CEA Comité Español de Automática

XLVI JORNADAS AUTOMÁTICA CARTAGENA 2025

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Simulador para validación de funcionalidades con vehículos automatizados de competición

Oroