resolución máxima y 10 Hz de frecuencia de publicación.
- La arquitectura debe ser modular y parametrizable, permitiendo ajustar parámetros (e.g., `eps`, `min_samples`, umbrales geométricos) en ejecución sin necesidad de recompilación.
- El sistema debe integrar herramientas de validación y depuración, como visualización en RViz y análisis a partir de registros reproducibles mediante `rosviz`.
- El uso de recursos computacionales debe ser eficiente, evitando que el consumo de CPU bloquee otros nodos críticos del *stack* Driverless.
- La implementación debe ser portable y mantenible, con separación clara de nodos y documentación suficiente para permitir futuras modificaciones por parte del equipo.

4.4. Solución propuesta

4.4.1. Arquitectura

En el nuevo desarrollo se ha sustituido la cadena de tres nodos por una arquitectura centrada en una **única API** (`API_lidar`) que maneja directamente las suscripciones y publicaciones del sistema. Esta arquitectura puede observarse en la Figura 4.4.

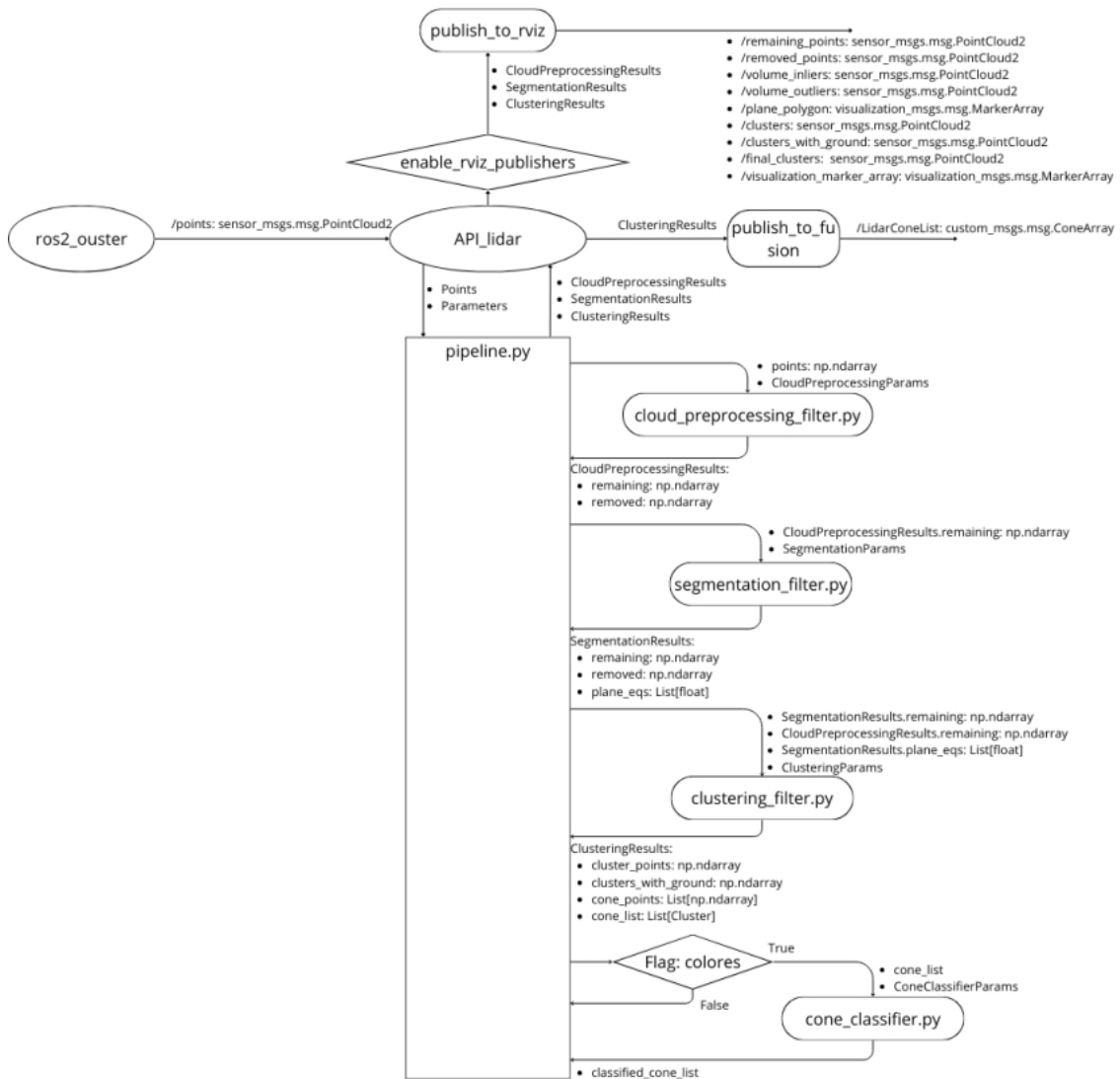


Figura 4.4: Arquitectura propuesta basada en una única API.

La principal motivación de este rediseño radica en los elevados costes computacionales asociados a la arquitectura anterior. Con tres nodos encadenados, el flujo de datos requería múltiples conversiones entre el formato estándar de ROS 2 (`PointCloud2`) y estructuras de `arrays` procesables en Python. Concretamente, el proceso era el siguiente:

- El primer nodo recibía el mensaje `/points` en formato `PointCloud2`, lo convertía a un `array` y aplicaba el recorte geométrico mediante `bounding box`.
- A continuación, debía volver a convertir los datos en `PointCloud2` para enviarlos al siguiente nodo.
- El segundo nodo repetía la conversión a `array` para aplicar RANSAC, y posteriormente transformaba los dos resultados, *inliers* y *outliers*, otra vez a `PointCloud2`.
- Finalmente, el nodo de *clustering* volvía a convertir la nube de *outliers* a `array`, realizaba el procesamiento correspondiente y publicaba de nuevo tanto la nube de puntos con los *clusters* en `PointCloud2` como la lista de conos detectados.

Estos ciclos repetidos de *serialización y deserialización* resultaban muy costosos computacionalmente. Si bien la existencia de varios nodos en paralelo permitía procesar distintas nubes a la vez y reducía el riesgo de perder mensajes del LiDAR, el tiempo total para transformar una nube en una lista de conos aumentaba de forma significativa. En condiciones reales de pista, esto se traducía en que los mensajes llegaban tarde: cuando la información de los conos alcanzaba al planificador, el vehículo ya podía haberse desplazado y los conos detectados ya no se encontraban en la misma posición.

Con la nueva arquitectura de una sola API se eliminan estas conversiones redundantes, concentrando todo el flujo de procesamiento dentro de un único nodo. Este enfoque aporta varias ventajas de gran relevancia práctica:

- Todos los parámetros del sistema se gestionan desde un único lugar, simplificando los ajustes durante pruebas y entrenamientos.
- El despliegue en competición resulta mucho más sencillo, pues basta con lanzar un solo nodo para poner en marcha el *pipeline* completo.

El único inconveniente de este diseño es que obliga a garantizar que el tiempo de procesamiento completo de una nube de puntos no supere los 100 ms, de manera que el sistema pueda operar en tiempo real con el sensor sin perder ninguno de sus mensajes.

Además, desde la propia API, mediante el parámetro `enable_rviz_publishers`, es posible habilitar o deshabilitar la publicación de nubes intermedias (`PointCloud2`) en cada etapa y una representación de los conos detectados, además de otros resultados de interés. Esto resulta muy útil para depuración y validación durante el desarrollo (por ejemplo, en RViz o mediante análisis de `rosviz`). Sin embargo, en pruebas en pista estos *publishers* pueden desactivarse con un único parámetro, evitando el envío de información innecesaria y reduciendo la sobrecarga de comunicaciones, de modo que únicamente se publique la lista final de conos detectados.

Esta arquitectura sienta las bases sobre las que se estructuran las distintas etapas del *pipeline* (filtrado inicial, segmentación, *clustering* y clasificación), que se detallan en las siguientes secciones.

4.4.2. Filtrado inicial de la nube de puntos

Este primer módulo tiene como objetivo preparar la nube de puntos para las etapas posteriores del *pipeline*, asegurando que la información procesada sea relevante y eficiente desde el punto de vista computacional. Se han implementado dos filtros principales:

- **Máscara de filtrado configurable:**

En esta etapa se aplica una máscara sobre la nube cruda recibida en `/points`. Su motivación principal no es únicamente reducir el volumen de datos para disminuir el tiempo de cómputo, sino evitar procesar zonas del espacio donde, por física del sensor, un cono distante queda representado por uno o dos puntos, lo que lo hace indetectable de forma fiable. De igual modo, se descartan también los retornos espurios producidos por haces que impactan sobre la propia carrocería del vehículo, que sólo introducen ruido en la nube.

Esta máscara incluye como caso base una *bounding box* sobre las coordenadas (x, y, z) , pero puede extenderse fácilmente con otros criterios, como:

- **Anillos del sensor (ring):** seleccionar únicamente ciertos canales inferiores.
- **Reflectividad:** imponer un umbral mínimo y descartar retornos débiles.
- **Rango:** limitar la zona de interés por la distancia de los puntos al sensor.

Los límites de la *bounding box* están parametrizados, lo que permite ajustar rápidamente la región de interés sin modificar el código. Además, el diseño es robusto frente a cambios de orientación del sensor. En la práctica, el LiDAR se suele montar con el cable hacia atrás o de lado, lo que implica que x e y pueden cambiar de sentido respecto a la referencia original, sin embargo, la máscara se adapta automáticamente a dicha situación.

■ Filtro de ruido adaptativo:

Tras aplicar la máscara, se dispone de un filtro de ruido de tipo ROR dependiente de la distancia (**noise_filter**) (Véase la Sección 3.3.1), cuyo objetivo es eliminar puntos aislados que carecen de densidad suficiente en su vecindad. Aunque está disponible, suele desactivarse en las configuraciones de pista porque introduce un coste temporal apreciable y los pocos puntos que elimina tienden a descartarse igualmente en etapas posteriores (especialmente en *clustering*, véase la sección 4.4.4). No obstante, resulta útil en sesiones de depuración o en análisis de *rosbags* con condiciones especialmente ruidosas.

Tras estos dos pasos iniciales (*máscara* y *ruido*), el diseño contempla la sustitución o concatenación de filtros adicionales según sea necesario (p. ej., *voxel downsampling* (Sección 3.3.2), recortes angulares, etc.), manteniendo la misma estructura: cada filtro recibe los **remaining** de la etapa anterior y devuelve el par (**remaining**, **removed**); los puntos descartados se van acumulando en **removed** para su trazabilidad y visualización. Esta composición, controlada por parámetros, permite activar o desactivar filtros sin modificar la lógica central y asegura coherencia con las etapas posteriores del *pipeline*.

En la Figura 4.5 se muestra un ejemplo del resultado de esta etapa. Para una mejor interpretación, en verde se representan los puntos que continúan en el *pipeline* y en rojo aquellos descartados, bien por encontrarse fuera de la región de interés o bien por corresponder a puntos aislados eliminados por el filtro de ruido.

En las pruebas realizadas, la máscara aplicada filtraba únicamente los puntos situados entre 1 y 35 metros en el eje longitudinal, dentro de un ancho de ± 30 metros en el eje lateral y con una altura máxima de 2.5 metros, lo que corresponde a la configuración `xmin=1`, `xmax=35`, `ymin=-30`, `ymax=30` y `zmax=2.5`.

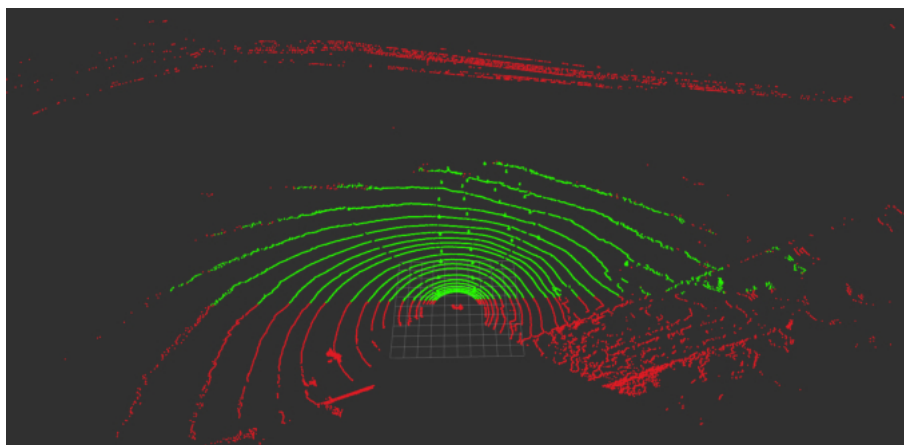


Figura 4.5: Resultado del filtrado inicial aplicado a la nube de puntos del LiDAR.

4.4.3. Segmentación de la nube de puntos

El objetivo general de este módulo es dividir la nube de puntos en dos conjuntos diferenciados:

1. puntos que pertenecen a estructuras planas dominantes, principalmente el suelo.
2. puntos que no pertenecen a estas estructuras, y que por tanto son los candidatos a incluir conos y demás obstáculos relevantes para la conducción autónoma.

En este contexto, donde los circuitos son normalmente planos, la única separación que se requiere es la de los puntos pertenecientes al suelo. Para realizar esta separación se aplica el algoritmo **RANSAC (Random Sample Consensus)** (Sección 3.3.3). Su funcionamiento consiste en:

1. Seleccionar de manera aleatoria un subconjunto de 3 puntos de la nube, cuyos vectores no sean colineales, de modo que la normal resultante no sea nula.
2. Ajustar un plano que pase por esos tres puntos.
3. Calcular cuántos puntos de la nube completa se encuentran a una distancia menor que un umbral (τ) respecto a dicho plano (considerados *inliers*).
4. Repetir el proceso un número fijo de iteraciones y conservar el plano que maximice el número de *inliers*.
5. Finalmente, se eliminan los *inliers* del plano dominante
6. Repetir tantas veces como planos dominantes se quieran eliminar. En este caso, con realizar este proceso una sola vez es suficiente.

La selección del número máximo de iteraciones de RANSAC plantea un compromiso relevante. Un número demasiado bajo puede ser insuficiente para encontrar un plano consistente, especialmente en nubes con ruido o con superficies incompletas, lo que provocaría que queden restos de suelo en la nube de obstáculos. Esta situación repercute directamente en la etapa de *clustering*, ya que la presencia de grandes superficies planas mal segmentadas tiende a ralentizar el algoritmo y a generar agrupaciones erróneas. Por el contrario, un número excesivo de iteraciones aumenta innecesariamente el tiempo de cómputo en la propia segmentación, comprometiendo el funcionamiento en tiempo real.

Este parámetro puede fundamentarse en términos probabilísticos. El objetivo es garantizar que, tras un número N de iteraciones, se haya seleccionado al menos un subconjunto de puntos formado exclusivamente por *inliers*, lo que permitirá ajustar un plano fiable.

Sea p la probabilidad deseada de éxito (p. ej. $p = 0,99$), w la proporción estimada de *inliers* en la nube de puntos y s el número mínimo de puntos necesarios para definir el modelo (en el caso de un plano, $s = 3$). La probabilidad de que una iteración escoja un subconjunto válido es w^s , y por tanto, la probabilidad de fracasar en una iteración es $1 - w^s$. Si se realizan N iteraciones independientes, la probabilidad de fracasar en todas ellas es $(1 - w^s)^N$.

Imponiendo que esta probabilidad de fracaso sea inferior a $1 - p$, se obtiene la condición:

$$(1 - w^s)^N \leq 1 - p,$$

lo que implica:

$$N \geq \frac{\log(1 - p)}{\log(1 - w^s)}.$$

Esta expresión permite estimar el número de iteraciones necesario para alcanzar una probabilidad de éxito p en función de la calidad de los datos (w) y de la complejidad del modelo (s). En la práctica, este análisis proporciona una referencia útil para dimensionar el parámetro N , evitando, en la medida de lo posible, tanto un ajuste insuficiente como un sobre coste computacional innecesario, aunque plantea el problema de determinar w .

De forma análoga, la selección del umbral de distancia (τ) al plano también supone un compromiso importante: si se fija un valor demasiado alto, existe el riesgo de eliminar puntos válidos de conos cercanos al suelo, lo que dificultaría el proceso de *clustering*; si, por el contrario, se escoge un valor demasiado bajo, permanecerán restos del suelo que incrementan el ruido para la siguiente etapa.

Para mitigar este efecto, se emplea un valor relativamente amplio; aunque esto introduce el riesgo eliminar la base de los conos, esta información residual se recupera y depura más adelante en la etapa de *clustering*.

En este caso, el filtro también elimina los puntos a una altura parametrizada con respecto al plano, ya que conservar los puntos a una altura mayor que la altura de un cono puede no ser útil.

Por otra parte, los puntos que aparecen por debajo del plano del suelo como consecuencia de reflexiones múltiples (*multipath reflections*) también son eliminados. Este fenómeno ocurre, como se muestra en la Figura 4.6, cuando un haz láser incide sobre una superficie reflectante y rebota antes de volver al sensor, generando *puntos fantasma* que no corresponden a objetos reales en la escena.

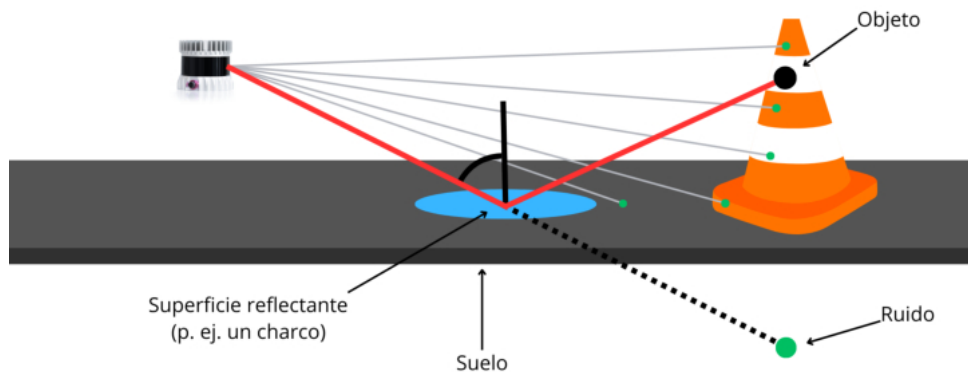


Figura 4.6: Ejemplo de puntos fantasma (*ghost points*) bajo el suelo debido a reflexiones múltiples en una superficie reflectante.

De forma análoga a lo implementado en el módulo de *cloud preprocessing*, este bloque ha sido diseñado para que sea sencillo sustituir el filtro de segmentación por otro método alternativo, siempre que se respete la estructura de salida diferenciando entre **removed** (puntos eliminados) y **remaining** (puntos que continúan en el flujo).

Aunque se exploró como alternativa el método *Region Growing Segmentation*, las nubes de puntos que devuelve el sensor no son lo suficientemente homogéneas en términos de densidad. Los huecos entre puntos o anillos hacen que este no sea un método adecuado y los resultados obtenidos fueron considerablemente peores en términos de eficiencia y preservación de conos. De este modo, se confirmó que el algoritmo **RANSAC**, ya empleado en el sistema original, seguía siendo la mejor opción para este proyecto por su robustez, precisión y viabilidad en tiempo real. Asimismo, la ecuación del plano que devuelve este método es de mucha utilidad en la siguiente etapa del *pipeline*.

Además, aprovechando el plano que devuelve RANSAC, se incluye una opción de **autonivelado**

(*autolevel*) que rota la nube de puntos de forma que el plano detectado se alinee con el eje horizontal del sistema de referencia. El procedimiento seguido es el siguiente:

1. **Cálculo de la orientación:** con la ecuación general del plano dominante ($Ax + By + Cz + D = 0$) se normaliza $\vec{n} = (A, B, C)$ y se compara con el eje vertical del sistema ($\hat{z} = (0, 0, 1)$) para determinar el ángulo de inclinación $\theta = \arccos(\vec{n} \cdot \hat{z})$.
2. **Determinación del eje de rotación:** se obtiene como el producto vectorial $\vec{r} = \vec{n} \times \hat{z}$, que indica la dirección alrededor de la cual hay que rotar la nube.
 - **Convención de normal:** si $\vec{n} \cdot \hat{z} < 0$, se invierte la normal, $\vec{n} \leftarrow -\vec{n}$, para evitar rotaciones cercanas a π radianes.
 - **Casos degenerados y estabilidad:** si $\theta \approx 0$, no se aplica rotación. Si $\theta \approx \pi$ o $\|\vec{r}\| < \varepsilon$, se elige un eje fijo ortogonal a $\hat{z}$ (p. ej., $(1, 0, 0)$) para una rotación de π radianes.
3. **Construcción de la matriz de rotación:** para alinear el plano con el eje horizontal se construye una matriz de rotación a partir del eje de giro $\vec{r}$ y el ángulo θ . Para ello se emplea la **rotación de Rodrigues** [46], que genera la matriz de rotación mediante

$$R = I + \sin \theta [\vec{r}]_{\times} + (1 - \cos \theta) [\vec{r}]_{\times}^2,$$

donde I es la matriz identidad, θ es el ángulo entre el vector normal del plano y el eje vertical, y $[\vec{r}]_{\times}$ es la matriz de producto cruzado asociada al eje de rotación $\vec{r} = (r_x, r_y, r_z)$:

$$[\vec{r}]_{\times} = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}.$$

4. **Aplicación a la nube:** una vez construida la matriz de rotación R , se aplica a toda la nube de puntos de manera vectorizada. Si la nube original se representa como una matriz

$$P = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix},$$

en la que cada fila corresponde a un punto de la nube. La nube corregida se obtiene mediante

$$P' = P \cdot R^{\top},$$

donde P' es la nube transformada y R la matriz de rotación calculada con la fórmula de Rodrigues. De este modo, todos los puntos se transforman simultáneamente, garantizando que el plano estimado por RANSAC quede alineado con el plano XY y que los conos u obstáculos conserven su geometría en coordenadas coherentes.

En la Figura 4.7 se aprecia la nube original (rojo) y el resultado de aplicar la operación de **autonivelado** (verde), donde la nube de puntos queda alineada con el plano horizontal, facilitando así que los conos u obstáculos aparezcan verticales y en coordenadas consistentes con el sistema de referencia del vehículo.

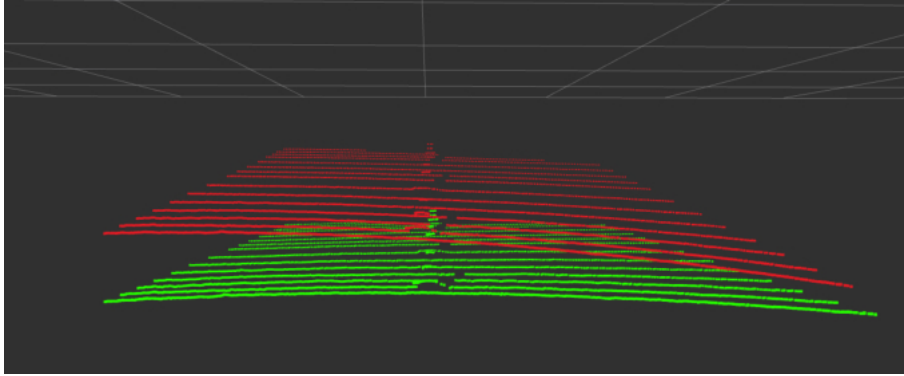


Figura 4.7: Resultado del autonivelado en la nube de puntos.

No obstante, se deben tener en cuenta los siguientes factores:

- La inclinación del sensor en el vehículo suele ser despreciable, por lo que el beneficio es limitado.
- Existe el riesgo de que, si el algoritmo selecciona como plano dominante una superficie incorrecta (p. ej. una pared), se produzca una rotación errónea que afecte a todas las coordenadas de la nube, comprometiendo no solo a etapas posteriores, sino también a los sistemas SLAM y *Path Planning* del *stack*.

Por ello, aunque se implementó y validó, se mantiene desactivado por defecto en el uso competitivo.

En la Figura 4.8 se muestra un ejemplo del resultado de la segmentación. Los puntos clasificados como *inliers* del plano del suelo se representan en rojo, mientras que los *outliers*, que incluyen los posibles conos y obstáculos relevantes, se muestran en verde. De este modo se aprecia visualmente cómo el módulo elimina gran parte del suelo y conserva los elementos de interés para las etapas posteriores del *pipeline*.

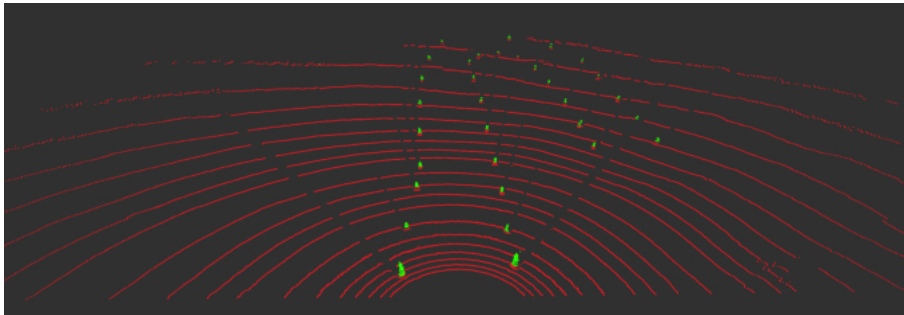


Figura 4.8: Resultado de RANSAC aplicado a la nube de puntos.

Con ello, el módulo de segmentación entrega al **pipeline** una nube depurada del suelo, lista para ser procesada por la etapa de *clustering*, que se encarga de reagrupar los puntos restantes (*outliers*) para su posterior clasificación.

4.4.4. *Clustering* en la nube de puntos

Tras la etapa de segmentación, se obtiene una nube libre de suelo en la que, *a priori*, sólo permanecen puntos pertenecientes a conos, otros obstáculos y estructuras adicionales (vallas, bordes

de pista, etc.). El objetivo global de este módulo es estimar la posición de los conos a partir de los puntos restantes.

En cualquier caso, para este módulo no existe una solución perfecta, pues cualquier decisión de diseño supone un compromiso entre **sensibilidad** (detectar todos los conos reales) y **precisión** (evitar falsos positivos procedentes de ruido u otras estructuras).

Para ello, el código se organiza en cuatro etapas:

1. **Agrupar** los puntos de la nube en función de su pertenencia a objetos individuales.
2. **Filtrar rápidamente** los grupos que definitivamente no son conos.
3. **Recuperar** (si procede) los puntos del objeto que fueron eliminados en la segmentación previa (base del cono u otros puntos cercanos al plano).
4. **Validar y seleccionar** los *clusters* que realmente son conos, obteniendo su posición final.

A continuación se describe en detalle cada una de estas etapas y las decisiones de diseño adoptadas en su implementación.

1) Agrupamiento de puntos en objetos

Para esta etapa se ha mantenido el algoritmo **DBSCAN** (*Density-Based Spatial Clustering of Applications with Noise*, Sección 3.4.3), utilizando únicamente las coordenadas x e y para reducir dimensionalidad, y por lo tanto, coste. Su funcionamiento se basa en identificar regiones densas de puntos y agruparlos en función de su proximidad dentro de un radio ε .

A diferencia del método clásico *Euclidean Cluster Extraction*, DBSCAN requiere especificar el número mínimo de puntos (m) en cada *cluster*, por lo que es capaz de gestionar el ruido de forma implícita, descartando los *clusters* que no presentan densidad suficiente en su entorno. De forma conceptual, su funcionamiento consiste en:

- Un punto p es *núcleo* si tiene al menos m vecinos a distancia $< \varepsilon$.
- Un punto es *frontera* si no es núcleo, pero está a distancia $< \varepsilon$ de algún núcleo.
- Los puntos no alcanzables desde núcleos se etiquetan como *ruido*.

En este caso se ha configurado DBSCAN con `min_samples = 2`. La elección de este valor se justifica porque, al utilizar un umbral amplio para eliminar el suelo en la etapa de segmentación, pueden perderse algunos puntos que en realidad pertenecen a conos. Este efecto es especialmente crítico en los conos más lejanos, que de por sí tienden a estar representados por un número reducido de muestras. Establecer este umbral permite que dichos conos sigan siendo detectados, pero al mismo tiempo evita clasificar como *cluster* ruido residual, que acostumbra a aparecer en forma de puntos aislados. De esta manera, se consigue un efecto similar al filtro de ruido mencionado en la Sección 4.4.2, sin necesidad de aplicar una etapa adicional de preprocesado.

En cuanto al parámetro ε (**radio de vecindad**), como ya se ha mencionado, la separación entre impactos crece proporcionalmente a la distancia al sensor. Con el fin de garantizar que los objetos se agrupen de forma coherente incluso en rangos largos, se ha fijado un valor de $\varepsilon = 0,5$. Este umbral permite unir los puntos de conos lejanos, que tienden a estar representados por un número reducido de muestras, y al mismo tiempo asegura que estructuras alargadas como vallas o bordes de pista no se fragmenten en múltiples *clusters* cuando presentan huecos en su superficie.

Únicamente se da un caso en el que dos conos reales aparecen tan próximos como para formar un mismo *cluster* bajo este umbral, situación para la que se ha implementado un tratamiento específico que se detalla en el siguiente apartado.

En la Figura 4.9 se muestra un ejemplo visual del resultado de esta etapa. En la imagen superior puede observarse la nube de puntos tras la segmentación, en la que los conos aparecen únicamente como conjuntos dispersos de impactos. En la inferior se aprecian los *clusters*, diferenciados por colores, obtenidos mediante DBSCAN, donde los puntos pertenecientes a un mismo objeto han sido agrupados y etiquetados en base a su proximidad. Este resultado constituye la base para las etapas posteriores de filtrado y clasificación, en las que se decidirá qué grupos corresponden realmente a conos válidos.

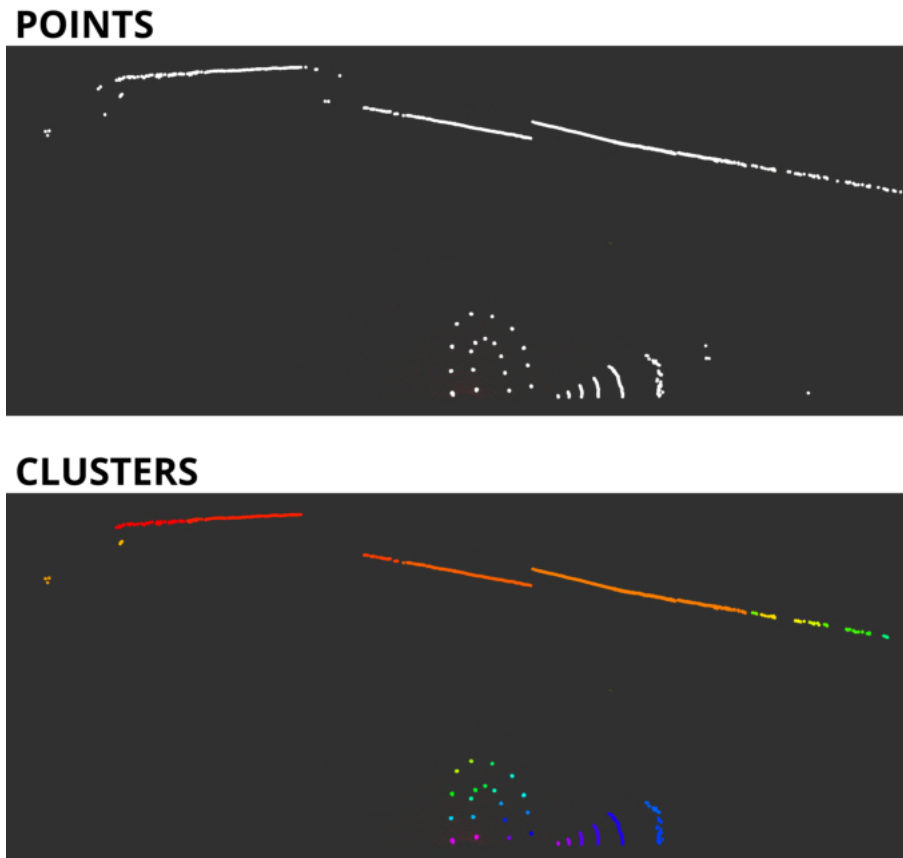


Figura 4.9: Resultado de la agrupación: nube original tras la segmentación (arriba) y agrupamiento de puntos en *clusters* (abajo).

2) Filtro preliminar (descartar lo que no puede ser un cono)

Una vez obtenidos los *clusters* mediante DBSCAN, se aplica un filtrado rápido destinado a eliminar aquellos grupos de puntos que, por sus características geométricas o de distribución en los anillos del LiDAR, no pueden corresponder a conos reales.

En este filtro se analizan métricas como:

- Altura local del *cluster*
- Distancia del *cluster* al suelo
- Diámetro del *cluster*
- Número de anillos de los que provienen sus puntos

- Cantidad de puntos por anillo

De este modo se descartan estructuras extensas como vallas o bordes de pista, así como trozos de suelo que no se hayan filtrado en la etapa de segmentación, reduciendo de forma significativa el número de candidatos a procesar en etapas posteriores.

Dentro de este mismo filtrado se resuelve además la única casuística en la que el valor $\varepsilon = 0,5$ puede provocar la agrupación de varios conos reales en un único *cluster*. Este caso se produce en las pruebas *Skidpad*, *Autocross* y *Trackdrive*, donde se colocan bien dos conos naranjas muy cercanos entre sí, o bien dos conos naranjas separados por un tercer cono (amarillo o azul, dependiendo del lado).

Para evitar que dichos conos se interpreten como un único obstáculo, y por lo tanto sean eliminados al ser su tamaño demasiado grande, se implementó una lógica en el filtro en la que, si el diámetro de un *cluster* se encuentra entre dos valores parametrizados (mayor que el de un cono pero menor que el de un grupo reducido de conos), se le da una **segunda oportunidad**.

Esta segunda oportunidad consiste en una ejecución adicional de DBSCAN, esta vez con un ε más restrictivo, con el objetivo de comprobar si el *cluster* inicial está en realidad compuesto por varios conos individuales.

Si el resultado de esta segunda pasada arroja más de un *subcluster* válido que cumpla las métricas geométricas definidas, se sustituyen en la nube de salida, garantizando que los objetos de salida queden representados de manera correcta y no se clasifiquen de forma errónea.

En la siguiente figura se muestra el resultado de esta “segunda oportunidad”, en la que los conos naranjas más cercanos al sensor, separados por una distancia suficientemente pequeña para ser inicialmente agrupados en un mismo *cluster*, son finalmente separados en dos detecciones individuales.

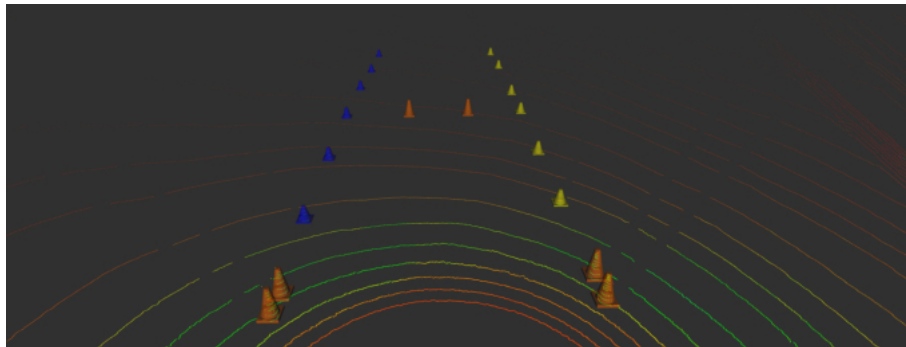


Figura 4.10: Ejemplo de separación de *cluster* con dos conos cercanos.

3) Recuperación de puntos eliminados en segmentación

Con un umbral τ de RANSAC deliberadamente *amplio* (Sección 4.4.3) puede ocurrir que la base de un cono quede parcialmente absorbida como *inlier* del plano. Para restaurar la forma completa, para cada *cluster* se realizan las siguientes operaciones:

- a) **Localización del objeto:** se calcula un centro en planta $c = (\bar{x}, \bar{y})$ del *cluster* candidato.
- b) **Búsqueda en la nube original:** se inspecciona la nube *previa a segmentación* en un vecindario cilíndrico de radio $R = 0,15 \text{ m}$ alrededor de c .

- c) **Consistencia con el plano:** de esos puntos vecinos, sólo se *reincorporan* los que están a una distancia *ligeramente* negativa respecto al plano ($-h_{\text{máx}} \leq d_{\pi}(p) < 0$), o dentro de una *banda de tolerancia* alrededor del plano ($|d_{\pi}(p)| \leq \delta$), para recuperar la base sin incluir *puntos fantasma* (Figura 4.6).

El resultado de este proceso es el *cluster enriquecido* $C^* = C \cup \{\text{puntos recuperados coherentes}\}$, que preserva una geometría del objeto más acertada.

En la Figura 4.11 se muestra un ejemplo del efecto de este procedimiento. En la parte superior aparecen los *clusters* obtenidos tras el filtrado inicial, donde la base de algunos conos estaba recortada. En la parte inferior se observan los mismos *clusters* tras la recuperación del suelo, que permite restaurar la geometría completa de los objetos sin introducir ruido adicional.

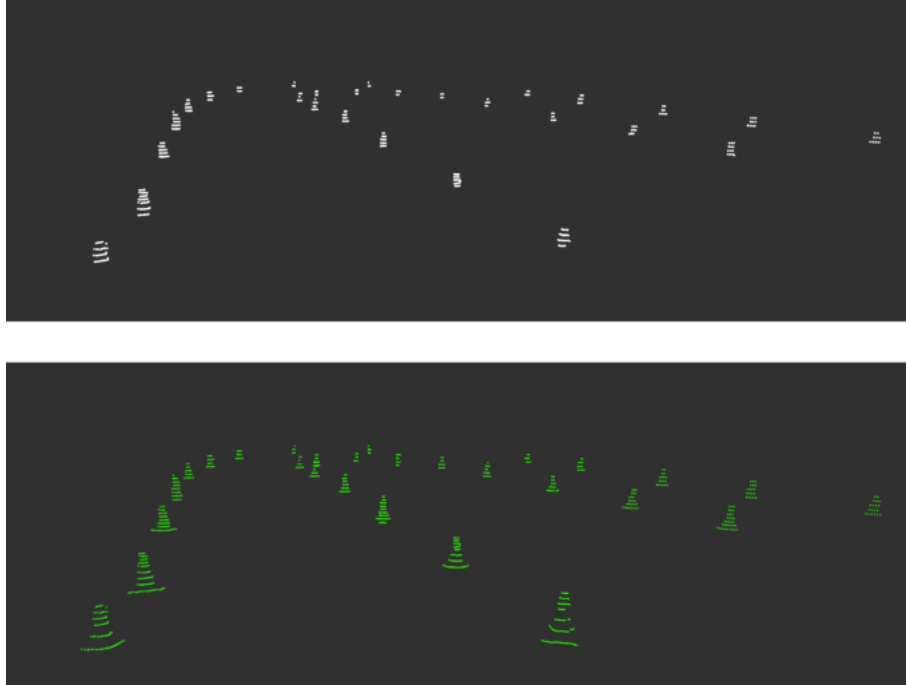


Figura 4.11: Recuperación de puntos eliminados en segmentación: *clusters* originales tras RANSAC (arriba) y *clusters* enriquecidos con la base recuperada (abajo).

4) Validación fina y selección de conos

Con el conjunto de puntos completo (C^*) se lleva a cabo la **validación fina**, en la que se decide definitivamente si los objetos se tratan de conos. Para ello se aplican las siguientes reglas:

- **Número mínimo de anillos:** el *cluster* debe tener puntos de al menos tres anillos distintos, de lo contrario se descarta.
- **Distribución por anillos:** se descartan *clusters* donde todos sus anillos están representados por un único punto o contienen huecos verticales anómalos.
- **Altura máxima:** la altura del *cluster* $h = \text{máx}(z) - \text{mín}(z)$ no debe superar 0,55 m.
- **Diámetro máximo:** el diámetro en planta $\sqrt{(\text{máx}(x) - \text{mín}(x))^2 + (\text{máx}(y) - \text{mín}(y))^2}$ no debe superar 0,43 m.

- **Orientación:** se rechazan los *clusters* cuya geometría corresponde a un cono invertido (suelen ser trozos de pared).
- **Compatibilidad con el suelo:** utilizando el plano de RANSAC, se elimina cualquier *cluster* si no contiene ningún punto a menos de 0,20 m del suelo.

Aquellos *clusters* que superan todas las condiciones anteriores se consideran conos válidos y para cada uno de ellos se realiza lo siguiente:

- Se calcula un punto representativo del cono a partir de sus coordenadas: las componentes x e y se obtienen como la media entre los puntos de los dos anillos superiores (robustez ante ruido), mientras que la componente z se toma únicamente del anillo inferior.
- Con esta información se inicializa un objeto **Cluster** de tipo “Cone”. De forma predeterminada, todos los conos se crean con color **Unknown** y un identificador de los puntos que lo componen.
- A continuación se aplica una regla heurística para identificar los conos naranjas: si la diferencia de altura ($\text{máx}(z) - \text{mín}(z)$) supera un umbral de 0.38 m, el *cluster* se clasifica directamente como **Orange**.
- En caso contrario, se mantienen asociados los puntos originales para que, si procede, puedan ser clasificados en la etapa posterior.

El resultado de esta fase es una lista **ConeArray** con todos los conos válidos detectados, incluyendo aquellos ya identificados como naranjas y aquellos pendientes de su clasificación en el siguiente módulo.

Salidas del módulo El bloque expone tres salidas diferenciadas y coherentes con el resto de la *pipeline*:

- **initial_clusters_points:** nubes por *cluster* tras DBSCAN (antes del primer filtrado).
- **clusters_with_ground:** nubes por *cluster* tras recuperar la base.
- **final_clusters_points:** nubes por *cluster* tras el filtrado geométrico (sólo candidatos válidos).
- **cone_list:** lista de conos confirmados, cada uno con su posición (x_c, y_c, z_c) y su *etiqueta preliminar de color*.

Con este diseño, el módulo transforma de forma fiable y en tiempo real la nube sin suelo en una lista de conos localizados, lista para el postprocesado y la clasificación de color. La Figura 4.12 muestra un ejemplo del resultado: todos los objetos detectados corresponden a conos reales; los representados en naranja son aquellos cuya clase de color ya ha sido identificada en esta etapa, mientras que los de color gris permanecen como *Unknown* y serán clasificados en la siguiente fase del *pipeline*.

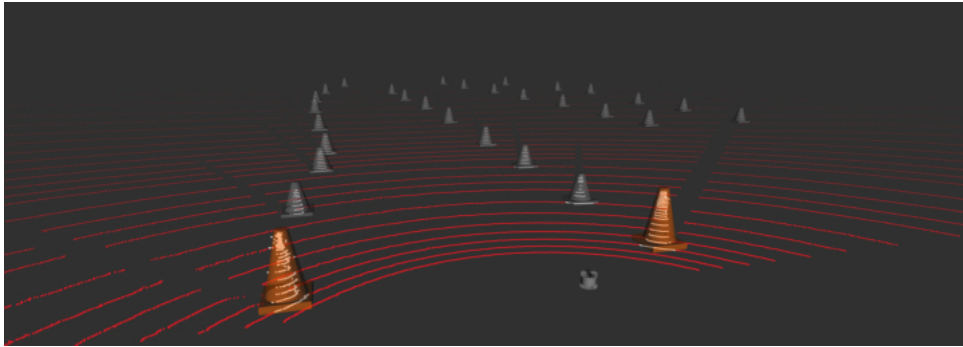


Figura 4.12: Resultado de la etapa de *clustering*: conos detectados en la nube de puntos.

4.4.5. Clasificación de color basada en datos de reflectividad

Una vez obtenida la lista de conos final, el último paso del *pipeline* es determinar su color. Como ya se ha mencionado, este aspecto resulta esencial para el módulo de *Path Planning*, que utiliza los conos azules para delimitar el lado izquierdo del circuito y los amarillos para el lado derecho. Cruzar entre conos del mismo color supondría, en la práctica, salir de la pista.

Como se observa en la Figura 2.1, los conos amarillos y azules de *Formula Student* no pueden diferenciarse de manera fiable por tamaño ni geometría. La única diferencia reside en su color característico y en la franja central, que es negra en los conos amarillos y blanca en los conos azules. Esta propiedad visual puede aprovecharse en LiDAR gracias al campo de reflectividad de los puntos, ya que los objetos oscuros tienden a reflejar menos energía láser que los claros, generando valores menores de retorno.

Tal y como se indicó en la Sección 4.3.1, en el sistema inicial la distinción entre conos azules y amarillos se realizaba mediante un enfoque heurístico: se analizaba la reflectividad media en la zona central del cono, comparándola con el resto del objeto. Si esa zona aparecía más oscura, se asumía la presencia de una banda negra, y por tanto el cono se clasificaba como amarillo, o azul en caso contrario.

Enfoque basado en aprendizaje automático

Con el fin de mejorar esta clasificación, se ha optado por un enfoque apoyado en aprendizaje automático (*Machine Learning*). Esta aproximación presenta varias ventajas frente a las reglas heurísticas:

- La detección de bandas oscuras se realiza a partir de los propios valores de reflectividad, no únicamente de la geometría.
- El modelo proporciona una salida probabilística, lo que permite definir un umbral de confianza ajustable.
- Es posible generalizar mejor a distintas distancias o condiciones de adquisición.

El principal reto de este enfoque es que el número de puntos que forman un cono varía con la distancia al sensor: los conos cercanos contienen cientos de puntos, mientras que los lejanos apenas unos pocos. Un clasificador supervisado requiere vectores de entrada de longitud constante, por lo que fue necesario definir un procedimiento de interpolación.

Construcción de una malla de reflectividad

La solución adoptada fue transformar los puntos de cada cono en una **mallla regular** de reflectividad de tamaño fijo 5×10 celdas. De esta manera, todos los conos, independientemente de su distancia al sensor y del número de puntos que los compongan, quedan representados mediante un vector de 50 valores, garantizando así una entrada de longitud constante para el clasificador supervisado.

En una primera aproximación, se intentó utilizar las coordenadas (y, z) para generar una representación bidimensional de la distribución de reflectividades de cada cono. Sin embargo, este planteamiento presentaba un inconveniente importante: el LiDAR únicamente capta la cara del cono que se encuentra orientada hacia el sensor. De este modo, cuando un cono es visto lateralmente, los puntos no describen un perfil cónico completo, sino más bien una “vista lateral” de su pendiente. Este efecto se ilustra en la Figura 4.13, donde se aprecia la falta de simetría en la malla obtenida a partir de esta proyección.

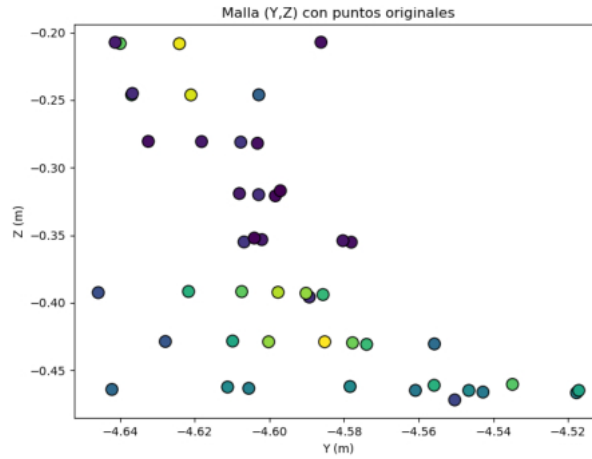


Figura 4.13: Ejemplo de malla obtenida con la proyección (y, z) . Se observa cómo la forma queda deformada cuando el cono no es visto de frente.

Para evitar este problema, se adoptó finalmente una proyección basada en el ángulo horizontal y la altura z . De esta forma, cada punto del cono se representa en un plano (θ, z) que mantiene una distribución más coherente con la geometría del objeto, independientemente de la orientación desde la que sea observado.

Estos valores se normalizan y se interpolan en una malla fija ajustada al tamaño exacto de cada *cluster*, de manera que no queden celdas vacías ni puntos fuera de rango. En la práctica, se empleó el método *nearest* de la librería `scipy.interpolate.griddata` [47], que asigna a cada celda el valor del punto más próximo. Esta elección evita la aparición de valores nulos (`NaN`), un inconveniente habitual en otros métodos de interpolación. El resultado es una matriz de reflectividades que posteriormente se aplanan para formar un vector de entrada de longitud fija adecuado para el clasificador.

En la Figura 4.14 se muestran ejemplos de estas mallas finales: a la izquierda un cono azul y a la derecha un cono amarillo. Se aprecia claramente la diferencia de reflectividades entre ambos casos, información que el clasificador puede explotar de manera fiable.

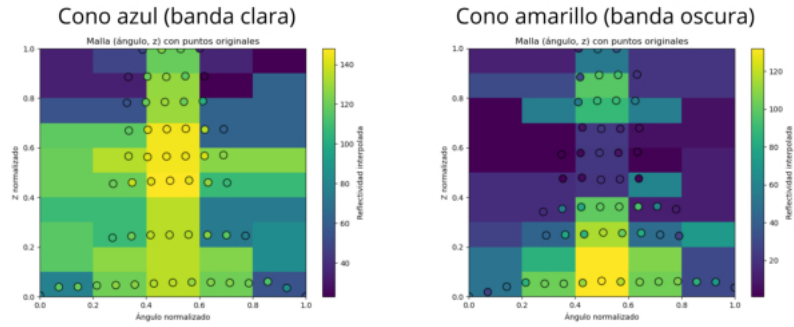


Figura 4.14: Ejemplos de mallas de reflectividad: cono azul (izquierda) y cono amarillo (derecha).

Entrenamiento del clasificador

Para generar el conjunto de entrenamiento se preparó un escenario con múltiples conos. En una primera fase se colocaron únicamente conos azules y se registraron sus vectores de reflectividad, etiquetándolos de manera manual como “Blue”. Posteriormente se repitió el proceso con conos amarillos (“Yellow”). De este modo se construyó un *dataset* con más de 15 000 muestras.

Tras evaluar distintos algoritmos comparados en la Sección 5.4.1, el modelo que mejor rendimiento ofreció fue un *Extreme Gradient Boosting Classifier* (XGBoost) configurado con los siguientes hiperparámetros:

- `n_estimators = 24`
- `colsample_bytree = 1.0`
- `learning_rate = 0.1`
- `max_depth = 5`
- `subsample = 1.0`

Esta elección proporciona una buena capacidad de generalización y un tiempo de inferencia reducido, compatible con la ejecución en tiempo real.

Es importante destacar que, dado que el vector de entrada depende directamente del procedimiento descrito para la construcción de la malla, cualquier modificación en este proceso puede alterar de forma significativa la distribución de los datos. Por ello, resulta altamente recomendable volver a grabar un nuevo *dataset* y reentrenar el modelo cada vez que se introduzcan cambios en la generación de la malla.

Inferencia en tiempo real

Durante la ejecución del sistema, para cada cono de la lista generada en el módulo anterior con color inicial **Unknown**, se genera su vector de reflectividad mediante el procedimiento descrito. Una vez obtenidos todos los vectores, estos se pasan al modelo entrenado, que devuelve probabilidades de pertenencia a cada clase. El sistema emplea un umbral de confianza de 0,75. Si la probabilidad de tratarse de un cono azul o amarillo supera ese valor, el color del cono se actualiza en consecuencia. En caso contrario, el cono permanece etiquetado como **Unknown**.

Dejar el color de un cono en estado **Unknown** resulta conveniente, ya que el módulo de SLAM mantiene un contador de observaciones para cada cono y consolida su color en función de las clasificaciones acumuladas. Forzar una decisión poco fiable introduciría ruido en ese contador,

mientras que mantener la etiqueta desconocida permite preservar la consistencia hasta disponer de observaciones más seguras.

Resultado final

Con esta etapa concluye el procesamiento de la nube de puntos. El resultado final es una lista de conos con posición y color que se publica para su visualización y para el resto de módulos del *stack driverless*, constituyendo la base sobre la que se construyen el mapeo, la localización y la planificación de trayectoria. Para ello, únicamente se publican los conos que presentan al menos dos vecinos en su entorno (5 m), ya que la detección de conos aislados no aporta información útil para el cálculo de trayectorias y, aunque podría contribuir a tareas de localización para el sistema de SLAM, introduce un elevado riesgo de falsos positivos que se evita con este criterio.

En la Figura 4.15 se muestra una comparación entre la nube de puntos original, sin procesar, y el resultado tras aplicar todo el *pipeline* y clasificar el color de los conos.

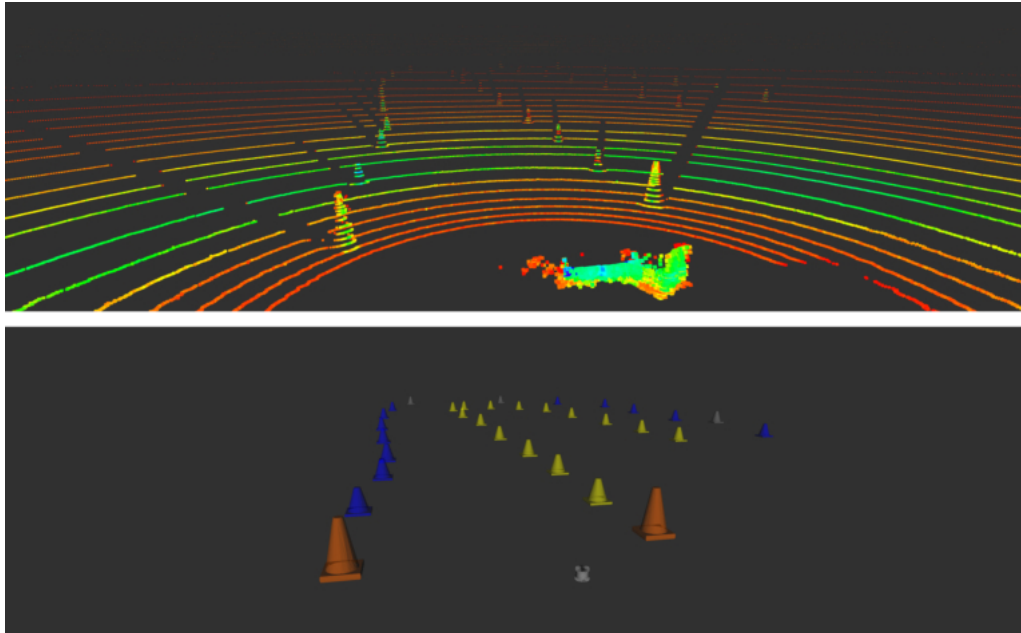


Figura 4.15: Comparación de la nube de puntos original y después del procesamiento completo.

A continuación, en el siguiente capítulo, se presentan los resultados experimentales obtenidos al evaluar cada una de las etapas del *pipeline* en distintos escenarios de prueba. Este análisis permitirá comprobar de forma objetiva el rendimiento del sistema en términos de tiempo de cómputo, precisión geométrica en la localización de conos, fiabilidad en la clasificación de color y robustez frente a ruido o estructuras no deseadas, así como valorar el grado de cumplimiento de los objetivos planteados en la Sección 1.2.

Capítulo 5

Resultados

En este capítulo se muestran los resultados obtenidos tras validar el sistema de percepción en entornos reales de prueba. Se presentan las métricas registradas durante la ejecución del *pipeline*, incluyendo los tiempos de procesamiento de cada etapa, la evolución en el número de puntos a lo largo del filtrado, la capacidad de detección de conos y la clasificación de su color.

Cada apartado recoge de forma estructurada los resultados más relevantes, apoyados en gráficas y tablas que permiten analizar el comportamiento del sistema bajo condiciones cercanas a las de competición.

5.1. Metodología de pruebas

Las pruebas se llevaron a cabo en la pista de ensayos del INSIA (Figura 5.1), un entorno amplio y relativamente plano que reproduce condiciones muy similares a las que se dan en competición. En este escenario se dispusieron diferentes configuraciones de conos, con el objetivo de validar el sistema de percepción en situaciones controladas pero realistas.



Figura 5.1: Pista de ensayos del INSIA.

Para la adquisición de datos se empleó un carro de herramientas adaptado por la división *Driverless* del equipo, que permite montar el sensor LiDAR a una altura equivalente a la de la *mounting plate* del ADS-DV utilizado en la *Formula Student* de Reino Unido, en la que los equipos montan sus sensores y demás dispositivos. La Figura 5.2 muestra el sensor LiDAR y la IMU montados en este carro.



Figura 5.2: Carro de pruebas adaptado para la instalación de los sensores.

Por otra parte, el procesamiento de las nubes de puntos se ha realizado en un ordenador portátil *Asus TUF Gaming*, equipado con un procesador AMD Ryzen 7 3750H de cuatro núcleos a 2.3 GHz y 16 GB de memoria RAM, ejecutando Ubuntu 22.04 LTS y ROS 2 Humble. Este equipo de cómputo de gama media es suficiente para validar el sistema en fase de desarrollo.

No obstante, en competición se emplea un ordenador embarcado In-CarPC CQ89G, equipado con un procesador Intel Core i9-10900TE (10 núcleos, 20 hilos), una GPU NVIDIA Quadro RTX 3000 (1920 núcleos CUDA y 240 núcleos Tensor) y 64 GB de RAM DDR4, diseñado específicamente para aplicaciones en automoción. Este *hardware* proporciona una capacidad de cómputo muy superior y asegura que el sistema pueda ejecutar en paralelo todos los módulos del *stack* en tiempo real.

La obtención de métricas en este tipo de sistemas resulta especialmente compleja. En un circuito real, con el sensor en movimiento, sería necesario mapear en cada *frame* la posición exacta y el color de todos los conos visibles para poder establecer una comparación directa con la salida del sistema. Con la volumetría de datos que se manejan en este proyecto, este procedimiento resulta prácticamente inviable por la enorme carga de trabajo que supondría, además de la dificultad para garantizar precisión y coherencia en la anotación manual.

Para simplificar el proceso de validación y mantener control experimental, se diseñaron dos tipos de pruebas.

5.1.1. Pruebas cualitativas

En una primera fase se realizaron ensayos orientados a comprobar el funcionamiento general del sistema:

- Visualización en RViz de las distintas etapas del *pipeline* (puntos prefiltrados, segmentación del suelo, *clusters*, conos clasificados).
- Verificación de la estabilidad del clasificador de color frente a variaciones de distancia y ángulo de incidencia.
- Evaluación de la robustez en tiempo real, comprobando que la latencia se mantenía por debajo del ciclo de publicación del sensor (10 Hz).

Estas pruebas, aunque no cuantificables en métricas numéricas, fueron esenciales para detectar errores de implementación, ajustar parámetros y garantizar que el sistema era operativo en condiciones cercanas a las de competición.

5.1.2. Pruebas cuantitativas

Para disponer de métricas objetivas, se diseñaron escenarios controlados con un número conocido de conos de un único color colocados en una zona de la pista concreta. Este planteamiento evita la necesidad, impracticable en un circuito completo, de mapear la posición exacta de cada cono en todos los *frames*. En la siguiente figura se puede observar una de las pruebas, donde únicamente había 17 conos amarillos en la pista.

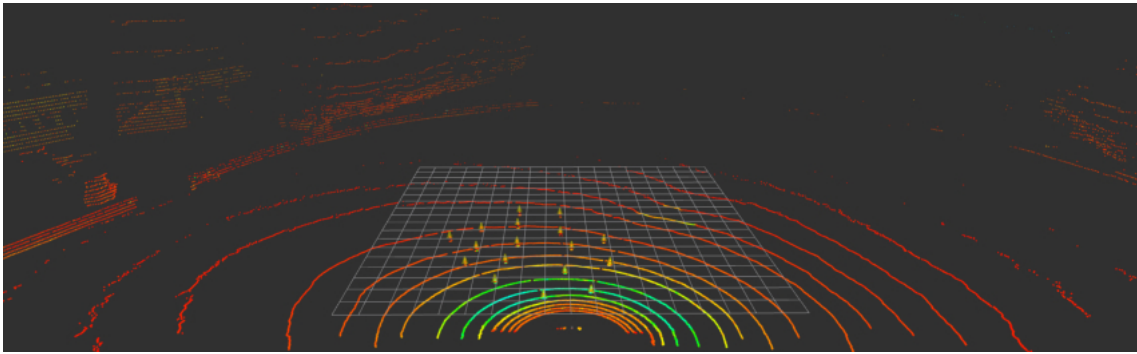


Figura 5.3: Escenario para extraer datos de los resultados del sistema.

En estas pruebas, se aseguró que no existiesen falsos positivos dentro de la región de conos reales (mostrada en la Figura 5.3 como una malla de 18 m de lado, desplazada 10 m en el eje x del sensor) y que los conos reales no saliesen de esa región en ninguno de los *frames*.

Gracias a esto, se registró una gran cantidad de información (Tabla A.1 del Anexo), entre la que se encuentra:

- Conos detectados dentro de la región de interés y falsos positivos fuera de ella.
- Datos individuales de cada cono, como el número de anillos que lo componían y el número de puntos asociados.
- Tiempos de ejecución de cada etapa del *pipeline*.
- Evolución del número de puntos tras cada fase de filtrado (*filtrado inicial*, *segmentación* y *clustering*).

Dado que la estructura del sistema original no incluía mecanismos para almacenar métricas, fue necesario desarrollar un nodo de registro adicional. Este nodo capturaba los mensajes publicados por los tres nodos y los volcaba en ficheros con un formato compatible, lo que permitió disponer de un conjunto de datos homogéneo para ambos sistemas, facilitando así su comparación objetiva.

Con el fin de garantizar la equidad en dicha comparación, se empleó en ambos casos la misma *bounding box* para delimitar la región de interés del sensor. El código original utilizaba por defecto unos límites mucho más reducidos ($x_{\max} = 10.5$, $x_{\min} = 1$, $y_{\max} = 5.0$, $y_{\min} = -5.0$, $z_{\max} = 2.5$), mientras que en las pruebas se aplicaron los siguientes parámetros, comunes a ambos sistemas:

```
node.declare_parameter('xmax', 50)    # Front
```

```

node.declare_parameter('xmin', 1)      # Back
node.declare_parameter('ymax', 50)     # Left -> Positive Value
node.declare_parameter('ymin', -50)    # Right -> Negative Value
node.declare_parameter('zmax', 2.5)    # Max Height

```

Este tamaño de región de interés permitió evaluar el comportamiento de los algoritmos ante un volumen de datos mucho mayor, incluyendo puntos situados fuera de la pista. De esta forma se pudo analizar tanto la capacidad de cada sistema para manejar cargas computacionales más elevadas como su robustez al enfrentarse a geometrías más complejas del terreno y otras estructuras fuera de la zona de pista. De este modo, los resultados obtenidos en las siguientes secciones reflejan el comportamiento real de cada *pipeline* exactamente bajo las mismas condiciones de entrada.

5.2. Eficiencia en la reducción de puntos

El sensor Ouster OS1-32 genera en cada rotación una nube de más de 65 000 puntos en el modo de alta resolución configurado para este proyecto. Procesar de forma directa tal volumen de datos sería inviable en tiempo real, por lo que el *pipeline* aplica sucesivas etapas de reducción que eliminan información redundante o irrelevante y conservan únicamente los puntos útiles para la detección de conos.

En la Figura 5.4 se muestra la evolución del número medio de puntos a lo largo de las distintas etapas del *pipeline*, comparando el sistema desarrollado con el sistema original tanto para la prueba con conos azules como para la prueba con conos amarillos.

En ambos casos se aprecia cómo el prefiltrado geométrico y la segmentación del suelo suponen una reducción drástica del volumen de datos respecto a la nube inicial. El sistema modular incluye además la etapa de *clustering*, lo que permite cuantificar de manera explícita la transición desde los puntos segmentados hasta los candidatos a cono, mientras que en el código original esta información intermedia no está disponible y la reducción se observa directamente desde los *outliers* hasta la lista final de conos.

El uso de una escala logarítmica en el eje vertical pone de manifiesto la magnitud de las reducciones, que abarcan varios órdenes de magnitud, pasando de decenas de miles de puntos en la nube inicial a un número medio de conos cercano al valor real presente en el escenario. Esta representación permite visualizar de manera clara la eficiencia de cada etapa y comparar el comportamiento de ambos sistemas en condiciones equivalentes.

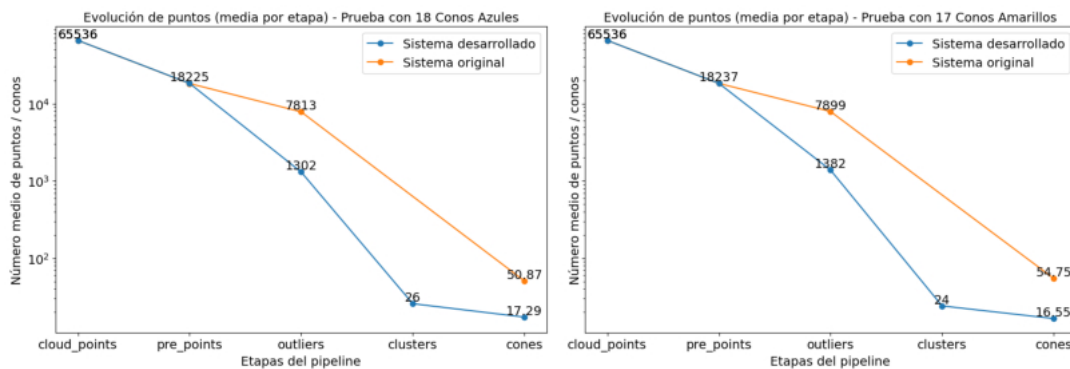


Figura 5.4: Comparación del nuevo sistema y el original en la evolución del número medio de puntos y conos a lo largo de las etapas del *pipeline*.

Estos resultados muestran una mejora muy notable en la reducción de volumetría respecto al código original. Con el nuevo sistema se consigue reducir, de media, un **92.64 %** del número de puntos en la transición de los puntos tras el primer filtrado a **outliers**, mientras que el sistema anterior alcanzaba únicamente una reducción del **56.91 %**. Esta diferencia pone de manifiesto la mayor eficacia del diseño propuesto para descartar información irrelevante y conservar únicamente los puntos útiles para la detección de conos.

Por otra parte, se aprecia que el número de detecciones obtenidas en las pruebas con el sistema original es muy superior a la cantidad real de conos presentes en pista (18 en el caso de los azules y 17 en de los amarillos). En contraste, el nuevo sistema modular ofrece un número medio de detecciones que se mantiene ligeramente por debajo de esos valores, aproximándose mejor a la realidad del escenario. En el siguiente análisis se abordará esta cuestión con mayor detalle, con el objetivo de verificar cuántas de dichas detecciones corresponden a conos reales y cuántas a falsos positivos.

Además de este análisis global, en la Figura 5.5 se representa la evolución temporal del número de puntos tras el prefiltrado y la segmentación. Como se puede observar, la nueva segmentación no solo elimina un mayor número de puntos, sino que también presenta una señal mucho más estable, con menos picos repentinos que el código original. Esto repercute directamente en la etapa de *clustering*, cuyo tiempo de procesado resulta más constante y predecible al no tener que agrupar tantos puntos.

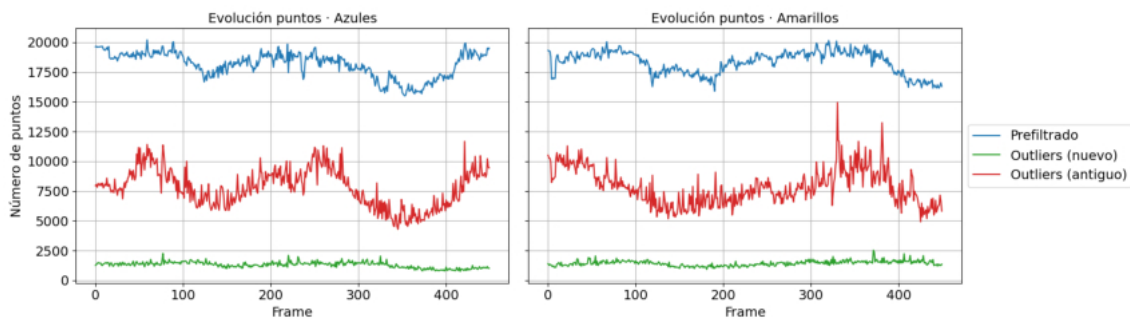


Figura 5.5: Evolución temporal del número de puntos tras el prefiltrado y la segmentación. Comparativa entre el sistema original y el desarrollado.

Gracias a estos resultados se puede confirmar que el nuevo sistema consigue una reducción de puntos más estable y eficiente. En la siguiente sección se comprueba si esta reducción ha podido comprometer la información relevante para la detección de conos.

5.3. Detección de conos

En esta sección se analizan los resultados obtenidos en la etapa de detección de conos. La Figura 5.6 muestra la evolución temporal del número total de detecciones en cada *frame*, comparando el *pipeline* desarrollado con el código original empleado como referencia.

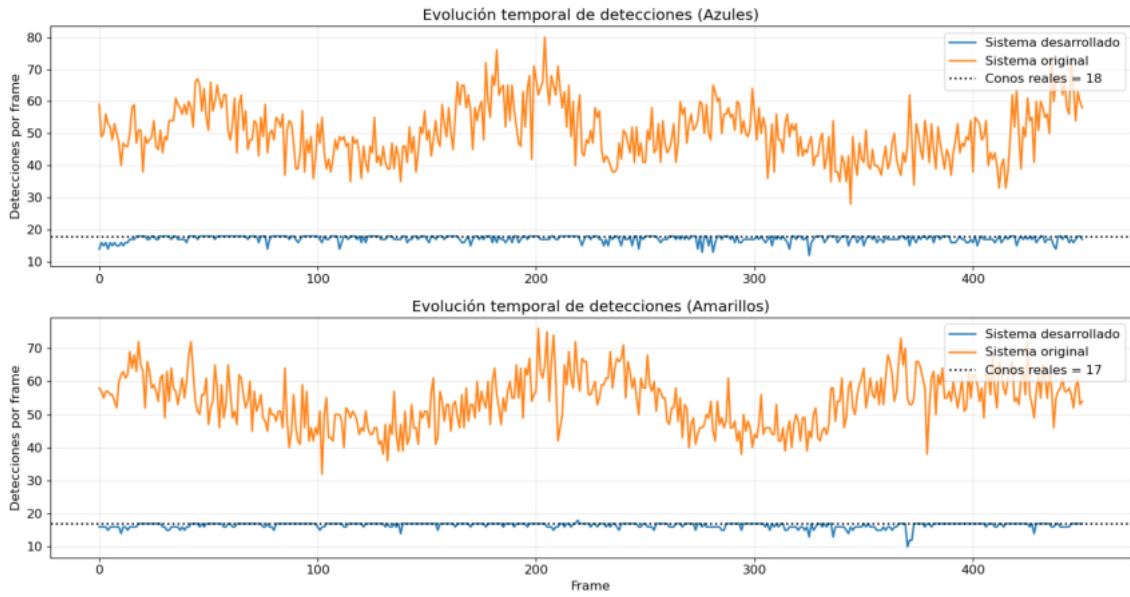


Figura 5.6: Evolución temporal del número de conos detectados por *frame* en cada sistema.

Como se puede ver, el sistema desarrollado mantiene el número de detecciones de forma estable y cercana al valor real de conos presentes en pista, aunque subestima este número ligeramente. En contraste, el código original presenta una gran disparidad, alcanzando picos muy superiores al máximo esperable: hasta 76 en la prueba de conos amarillos y 80 en la de conos azules.

A continuación, en la Figura 5.7 se representa la evolución temporal del número de conos detectados, distinguiendo entre aquellos que fueron correctamente identificados y los que resultaron ser falsos positivos.

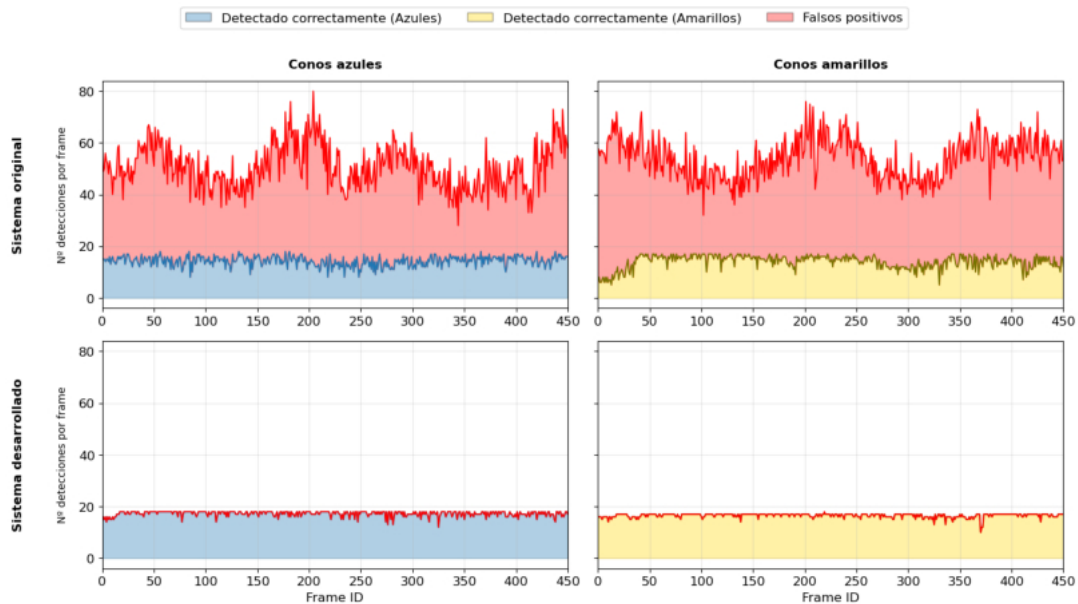


Figura 5.7: Evolución temporal del número de conos detectados, separando correctos y falsos positivos.

Se observa que, aunque el sistema original acumula siempre un mayor número total de detecciones, la proporción de conos realmente correctos es menor que en el nuevo sistema. En otras palabras, la sensibilidad de detección del sistema anterior no se traduce en una mayor capacidad para identificar conos válidos, sino en un exceso de falsos positivos que degradan la calidad de la información percibida. Por otra parte, el sistema desarrollado en este proyecto detecta menos elementos en total, pero prácticamente todas las detecciones corresponden a conos válidos. Es especialmente relevante señalar que entre las dos pruebas únicamente existen dos *frames* con un único falso positivo, lo que confirma su elevada fiabilidad.

Dataset	Detecciones correctas	Falsos positivos	Precisión
Amarillos · Nuevo desarrollo	7430	1	0.9999
Amarillos · Sistema original	6296	18048	0.2586
Azules · Nuevo desarrollo	7760	1	0.9999
Azules · Sistema original	6430	16314	0.2827

Tabla 5.1: Comparación del número de conos correctamente detectados, falsos positivos y precisión en los dos sistemas (450 *frames*).

Como se puede apreciar, la reducción de falsos positivos es prácticamente total. En los conos amarillos se pasa de **18048** a **1**, mientras que en los azules se reduce de **16314** a **1**, lo que supone una reducción del 99,994 % aproximadamente en ambos casos. Esta mejora supone que la información entregada al planificador es mucho más limpia y fiable, eliminando prácticamente por completo detecciones espurias que en el sistema original dominaban los resultados.

Este mismo fenómeno puede apreciarse desde otra perspectiva en la Figura 5.8, donde se representa la distribución del número de conos reales detectados por *frame*.

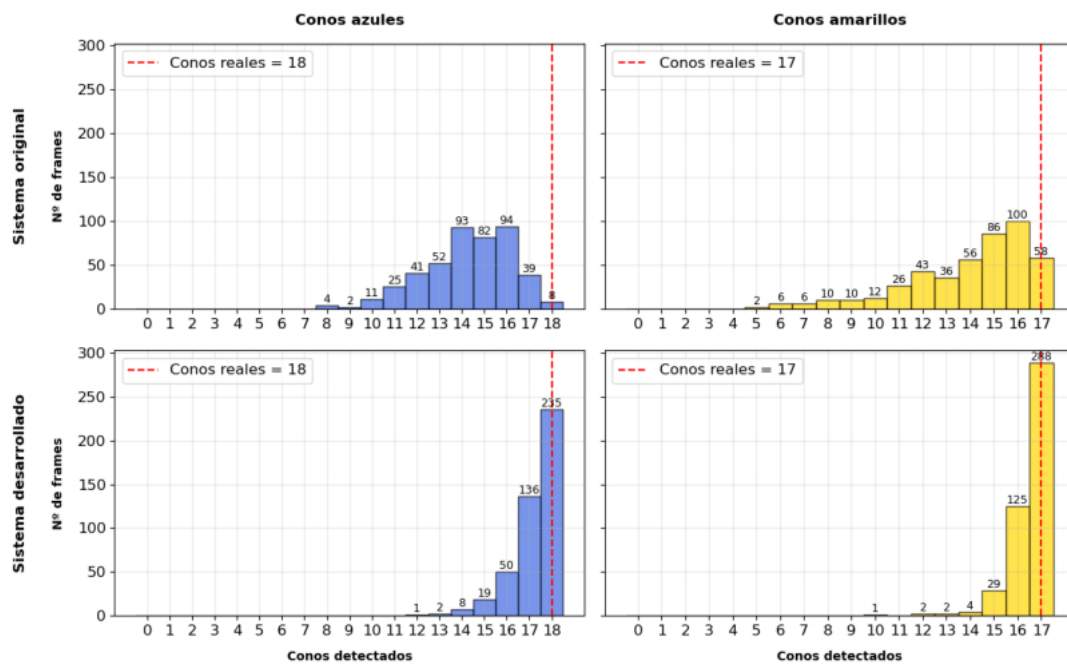


Figura 5.8: Distribución del número de conos reales detectados por *frame* en cada sistema.

La comparación muestra cómo el sistema concentra la práctica totalidad de sus detecciones exactamente en torno al número real de conos presentes en pista (17–18), lo que refleja un comportamiento estable y coherente. En cambio, el sistema original reparte sus detecciones en un rango más amplio, con numerosos *frames* en los que se detectan menos conos válidos de los que realmente había en pista. Esta dispersión confirma que, a pesar de detectar más elementos en términos absolutos, gran parte de ellos no corresponden a conos válidos y suponen falsos positivos o detecciones inconsistentes.

Finalmente, en la Figura 5.9 se muestra la acumulación espacial de todas las detecciones registradas en las pruebas. Los conos correctos se representan con puntos y los falsos positivos con cruces, utilizando como color el asignado por el sistema.

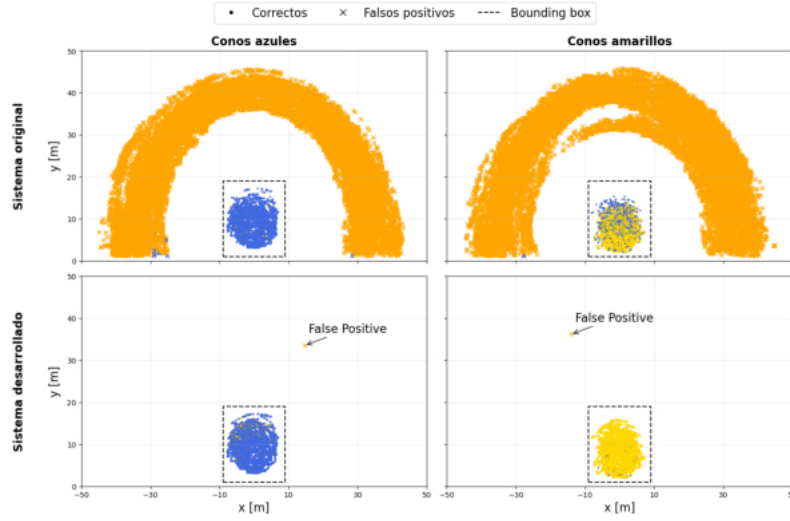


Figura 5.9: Acumulado espacial de todas las detecciones.

Como se puede ver, los conos reales están agrupados en el centro, dentro de la *Bounding box* mencionada. Se aprecia cómo el *pipeline* desarrollado concentra las detecciones en posiciones coherentes con el trazado real del circuito, a falta de un falso positivo en cada prueba, mientras que el sistema original presenta una dispersión notablemente mayor, con falsos positivos repartidos en zonas donde no existían conos reales.

Esta cantidad de falsos positivos se debe al incorrecto manejo de grandes estructuras y geometrías complejas en el terreno en la etapa de *clustering*. En la siguiente figura se muestra un ejemplo en el que una valla y unos árboles lejanos han sido detectados como una gran cantidad de conos naranjas por el sistema original:

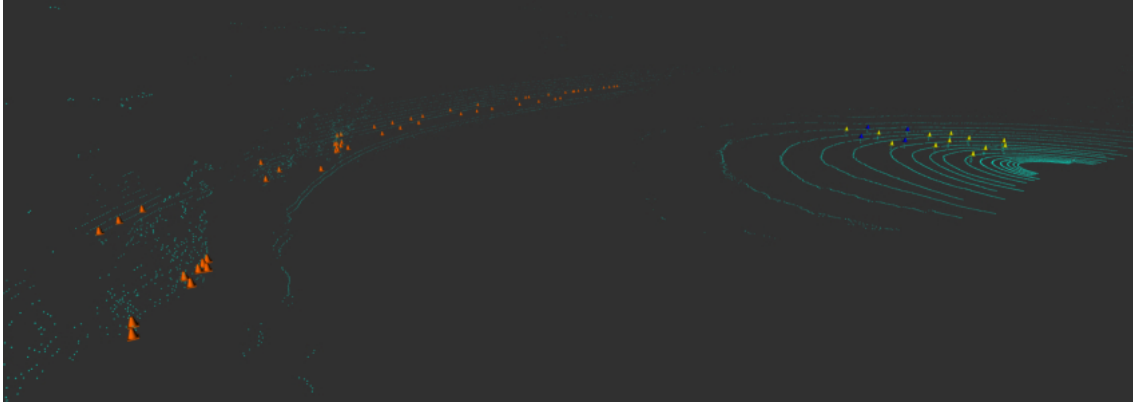


Figura 5.10: Ejemplo de falsos positivos en el sistema original.

En cuanto a la distancia de detección, la Tabla 5.2 muestra la distribución de las detecciones reales en función de la distancia al cono, agrupadas en intervalos de 5 metros. Se comparan los resultados obtenidos con el sistema desarrollado frente al código original.

Intervalo (m)	Sistema desarrollado	Sistema original
0–5	1487	1399
5–10	7456	7025
10–15	5951	4220
15–20	330	110
Detecciones totales	15 224	12 754

Tabla 5.2: Distribución de detecciones de conos reales por intervalos de distancia

Se observa que, en total, el sistema nuevo consigue detectar un mayor número de conos (15 224) frente al sistema original (12 754), con especial ventaja en los rangos de 10–15 metros, donde lo supera en más de 1 700 detecciones. Además, también mejora en las detecciones cercanas (0–5 m y 5–10 m), lo que refleja una mayor consistencia en todo el rango de operación. Sin embargo, la diferencia más significativa se aprecia a partir de los 15 metros, donde ambos sistemas reducen drásticamente el número de detecciones. Este hecho puede deberse, además de la baja cantidad de puntos que presentan los conos a estas distancias, a que en la prueba había menos conos colocados en ese intervalo. Aun así, en la comparativa se aprecia que el sistema desarrollado detecta exactamente el triple de conos que el código original, habiendo sido probados en igualdad de condiciones.

En conjunto, estos resultados evidencian que la solución desarrollada amplía el alcance efectivo de detección y mantiene una mayor fiabilidad en la identificación de conos en todas las distancias. A ello se suma una reducción notable de falsos positivos respecto al sistema original, lo que refuerza la estabilidad global de la percepción. En definitiva, el sistema desarrollado no solo detecta más conos reales, sino que también ofrece una salida más limpia y coherente, mejorando las capacidades de percepción necesarias para la navegación autónoma.

5.4. Clasificación de color

Una vez verificado el alcance y la fiabilidad de las detecciones, resulta esencial evaluar la capacidad del sistema desarrollado para distinguir correctamente el color de cada cono. Esta tarea representa uno de los mayores cambios introducidos respecto al código original, ya que se sustituyó el enfoque heurístico basado en cocientes de intensidad por un clasificador de aprendizaje automático entrenado sobre descriptores de reflectividad.

5.4.1. Comparativa de modelos candidatos

Para seleccionar el modelo final se evaluaron cuatro alternativas clásicas: regresión logística (LogReg), SVMs con *kernel* RBF, bosques aleatorios (*Random Forest*) y XGBoost. Cada uno se entrenó mediante validación cruzada estratificada (5 *folds*) empleando *GridSearchCV* sobre un espacio acotado de hiperparámetros. La Tabla 5.3 resume el mejor resultado de cada modelo (según F1 medio en CV), junto con los parámetros asociados, mientras que en la Tabla 5.4 se resumen los principales resultados de los modelos evaluados.

Modelo	# Cand.	Mejor configuración
SVM (RBF)	9	C = 5 gamma = “scale”
LogReg	3	C = 1
<i>Random Forest</i>	18	max_depth = None min_samples_leaf = 1 n_estimators = 200
XGBoost	24	colsample_bytree = 1.0 learning_rate = 0.1 max_depth = 5 subsample = 1.0

Tabla 5.3: Espacio de búsqueda y mejores hiperparámetros por modelo (selección por F1 en CV).

Modelo	F1 (CV)	PR-AUC	ROC-AUC	Accuracy	Tiempo de ajuste [s]
SVM (RBF)	0.9911	0.9992	0.9992	0.9910	155.38
LogReg	0.9851	0.9979	0.9980	0.9849	1.21
<i>Random Forest</i>	0.9905	0.9991	0.9993	0.9904	96.06
XGBoost	0.9918	0.9996	0.9996	0.9918	35.11

Tabla 5.4: Comparación de modelos candidatos para la clasificación de color (validación cruzada).

Los resultados muestran un rendimiento muy similar entre SVM, *Random Forest* y XGBoost,

con una ligera ventaja de este último en F1 y métricas de área bajo curva, además de tiempos de ajuste considerablemente menores. Por este motivo, se seleccionó XGBoost como clasificador final.

5.4.2. Rendimiento del clasificador XGBoost

Una vez elegido, el modelo fue evaluado sobre el conjunto de test independiente. Los resultados globales se resumen a continuación:

- Accuracy: 0.9924
- Precisión: 0.9935
- Recall: 0.9915
- F1-score: 0.9925
- ROC-AUC: 0.9997
- PR-AUC: 0.9997

La matriz de confusión asociada a este experimento (Tabla 5.5) confirma un excelente comportamiento, con muy pocos errores de clasificación entre conos azules y amarillos.

	Pred. Azul	Pred. Amarillo
Azul real	1494	10
Amarillo real	13	1524

Tabla 5.5: Matriz de confusión en el conjunto de test (XGBoost).

5.4.3. Comparación final con Random Forest

Para corroborar la elección, se comparó el clasificador XGBoost entrenado con los mejores hiperparámetros frente a un *Random Forest* de referencia (40 árboles). A diferencia de las evaluaciones anteriores, en esta comparativa se empleó el conjunto completo de datos (entrenamiento + test). El objetivo no era estimar la capacidad de generalización —ya validada mediante *GridSearchCV* y el conjunto de test independiente (Tablas 5.4 y 5.5)— sino disponer de una referencia directa entre modelos en condiciones idénticas y sobre el mayor volumen de datos posible. De este modo, se evaluaron tanto las métricas clásicas como la latencia media de inferencia por muestra, un aspecto crítico para la ejecución en tiempo real en el vehículo.

Métrica	XGBoost	Random Forest
Accuracy	0.9985	0.9968
Precisión	0.9987	0.9982
Recall	0.9983	0.9954
F1-score	0.9985	0.9968
ROC-AUC	1.0000	0.9997
PR-AUC	1.0000	0.9996
Latencia [ms/muestra]	0.004	0.006

Tabla 5.6: Comparativa de rendimiento entre XGBoost y *Random Forest* en el conjunto completo.

Los resultados confirman la ligera superioridad del modelo XGBoost, tanto en métricas clásicas como en latencia de inferencia. En particular, la latencia media de 0,004 ms por muestra permite ejecutar la clasificación de color sin comprometer el ciclo de 10 Hz del sensor LiDAR, lo que confirma su idoneidad para aplicaciones en tiempo real. La matriz de confusión asociada (Tabla 5.7) muestra una tasa de acierto prácticamente perfecta.

	Pred. Azul	Pred. Amarillo
Azul real	7509	10
Amarillo real	13	7671

Tabla 5.7: Matriz de confusión con el modelo final XGBoost (conjunto completo).

De forma análoga, en la Tabla 5.8 se presenta la matriz de confusión obtenida con el *Random Forest* de referencia. En este caso se observa un mayor número de errores, especialmente en conos amarillos, lo que refuerza la elección de XGBoost como modelo final.

	Pred. Azul	Pred. Amarillo
Azul real	7505	14
Amarillo real	35	7649

Tabla 5.8: Matriz de confusión con el modelo *Random Forest* (conjunto completo).

Los experimentos muestran que el clasificador XGBoost supera a las alternativas evaluadas tanto en rendimiento como en eficiencia de inferencia, consolidándose como la opción idónea para la clasificación de color en el sistema desarrollado. A continuación, se presenta una comparación directa con el enfoque heurístico original, lo que permitirá cuantificar el salto obtenido respecto a la estrategia utilizada en las versiones previas del sistema.

5.4.4. Pruebas en entornos reales

Para evaluar el rendimiento de la clasificación de color se ha empleado la información registrada en las mismas pruebas realizadas en la pista del INSIA, ya descritas en los apartados anteriores. A partir de estos datos se han analizado los aciertos, fallos y *unknowns* obtenidos con el código original, con el *Random Forest* y con el clasificador *XGBoost*.

Dataset	Aciertos	Fallos	Unknowns	Precisión
Amarillos · Original	4623	1673	0	0.734
Amarillos · RF	7289	6	135	0.981
Amarillos · XGB	7419	7	17	0.997
Azules · Original	6336	94	0	0.985
Azules · RF	7128	49	583	0.919
Azules · XGB	7582	68	107	0.977

Tabla 5.9: Resultados de la clasificación de color en las pruebas en la pista del INSIA, incluyendo la precisión considerando como fallo tanto los errores como los *unknowns*.

Los resultados ponen de manifiesto las limitaciones del código original, especialmente en la clasificación de conos amarillos, donde se acumulan un gran número de fallos. En contraste, este enfoque heurístico lograba un desempeño relativamente mejor en conos azules, lo que sugiere una cierta tendencia a clasificarlos con mayor facilidad. El nuevo sistema, probado tanto con el *Random Forest* como con *XGBoost*, invierte parcialmente esta situación: reduce drásticamente los errores en amarillos, aunque introduce más en el caso de los azules. Entre ambos clasificadores, el *Random Forest* presenta un número elevado de *unknowns*, mientras que *XGBoost* ofrece un mejor equilibrio, manteniendo un número ligeramente superior de fallos pero reduciendo de forma significativa los casos sin clasificar.

Para analizar con mayor detalle la distribución de errores por *frame*, la Figura 5.11 muestra, para cada enfoque y color, un histograma del número de conos mal clasificados por *frame*. El apilado distingue explícitamente los casos *Unknown* (gris, parte inferior) de los fallos (rojo, parte superior). El bin 0 representa los *frames* sin errores.

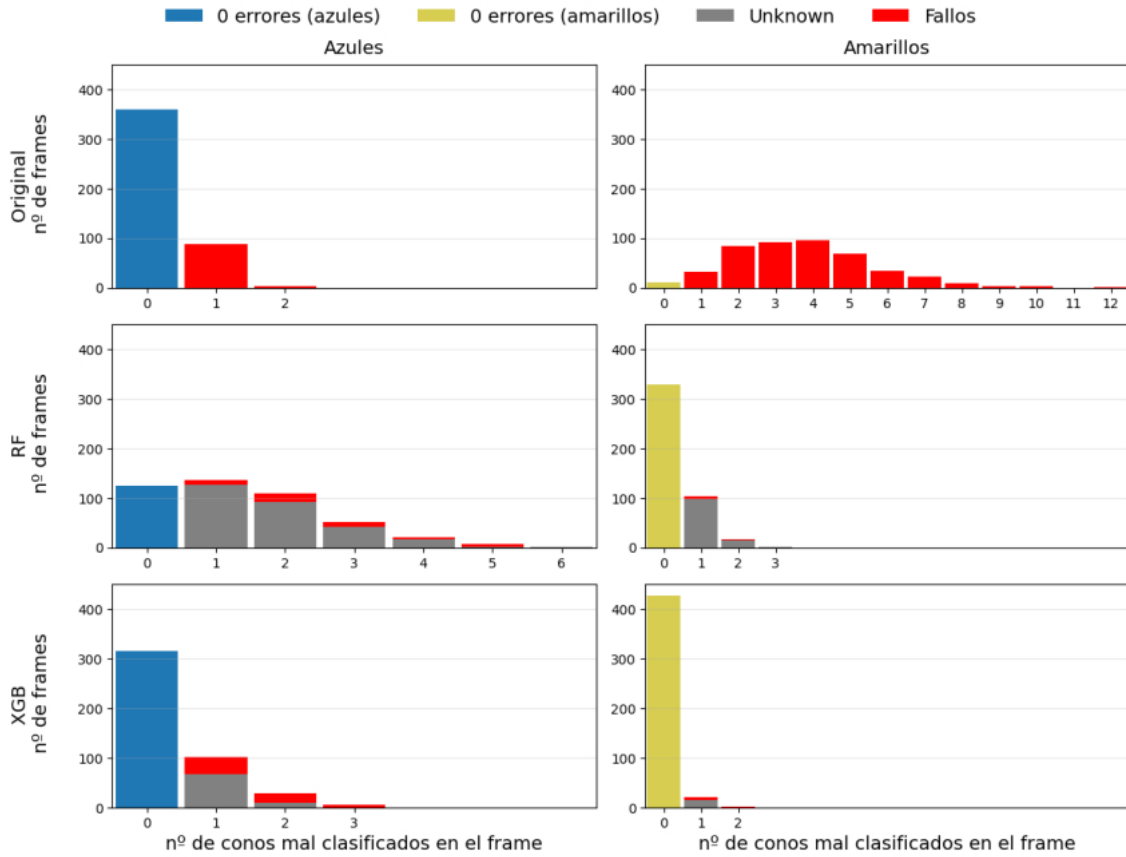


Figura 5.11: Distribución de nº de conos mal clasificados por *frame*.

La figura corrobora las tendencias observadas. En la prueba con conos amarillos (columna derecha):

- El sistema **original** presenta una cola extensa hacia *bins* altos (incluso 12 errores), con sólo 10/450 *frames* sin fallos (2,22 %).
- Con el **Random Forest** la masa se desplaza claramente al cero (329/450; 73,11 %), aunque gran parte de los errores residuales son *Unknown* (gris)
- **XGB** concentra aún más la distribución en el *bin* 0 (428/450; 95,11 %) y reduce de forma notable los *Unknown*, dejando una cola mínima de fallos reales.

Por otra parte, en la prueba con conos azules (columna izquierda):

- El sistema **original** parte de una situación favorable, con 359/450 *frames* sin errores (79,78 %).
- El **Random Forest** empeora este caso (125/450; 27,78 %), introduciendo una cola adicional donde la mayoría de errores son de conos clasificados como *Unknown*.
- **XGBoost** recupera buena parte del rendimiento (315/450; 70,00 %) y disminuye de forma clara los *Unknown*, aunque mantiene una cola corta con mayor proporción de errores de color que el *Random Forest*.

En conjunto, el histograma apilado refleja un comportamiento diferenciado según el color de los conos. El **Random Forest** se muestra especialmente sólido en la clasificación de amarillos, donde consigue desplazar la mayor parte de los *frames* al *bin* 0, aunque a costa de un mayor número de *Unknown*, lo que lo hace atractivo en escenarios donde resulta preferible la prudencia frente a cometer errores de color.

Por su parte, **XGBoost** concentra más aún la distribución en torno a cero errores y reduce notablemente la proporción de *Unknown*, aunque en conos azules introduce una cola con cierta cantidad de fallos de color. Esto lo convierte en un enfoque más adecuado cuando la prioridad es maximizar el número de aciertos incluso a costa de asumir algún error residual.

El sistema **original**, en cambio, queda desbordado en el caso de los amarillos y sólo aparenta un rendimiento aceptable en la detección de azules, pero esta diferencia responde a una clara tendencia del enfoque heurístico a favorecer dicho color, por lo que no se puede confiar en ese resultado como indicador real de robustez.

La Figura 5.12 complementa el análisis del histograma de errores, mostrando cómo evoluciona la clasificación en función de la distancia.

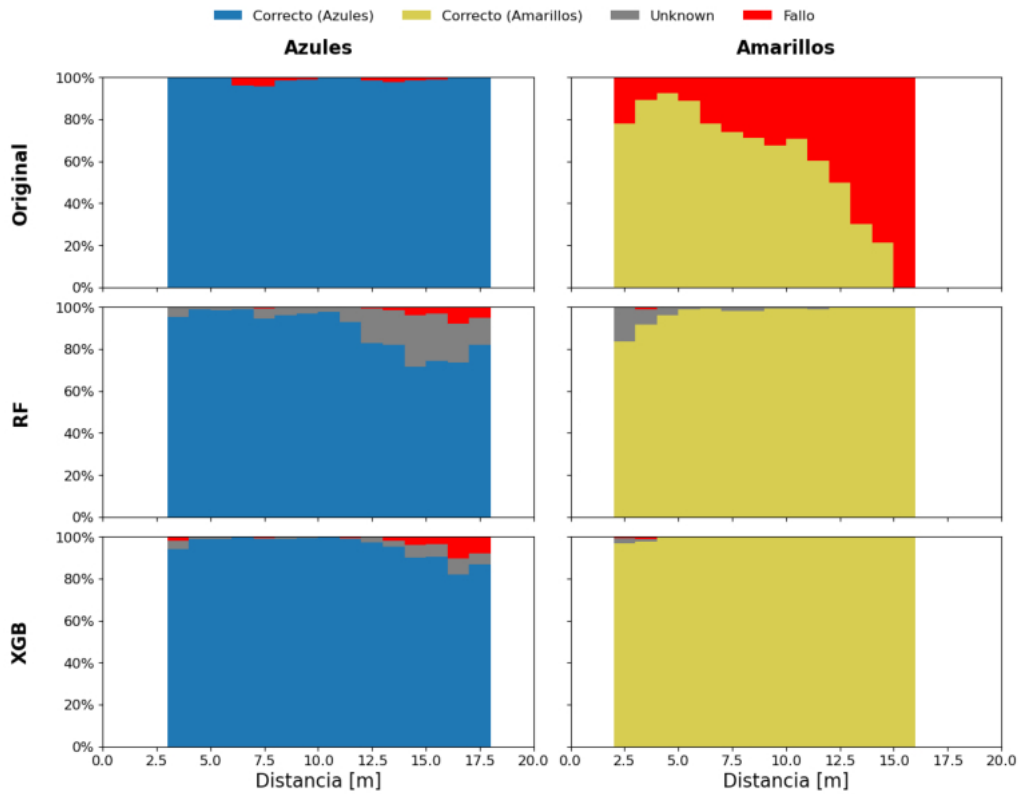


Figura 5.12: Porcentaje de aciertos, fallos y *Unknowns* en función de la distancia de detección.

En la prueba de conos amarillos, el sistema original es claramente deficiente. Aparecen fallos en todo el rango, crecientes con la distancia, hasta el punto de que en el último intervalo no se consigue ningún acierto. Por el contrario, tanto *Random Forest* como *XGBoost* eliminan estos fallos en distancias largas. En ambos casos los conos amarillos se clasifican de forma correcta incluso en detecciones lejanas, aunque en distancias cortas *Random Forest* presenta más *Unknowns*, mientras que con *XGB* apenas se observan errores.

Con conos azules, la situación es diferente. Con el sistema original los fallos son dispersos

y menos problemáticos que en los amarillos. *Random Forest* clasifica algunos *Unknowns* ya en distancias cortas y se vuelven dominantes a partir de medias-largas, aunque sin apenas fallos de color. XGB, en cambio, mantiene un buen nivel de aciertos hasta distancias algo mayores, pero a costa de introducir más fallos de color conforme la distancia aumenta.

En conjunto, se aprecia que *Random Forest* es más conservador: tiende a clasificar como *Unknown* antes que arriesgarse a cometer errores. Por su parte, XGB mantiene una mayor tasa de aciertos en rangos más amplios, pero introduce fallos residuales en conos azules a largas distancias. El sistema original, en cambio, se confirma como insuficiente, especialmente en amarillos donde su comportamiento es claramente inaceptable.

5.5. Rendimiento temporal

Además de la precisión en la detección y clasificación de conos, como ya se ha mencionado, el sistema debe cumplir con requisitos estrictos de latencia para poder operar en tiempo real. El sensor impone un límite de **100 ms** para procesar cada nube de puntos. Si el *pipeline* de percepción supera dicho umbral, se acumularía retraso entre la información adquirida y la disponible para planificación y control, comprometiendo el funcionamiento autónomo del vehículo. Por este motivo, el análisis del rendimiento temporal es fundamental para validar la viabilidad de la solución propuesta.

Aunque la segmentación mediante *RANSAC* resulta en general muy rápida en comparación con otros métodos, un número excesivo de iteraciones incrementa notablemente su coste computacional, por lo que resulta fundamental evaluar su rendimiento en función de este parámetro.

La Tabla 5.10 resume los resultados obtenidos tras ejecutar la segmentación con diferentes valores de iteraciones (10, 50, 100, 200 y 400). En ella se muestra el tiempo medio de ejecución, así como el número medio de puntos clasificados como *inliers* (eliminados al ajustar al plano) y como *outliers* (que continúan en la nube de puntos para etapas posteriores).

Iteraciones	Tiempo medio [ms]	Inliers medios	Outliers medios
10	0.49	16,814	1,419
50	1.31	16,861	1,372
100	2.27	16,858	1,375
200	4.22	16,871	1,362
400	8.07	16,871	1,362

Tabla 5.10: Rendimiento medio de la segmentación RANSAC en función del número de iteraciones.

En la Figura 5.13 se representa la evolución conjunta de estas métricas. En el eje izquierdo se muestran el número de *inliers* eliminados y de *outliers* restantes, mientras que en el eje derecho se representa el tiempo medio de ejecución por configuración.

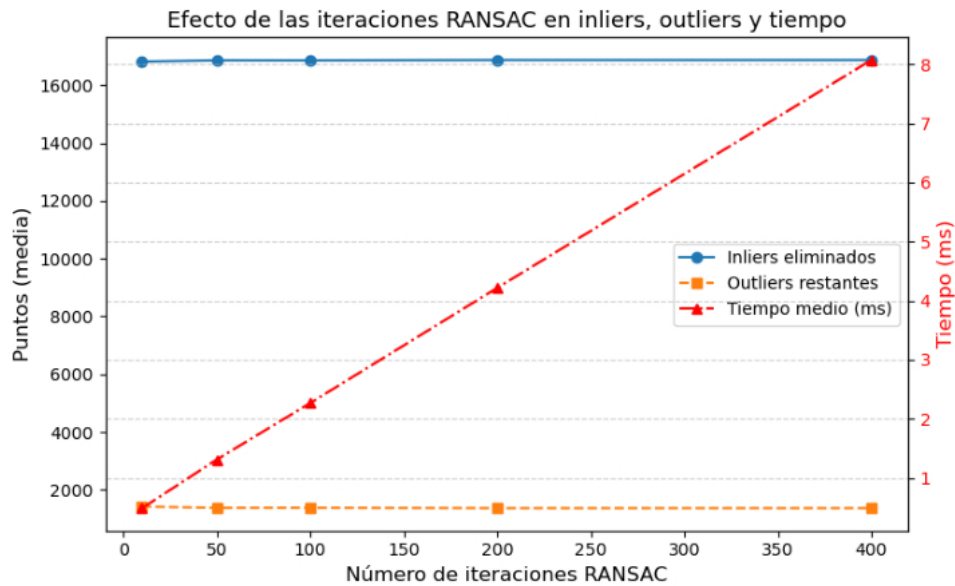


Figura 5.13: Influencia del número de iteraciones de RANSAC en el tiempo de ejecución y en la clasificación de puntos como *inliers* o *outliers*.

Los resultados muestran que el tiempo de procesamiento crece de forma prácticamente lineal con el número de iteraciones, pasando de apenas 0.5 ms con 10 iteraciones a más de 8 ms con 400. Por el contrario, el número de puntos clasificados como *inliers* o *outliers* permanece estable a lo largo de todas las configuraciones, con diferencias mínimas entre ellas. Esto indica que un incremento significativo del coste computacional no se traduce en mejoras apreciables en la segmentación del plano, por lo que valores moderados de iteraciones (entre 50 y 100) constituyen un compromiso adecuado entre precisión y eficiencia. En base a este compromiso entre precisión y eficiencia, para el resto de las pruebas del sistema se ha fijado en **70** el número de iteraciones de *RANSAC*.

Una vez fijado el número de iteraciones de la etapa de segmentación, resulta de interés estudiar el tiempo total de procesamiento de los sistemas. La Figura 5.14 muestra el tiempo acumulado de ejecución para las pruebas anteriores con conos amarillos y azules, comparando el sistema original con los nuevos clasificadores **Random Forest** (RF) y **XGBoost** (XGB). Como puede apreciarse, el nuevo sistema presenta un rendimiento temporal prácticamente idéntico con ambos clasificadores, en contraste con la considerable diferencia respecto al sistema original.

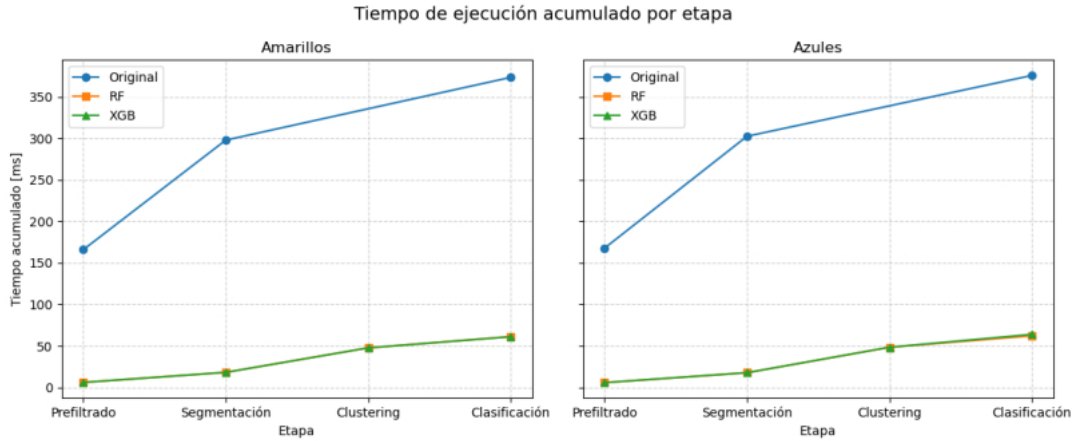


Figura 5.14: Tiempo acumulado de ejecución por etapa del *pipeline* para conos amarillos (izquierda) y azules (derecha).

5.5.1. Tiempos globales de ejecución

En las Tablas 5.11 y 5.12 se presentan los tiempos medios de cada etapa del sistema, sin acumular, tanto para el **sistema original** como para los clasificadores **Random Forest** (RF) y **XGBoost** (XGB).

Sistema	Prefiltrado	Segmentación	Clustering	Clasificación
Sistema original	165.85	131.89	75.54	-
RF	6.10	12.10	29.51	13.47
XGB	6.02	12.10	30.12	12.77

Tabla 5.11: Tiempos medios de ejecución por etapa para conos amarillos [ms].

Sistema	Prefiltrado	Segmentación	Clustering	Clasificación
Sistema original	167.71	134.68	73.25	-
RF	5.75	11.96	30.64	13.72
XGB	6.37	12.33	31.80	13.40

Tabla 5.12: Tiempos medios de ejecución por etapa para conos azules [ms].

Como era de esperar, se observa cómo en el sistema actual (RF y XGB) todas las etapas tienen prácticamente el mismo peso. Existen diferencias mínimas en todas ellas, debidas tanto a variabilidad inherente a las condiciones de ensayo, como a la aleatoriedad del algoritmo RANSAC en la segmentación. Sin embargo, estas diferencias son insignificantes en términos absolutos y no afectan a la capacidad de operar en tiempo real.

Por otra parte, en el **sistema original** estas dos fases resultaban con diferencia las más lentas, debido a los repetidos ciclos de conversión entre **PointCloud2** y arrays descritos en la Sección 4.3.1, lo que incrementaba notablemente la latencia. En el sistema desarrollado se eliminan

estas operaciones innecesarias y, además, todas las transformaciones se han vectorizado, reduciendo drásticamente el tiempo de ejecución total.

También se ha representado la evolución del tiempo de procesamiento acumulado a lo largo de los primeros 450 *frames*. Dado que los resultados obtenidos con *Random Forest* y *XGBoost* eran prácticamente idénticos, en esta comparación se ha considerado únicamente el clasificador *XGBoost*.

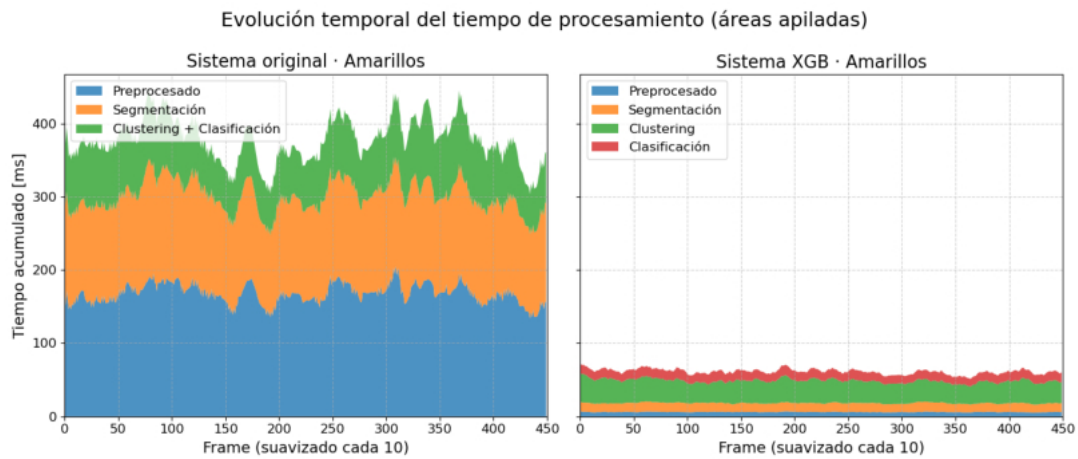


Figura 5.15: Evolución temporal del tiempo de procesamiento acumulado en cada etapa del sistema original (izquierda) y del sistema desarrollado con *XGBoost* (derecha).

Como se aprecia en la Figura 5.15, el nuevo sistema mantiene tiempos reducidos y estables en cada etapa, con una acumulación progresiva clara y sin picos significativos.

En el caso del sistema original, la división en tres nodos que se ejecutan en paralelo evita que la pérdida de *frames* sea tan acusada como cabría esperar. No obstante, el tiempo de procesado individual de cada nodo es lo suficientemente elevado como para que, cada cierto número de *frames*, alguno se pierda igualmente al no completarse antes de la llegada del siguiente, comprometiendo así la robustez en condiciones de tiempo real.

Con el fin de evaluar los escenarios más exigentes, se ha analizado el tiempo máximo registrado en cada etapa del *pipeline*, así como dos métricas agregadas: (i) el **máximo acumulado por frame**, que refleja el peor caso observado en la práctica, y (ii) el **acumulado de los máximos**, que corresponde a un escenario hipotético en el que los picos de cada etapa coincidiesen en un mismo *frame*.

Métrica	Tiempo [ms]
Preprocesado (t_{pre_ms})	298.93
Segmentación (t_{seg_ms})	241.78
<i>Clustering</i> + Clasificación	151.25
Máximo acumulado por <i>frame</i>	604.78
Acumulado de los máximos	691.95

Tabla 5.13: Peor caso (entre amarillos y azules) para el sistema original.

Métrica	Tiempo [ms]
Preprocesado (t_{pre_ms})	11.19
Segmentación (t_{seg_ms})	18.12
<i>Clustering</i> (t_{clu_ms})	58.86
Clasificación (t_{cla_ms})	23.90
Máximo acumulado por <i>frame</i>	92.59
Acumulado de los máximos	112.07

Tabla 5.14: Peor caso (entre amarillos y azules) para el sistema con **XGB**.

Tal y como muestran las Tablas 5.13 y 5.14, el sistema original alcanza valores máximos de hasta **604 ms** por *frame*, lo que excede con creces la cadencia de entrada del LiDAR (100 ms a 10 Hz). Incluso considerando el procesado en paralelo, este retardo inevitablemente provoca la pérdida de *frames*. El acumulado hipotético supera los 690 ms, evidenciando la elevada latencia que podían alcanzar ciertas iteraciones.

En contraste, el sistema con **XGBoost** reduce drásticamente los tiempos: el peor *frame* no supera los **93 ms**, ajustándose al límite temporal impuesto por la frecuencia del sensor. Además, incluso en el escenario de picos coincidentes, el acumulado máximo apenas sobrepasa los 112 ms, por lo que, aunque se diese este caso, muy probablemente en la siguiente iteración se recuperaría el tiempo. Estos resultados confirman que el nuevo sistema es capaz de mantener el procesado en tiempo real sin pérdidas significativas de información.

Para valorar mejor las implicaciones prácticas de esta diferencia, la Tabla 5.15 muestra el avance lineal que experimentaría el vehículo durante el tiempo de procesado máximo en cada sistema.

Velocidad	Avance en 604 ms (Original)	Avance en 92 ms (XGBoost)
10 km/h	1.68 m	0.26 m
20 km/h	3.36 m	0.52 m
30 km/h	5.04 m	0.78 m
40 km/h	6.72 m	1.03 m

Tabla 5.15: Avance lineal aproximado del vehículo durante el tiempo máximo de procesado de cada sistema.

Aunque en la práctica las distintas etapas del sistema puedan ejecutarse en paralelo, el retardo total marca el instante en que los resultados finales (lista de conos detectados y clasificados) están disponibles para el resto del *stack*. Esto implica que, en el caso del sistema original, cada cono era transmitido con un desfase equivalente a varios metros de avance del vehículo, mientras que en el nuevo sistema dicho retraso se reduce a una distancia prácticamente despreciable en comparación con la distancia de detección del sensor.

5.6. Cumplimiento de objetivos

En la Sección 4.3.2 se han expuesto los objetivos de este proyecto. A continuación se resume en qué medida se han alcanzado, apoyándose en los resultados presentados en este capítulo.

1. Aumentar el alcance de detección

El sistema desarrollado consigue detectar un total de 15 224 conos reales, frente a los 12 754 del sistema original (Tabla 5.2). Esto supone un incremento del 19,4 % en detecciones totales, con un 200 % más de conos detectados en el rango 15–20 m, extendiendo por tanto de manera efectiva el alcance de percepción sin comprometer la fiabilidad.

2. Reducir falsos positivos

La reducción es prácticamente total: de 18 048 falsos positivos en amarillos y 16 314 en azules con el sistema original, se pasa a 1 en cada prueba con el nuevo desarrollo (Tabla 5.1). Esto equivale a una reducción del 99,994 %, eliminando por completo las detecciones espurias que saturaban la salida del sistema previo.

3. Mejorar la clasificación de color

El clasificador XGBoost alcanza una *accuracy* del 99,85 %, con precisión y *recall* por encima del 99,8 % en todas las clases (Tabla 5.6). En la validación en pista se obtuvieron 7419/7443 aciertos en amarillos (99,7 %) y 7582/7657 en azules (99,0 %), confirmando un rendimiento muy superior al enfoque heurístico original, especialmente en conos amarillos, donde los fallos se reducen drásticamente.

4. Reestructuración modular del *pipeline*

El sistema se ha reorganizado en módulos independientes y parametrizados (prefiltrado, segmentación, *clustering* y clasificación), lo que ha facilitado tanto la depuración, como el remplazo y concatenación de algoritmos y modelos. Además, este enfoque permite realizar configuraciones accesibles desde *launchfiles*, de modo que se pueden tener varios preparados con distintos parámetros en función del escenario al que se va a enfrentar el sistema.

5. Robustez y eficiencia en tiempo real

El tiempo de procesamiento máximo observado en un *frame* fue de 92,6 ms (Tabla 5.14), cumpliendo el límite de 100 ms impuesto por la frecuencia del sensor (10 Hz). En contraste, el sistema original alcanzaba picos de 604 ms, provocando pérdidas de *frames*. Esto confirma que el nuevo desarrollo garantiza operación en tiempo real bajo condiciones de competición.

6. Integración de herramientas de validación

Se han incorporado registros automáticos en formato CSV y representaciones en RViz para todas las etapas, lo que ha permitido analizar tanto métricas temporales como espaciales de forma cuantitativa.

7. Facilitación de la continuidad del proyecto

Todo el código ha sido estructurado y documentado siguiendo un esquema modular muy intuitivo. La estructura modular permite identificar fácilmente la función de cada componente del *pipeline*, incluso cuando los métodos internos presentan elevada complejidad.

En conjunto, puede afirmarse que todos los objetivos planteados han sido cumplidos. El sistema resultante amplía las detecciones en un 20 %, reduce los falsos positivos en un 99,994 %, mejora la clasificación de color hasta una precisión superior al 99,8 % y mantiene la latencia muy por debajo del límite de 100 ms, validando así la viabilidad de la solución propuesta en un entorno de competición real.

5.6.1. Limitaciones del sistema

A pesar de los buenos resultados obtenidos, el sistema presenta varias limitaciones que conviene destacar. En primer lugar, el desempeño es sensible al número de anillos captados por el LiDAR en cada cono. Se ha comprobado que con menos de tres anillos resulta complicado distinguir entre un falso positivo y un cono real. Con únicamente dos anillos, la malla de reflectividad utilizada por el clasificador puede resultar demasiado pobre: existe la posibilidad de que ninguno de los dos anillos coincida con la banda del cono, lo que conlleva errores en la clasificación de color.

El número de anillos disponibles depende tanto de la distancia como de la inclinación del sensor. A mayor distancia, la separación angular entre anillos aumenta, reduciendo la probabilidad de obtener una muestra suficientemente densa en la superficie del cono. Por otra parte, una variación mínima en la inclinación del LiDAR puede provocar que un mismo objeto quede registrado con más o menos anillos, alterando la robustez del sistema.

En las pruebas realizadas se ha adoptado un umbral mínimo de tres anillos para considerar un cluster como candidato a cono. Este criterio garantiza un compromiso adecuado entre alcance y fiabilidad, pero implica una pérdida inevitable de sensibilidad a mayores distancias o en configuraciones menos favorables del sensor.

Capítulo 6

Presupuesto

El presente capítulo recoge una estimación de los recursos empleados en el desarrollo del sistema de percepción basado en LiDAR, considerando tanto los costes de *hardware* y *software* como el tiempo de trabajo dedicado. Aunque en el ámbito de *Formula Student* el esfuerzo de los estudiantes no se remunera, resulta conveniente cuantificarlo para proporcionar una visión realista de la magnitud del proyecto.

6.1. Costes de hardware

En la Tabla 6.1 se resumen los principales equipos y materiales utilizados. Los elementos más relevantes son el sensor LiDAR Ouster OS1-32 y el ordenador, cuyo precio de mercado supera ampliamente al resto de componentes.

Concepto	Unidad	Coste estimado [€]
LiDAR Ouster OS1-32	1	6 500
Ordenador embarcado (In-CarPC CQ89G)	1	4 500
Estación de carga portátil	1	600
Accesorios de integración (cables, soporte, alimentación)	–	500
Total hardware		12 100

Tabla 6.1: Costes de *hardware* empleados en el desarrollo.

6.2. Costes de software

Todo el *software* utilizado en el proyecto proviene de herramientas de código abierto, como *Python*, *ROS 2 Humble*, *scikit-learn*, *XGBoost* o *Ubuntu 22.04* y de licencias institucionales facilitadas por la Universidad. Esto permite disponer de un entorno de desarrollo completo sin necesidad de asumir costes directos en licencias comerciales. No obstante, con el fin de reflejar el valor económico real que supondría recurrir a soluciones equivalentes en el mercado, se ha estimado un coste de referencia para incluirlo en el presupuesto global.

Concepto	Licencias	Coste [€]	Total [€]
Microsoft 365 Business Basic	13	12 Meses $\times$ 5,60 €/mes	873,6
GitHub Team	13	12 Meses $\times$ 3,50 €/mes	546,0
Total software			1419,6

Tabla 6.2: Costes de *software* empleados en el desarrollo (anualizados).

6.3. Costes de personal

El trabajo desarrollado corresponde íntegramente al esfuerzo de los miembros de la división *Driverless* de *UPM Racing*, sin remuneración económica. No obstante, para fines de estimación se asigna un coste horario de referencia de 25 €/h, equivalente al de un ingeniero junior en prácticas.

Adicionalmente, debe considerarse el tiempo de supervisión y apoyo por parte del tutor académico, que se ha estimado en 40 horas de trabajo a una tarifa de referencia de 40 €/h, lo que representa un coste de 1 600 €.

En la Tabla 6.3 se resumen estos valores, alcanzando un total de 10 600 € en concepto de personal.

Actividad	Horas	Coste unitario [€/h]	Coste [€]
Desarrollo y validación del sistema (alumno)	360	25	9 000
Supervisión y tutorías (profesor)	40	40	1 600
Total personal	400	–	10 600

Tabla 6.3: Costes de personal estimados.

6.4. Resumen del presupuesto

En la Tabla 6.4 se presenta el presupuesto global del proyecto.

Partida	Coste [€]
<i>Hardware</i>	12 100
<i>Software</i>	1 419,60
Personal	10 600
Total	24 119,60

Tabla 6.4: Presupuesto global del proyecto.

En conclusión, el desarrollo de un sistema de percepción de estas características supone un coste

estimado de **24 119,60 €**, dominado principalmente por el *hardware* especializado y las horas de trabajo técnico. La colaboración con la universidad, que provee las licencias institucionales, reduce de manera significativa el coste real afrontado por el equipo.

Capítulo 7

Impacto del proyecto

El desarrollo de un sistema de percepción para vehículos autónomos de competición no sólo tiene un interés académico y deportivo, sino que también presenta múltiples implicaciones a nivel social, económico, tecnológico y medioambiental. Asimismo, puede contribuir al avance de los *Objetivos de Desarrollo Sostenible* (ODS) definidos por la ONU.

7.1. Impacto social

El proyecto fomenta la formación de ingenieros en un entorno altamente tecnológico y competitivo, contribuyendo a mejorar la empleabilidad de los estudiantes en sectores estratégicos como la automoción, la robótica o la inteligencia artificial. Además, promueve valores de trabajo en equipo, igualdad de oportunidades y colaboración interdisciplinar, generando un entorno de aprendizaje aplicable más allá del ámbito académico.

La sociedad también se beneficia de los avances en sistemas de percepción, que pueden transferirse a vehículos autónomos comerciales, mejorando la seguridad vial y reduciendo el número de accidentes de tráfico.

7.2. Impacto en salud y seguridad

La detección fiable de obstáculos mediante sensores LiDAR es clave para la seguridad en conducción autónoma. Este trabajo contribuye al desarrollo de algoritmos que permiten una identificación precisa y en tiempo real de elementos del entorno. En un futuro, esta línea de investigación podría reducir la siniestralidad en carretera y mejorar la seguridad de peatones, ciclistas y conductores, contribuyendo directamente al ODS 3 (*Salud y bienestar*).

7.3. Impacto ambiental

Los vehículos autónomos bien optimizados pueden reducir el consumo energético y las emisiones contaminantes, al promover una conducción más eficiente y fluida. El proyecto también se apoya en *software* libre y *hardware* existente, lo que minimiza la huella ambiental en comparación con alternativas que requieren licencias comerciales o recursos materiales adicionales. De este modo, se alinea con el ODS 12 (*Producción y consumo responsables*) y el ODS 13 (*Acción por el clima*).

7.4. Impacto económico

El sector de la automoción está inmerso en una transición hacia la digitalización y la autonomía. Este proyecto se enmarca en dicha transformación, aportando soluciones que podrían reducir costes de desarrollo mediante el uso de sensores accesibles y algoritmos eficientes. La experiencia adquirida

por los estudiantes supone también una inversión en capital humano altamente cualificado, lo que contribuye al crecimiento económico y la competitividad industrial (ODS 8: *Trabajo decente y crecimiento económico*).

7.5. Impacto tecnológico e industrial

El trabajo realizado demuestra cómo la investigación en entornos universitarios puede generar soluciones innovadoras con aplicaciones reales en la industria. El diseño modular del sistema de percepción facilita su transferencia a otros contextos más allá de la competición, como la logística automatizada, la robótica móvil o los sistemas avanzados de asistencia a la conducción (ADAS). De esta manera, se contribuye al ODS 9 (*Industria, innovación e infraestructura*) y se potencia la colaboración universidad-empresa.

7.6. Contribución a los Objetivos de Desarrollo Sostenible

En síntesis, el proyecto se relaciona directamente con los siguientes ODS:

- **ODS 3: Salud y bienestar**, mediante la reducción de accidentes gracias a sistemas de percepción más fiables.
- **ODS 8: Trabajo decente y crecimiento económico**, al formar profesionales especializados en sectores estratégicos.
- **ODS 9: Industria, innovación e infraestructura**, promoviendo el avance de tecnologías aplicables a la automoción y la robótica.
- **ODS 12: Producción y consumo responsables**, por el uso de *software* libre y recursos compartidos.
- **ODS 13: Acción por el clima**, al contribuir a una movilidad más sostenible y eficiente.

Este trabajo no sólo persigue un objetivo académico, sino que tiene un impacto transversal en la sociedad, la economía, el medio ambiente y la industria, alineándose con las prioridades globales establecidas en la Agenda 2030.

Capítulo 8

Conclusiones y Lineas Futuras

8.1. Conclusiones

El desarrollo de este Trabajo Fin de Grado ha permitido implementar y validar un sistema de percepción basado exclusivamente en tecnología *LiDAR*, capaz de detectar y clasificar los conos que delimitan el circuito en la modalidad *Driverless* de *Formula Student*. El proyecto se concibió con un doble propósito: por un lado, dotar al vehículo autónomo del equipo *UPM Racing* de un subsistema robusto y eficiente, y por otro, explorar los límites de un enfoque *LiDAR-only*, sin depender de arquitecturas multisensor más frágiles y complejas.

Desde el punto de vista del desarrollo, se ha diseñado un *pipeline* modular organizado en cuatro etapas principales: preprocesamiento de la nube de puntos, segmentación del plano del suelo, *clustering* de objetos y clasificación de conos. La implementación bajo ROS 2 y el uso de parámetros dinámicos han permitido ajustar y validar cada módulo de manera independiente, favoreciendo la depuración y la integración con el resto del *stack* autónomo del vehículo. El diseño modular ha demostrado ser un factor clave para acelerar la experimentación, comparar alternativas algorítmicas y reutilizar componentes en distintas configuraciones.

En cuanto a la mejora algorítmica, el trabajo ha introducido varias aportaciones relevantes:

- En la etapa de **preprocesamiento**, el filtrado mediante una *bounding box* y la rotación condicional de coordenadas han permitido reducir de manera temprana el número de puntos irrelevantes, disminuyendo el coste computacional en fases posteriores.
- En la **segmentación del suelo**, el uso de RANSAC con parámetros optimizados ha eliminado superficies planas con un equilibrio adecuado entre precisión y velocidad, ajustándose al límite temporal de 100 ms impuesto por la frecuencia del sensor.
- En el **clustering**, la integración de DBSCAN junto con heurísticas basadas en la geometría de los objetos ha permitido discriminar de manera fiable estructuras de ruido respecto a conos válidos, mejorando la coherencia de los grupos detectados y reduciendo prácticamente a cero los falsos positivos en condiciones reales.
- En la **clasificación de color**, la transición desde reglas heurísticas a modelos de *machine learning* ha supuesto un salto cualitativo.

La fase de **validación** se ha realizado en condiciones reales de pista, utilizando datasets registrados en pruebas en la pista de ensayos del INSIA y en los entornos de competición de FSUK. Los resultados muestran mejoras muy significativas respecto al sistema original:

- El tiempo máximo de procesamiento por *frame* se redujo de **604 ms** en el sistema original a tan solo **93 ms** con el nuevo *pipeline*, garantizando el cumplimiento de la restricción temporal del LiDAR (100 ms).
- El número de falsos positivos se redujo en un **99,994 %**, eliminando prácticamente todas las detecciones erróneas que anteriormente saturaban el sistema de planificación.

- La clasificación de color alcanzó un rendimiento consistente, con una precisión global del **99,8 %**, asegurando la correcta diferenciación entre conos azules y amarillos incluso en condiciones adversas, lo cual es crítico para la interpretación del trazado.

Estos resultados no solo superan con creces los objetivos definidos al inicio del proyecto, sino que además demuestran la viabilidad de un sistema *LiDAR-only* en entornos de competición. Su impacto ha sido tangible en el rendimiento del equipo *UPM Racing*, contribuyendo directamente los puestos destacados logrados en las dos últimas ediciones de *Formula Student UK* (4º y 6º puesto en Silverstone) [48]. Asimismo, el proyecto ha generado una base sólida y documentada que servirá como punto de partida para futuros desarrollos del equipo y para líneas de investigación adicionales.

En síntesis, las principales **conclusiones** que pueden extraerse son:

1. Es posible sustituir una arquitectura multisensor por un sistema basado únicamente en LiDAR sin comprometer la fiabilidad, siempre que se implementen algoritmos robustos de segmentación, *clustering* y clasificación.
2. La modularidad en el diseño del *pipeline*, junto con el uso de ROS 2, constituye una estrategia altamente efectiva para proyectos de percepción en tiempo real.
3. La aplicación de técnicas de *machine learning* aporta una mejora sustancial frente a enfoques heurísticos, especialmente en tareas sensibles como la clasificación de color.
4. El sistema desarrollado no solo cumple con los requisitos de competición, sino que también representa una aportación extrapolable a otros contextos de conducción autónoma, donde se requiera un sistema de percepción robusto, eficiente y de bajo coste en sensores.

Finalmente, conviene señalar que el sistema mantiene ciertas limitaciones. Su fiabilidad depende de que cada cono quede registrado por al menos tres anillos, pues con menos la densidad de puntos y la información de reflectividad son insuficientes para evitar falsos positivos o errores en la clasificación de color. Este requisito implica que, a mayores distancias o con ligeras variaciones en la inclinación del sensor, pueda perderse capacidad de detección fiable. Aunque el umbral adoptado asegura un funcionamiento robusto en la mayoría de escenarios, constituye una restricción relevante a tener en cuenta para futuras mejoras.

8.2. Líneas futuras

A partir de los resultados obtenidos, se identifican varias líneas de trabajo que permitirían seguir evolucionando el sistema de percepción desarrollado:

- **Optimización de la eficiencia computacional:**

Migrar las etapas más costosas del *pipeline* (*clustering*, RANSAC, clasificación) a implementaciones en C++ o CUDA, con el fin de reducir aún más la latencia y garantizar su ejecución en tiempo real en cualquier escenario de carga.

- **Acumulación de nubes de puntos:**

Combinar varios escaneos consecutivos para obtener más muestras por cono y ganar robustez frente a oclusiones temporales, mejorando así la detección a larga distancia y la clasificación, siempre que la alineación y la redundancia se gestionen sin comprometer el tiempo real.

- **Mejora de la clasificación de objetos:**

Explorar modelos de aprendizaje profundo específicos para nubes de puntos, con el objetivo de aumentar la capacidad de caracterizar los conos con mayor detalle y robustez. De esta forma se podría mejorar frente a condiciones adversas como baja reflectividad, ruido del sensor o variaciones en la iluminación, ampliando el margen de fiabilidad del sistema.

- **Fusión sensorial opcional:**

Reintroducir el sistema de cámaras como fuente complementaria de información, manteniendo la arquitectura *LiDAR-only* como núcleo redundante para garantizar un funcionamiento seguro en caso de fallo de otros sensores.

- **Aprendizaje en línea y adaptación:**

Investigar clasificadores capaces de recalibrarse durante la competición, ajustándose dinámicamente a variaciones en el entorno o incluso al tiempo de ejecución.

- **Validación a gran escala:**

Realizar campañas de prueba con un mayor volumen de datos y circuitos diferentes, lo que permitiría consolidar métricas de precisión, latencia y fiabilidad en condiciones variadas.

- **Evaluación global del sistema:**

Analizar el impacto del módulo de percepción en el rendimiento conjunto del vehículo, considerando métricas de navegación, estabilidad y seguridad, de forma que los resultados se interpreten en el contexto de todo el *stack* autónomo y no únicamente de manera aislada.

Bibliografía

- [1] Mercedes-Benz Group. Benz Patent Motor Car: The first automobile (1885–1886). [Online]. Available: <https://group.mercedes-benz.com/company/tradition/company-history/1885-1886.html>
- [2] Ford Corporate. The Moving Assembly Line. [Online]. Available: <https://corporate.ford.com/articles/history/moving-assembly-line.html>
- [3] K.-B. AG. (1930) Brake force regulators, in particular for motor vehicles. Patent DE492199C, 20 de febrero de 1930. [Online]. Available: <https://patents.google.com/patent/DE492199C/en>
- [4] F. Sabry, *Traffic Sign Recognition: Unlocking the Power of Computer Vision*. One Billion Knowledgeable, 2025.
- [5] D. E. Case, “Formula SAE - Competition History 1981 - 1996,” SAE International, Warrendale, PA, SAE Technical Paper 962509, Dec. 1996.
- [6] UPM Racing – #WeLoveWhatWeDo. [Online]. Available: <https://upm-racing.es/>
- [7] Universidad Politécnica de Madrid. [Online]. Available: <https://www.upm.es/>
- [8] (2017) FSG: Autonomous Driving at Formula Student Germany 2017. [Online]. Available: <https://www.formulastudent.de/pr/news/details/article/autonomous-driving-at-formula-student-germany-2017/>
- [9] Ultralytics. (2025) YOLOv9: Un salto adelante en la tecnología de detección de objetos. [Online]. Available: <https://docs.ultralytics.com/es/models/yolov9/>
- [10] C. Ortega, “Sistema de Localización y Mapeo Simultáneo de vehículos de carreras autónomos mediante filtros recursivos,” 2025.
- [11] Ouster, Inc., *OS1 Mid-Range High-Resolution Imaging LiDAR: Datasheet* ‘ç, Ouster, Inc., San Francisco, CA, 2021, hardware Version: 840-102145-05 (Rev 05), Firmware Version: v2.1.x. [Online]. Available: <https://ouster.com>
- [12] StereoLabs. (2025) ZED 2i – Industrial AI Stereo Camera. [Online]. Available: <https://www.stereolabs.com/store/products/zed-2i>
- [13] Xsens / Movella. (2025) MTi-680 RTK GNSS/INS Module. [Online]. Available: <https://www.xsens.com/products/sensor-modules/xsens-mti-680-rtk-gnss-ins>
- [14] Open Robotics / ROS 2 Documentation. (2025) ROS 2 Documentation — Humble Hawksbill. [Online]. Available: <https://docs.ros.org/en/humble/index.html>
- [15] Edinburgh University Formula Student. (2025) eufs_sim: Open-Source Formula Student Autonomous Vehicle Simulation. Repositorio movido a GitLab, ver https://gitlab.com/eufs/eufs_sim. [Online]. Available: https://github.com/eufsa/eufs_sim
- [16] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, “A systematic review of perception system and simulators for autonomous vehicles research,” *Sensors*, vol. 19, no. 3, p. 648, 2019.
- [17] M. Yahyaei, G. Seifert, T. Hempen, and W. Huber, “Review of exteroceptive sensors for autonomous driving,” in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 4005–4010.
- [18] Z. Chen and X. Huang, “Pedestrian detection for autonomous vehicle using multi-spectral cameras,” *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 211–219, 2019.

- [19] H. J. Trussell and M. J. Vrhel, *Fundamentals of digital imaging*. Cambridge University Press, 2008.
- [20] K. Sadekar. Stereo camera depth estimation with OpenCV (python/c++). [Online]. Available: <https://learnopencv.com/depth-perception-using-stereo-camera-python-c/>
- [21] J. Ma, S. Zhuo, L. Qiu, Y. Gao, Y. Wu, M. Zhong, R. Bai, M. Sun, and P. Y. Chiang, “A Review of ToF-Based LiDAR,” *Journal of Semiconductors*, vol. 45, no. 10, p. 101201, 2024. [Online]. Available: <https://doi.org/10.1088/1674-4926/24040015>
- [22] P. Pendse, “An Overview on LiDAR for Autonomous Vehicles,” 04 2024.
- [23] C. Wolff. Fundamentos de radar - radar FMCW. Publisher: Dipl.-Ing. (FH) Christian Wolff. [Online]. Available: <https://www.radartutorial.eu/02.basics/rp08.es.html>
- [24] H. Balta, J. Velagic, W. Bosschaerts, G. De Cubber, and B. Siciliano, “Fast statistical outlier removal based method for large 3d point clouds of outdoor environments,” *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 348–353, 2018.
- [25] P. Szutor and M. Zichar, “Fast radius outlier filter variant for large point clouds,” *Data*, vol. 8, no. 10, p. 149, 2023.
- [26] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang, “Voxel based object simplification,” in *Proceedings Visualization’95*. IEEE, 1995, pp. 296–303.
- [27] W. Lyu, W. Ke, H. Sheng, X. Ma, and H. Zhang, “Dynamic Downsampling Algorithm for 3D Point Cloud Map Based on Voxel Filtering,” *Applied Sciences*, vol. 14, no. 8, 2024. [Online]. Available: <https://www.mdpi.com/2076-3417/14/8/3160>
- [28] B. Wang, J. Lan, and J. Gao, “LiDAR filtering in 3D object detection based on improved RANSAC,” *Remote Sensing*, vol. 14, no. 9, p. 2110, 2022.
- [29] A. Khaloo and D. Lattanzi, “Robust normal estimation and region growing segmentation of infrastructure 3d point cloud models,” *Advanced engineering informatics*, vol. 34, pp. 1–16, 2017.
- [30] Y. Cao, Y. Wang, Y. Xue, H. Zhang, and Y. Lao, “FEC: Fast Euclidean clustering for point cloud segmentation,” *Drones*, vol. 6, no. 11, p. 325, 2022.
- [31] B.-Q. Shi, J. Liang, and Q. Liu, “Adaptive simplification of point cloud using k-means clustering,” *Computer-Aided Design*, vol. 43, no. 8, pp. 910–922, 2011.
- [32] M. El Yabroudi, K. Awedat, R. C. Chabaan, O. Abudayyeh, and I. Abdel-Qader, “Adaptive DBSCAN LiDAR point cloud clustering for autonomous driving applications,” in *2022 IEEE International Conference on Electro Information Technology (eIT)*. IEEE, 2022, pp. 221–224.
- [33] C. Feng, Y. Taguchi, and V. R. Kamat, “Fast plane extraction in organized point clouds using agglomerative hierarchical clustering,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6218–6225.
- [34] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, “Improvements to platt’s smo algorithm for svm classifier design,” *Neural computation*, vol. 13, no. 3, pp. 637–649, 2001.
- [35] M. Pal, “Random forest classifier for remote sensing classification,” *International journal of remote sensing*, vol. 26, no. 1, pp. 217–222, 2005.
- [36] Interactive Chaos, “Índice Gini,” Tutorial de Machine Learning, Interactive Chaos, 2025. [Online]. Available: <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/indice-gini>

- [37] scikit-learn developers. (2025) Oob errors for random forests — scikit-learn example. [Online]. Available: https://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html
- [38] P. Cunningham and S. J. Delany, “K-nearest neighbour classifiers-a tutorial,” *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.
- [39] J. Huang and S. You, “Point cloud labeling using 3d convolutional neural network,” in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2670–2675.
- [40] “Pointnet: Deep learning on point sets for 3d classification and segmentation’.”
- [41] W. Shi and R. Rajkumar, “Point-GNN: Graph neural network for 3d object detection in a point cloud,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1711–1719.
- [42] Formula Student Germany, “Results 2024,” <https://www.formulastudent.de/fsg/results/2024>, 2024, accedido: 2025-09-08.
- [43] T. D. Driverless, “Beating the human driver,” Conferencia en FSG Academy, Nov. 2024, disponible en: <https://www.formulastudent.de/academy/20241109-siemens/>.
- [44] Livox Technology, *Livox HAP LiDAR Sensor*, Livox, 2025, Primera LiDAR automotriz para producción en serie. Tests de confiabilidad automotriz y producción masiva desde 2021. [Online]. Available: <https://www.livoxtech.com/hap>
- [45] ros-drivers, “Ros2 ouster drivers (`ros2_ouster_drivers`),” https://github.com/ros-drivers/ros2_ouster_drivers, 2024, gitHub repository, tag 0.4.3 (commit b57d567), Apache-2.0.
- [46] G. José R., “La rotación de Rodrigues,” vol. 2, pp. 82–87, 04 2022.
- [47] SciPy Developers, *scipy.interpolate.griddata*, SciPy, 2025, disponible en: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html> [Accedido: 31-ago-2025].
- [48] Institution of Mechanical Engineers, “Formula Student – Previous Events,” <https://www.imeche.org/events/formula-student/previous-events>, 2025, accessed: 2025-09-08.

Anexo A

A.1. Información recogida en las pruebas

Campo	Descripción
frame_id	Identificador del <i>frame</i> procesado.
cone_id	Identificador único asignado a cada cono detectado.
status	Estado de la detección (p.ej. válido / falso positivo).
color	Color estimado del cono (azul, amarillo, naranja, desconocido).
n_points	Número de puntos LiDAR asociados al cono.
n_rings	Número de anillos distintos presentes en el cono.
x, y, z	Coordenadas estimadas del cono en el sistema de referencia del sensor.
t_pre_ms	Tiempo de ejecución de la etapa de preprocesado (ms).
t_seg_ms	Tiempo de ejecución de la etapa de segmentación (ms).
t_clu_ms	Tiempo de ejecución de la etapa de clustering (ms).
t_cla_ms	Tiempo de ejecución de la etapa de clasificación de color (ms).
cloud_points	Número de puntos de la nube original recibida del sensor.
pre_points	Número de puntos tras la etapa de preprocesado.
inliers	Número de puntos clasificados como suelo en segmentación.
outliers	Número de puntos restantes tras eliminar el suelo.
clusters	Número de agrupaciones detectadas antes del filtrado final.
n_cones_total	Número total de conos dispuestos en la pista.

Tabla A.1: Campos registrados durante las pruebas cuantitativas del sistema de percepción.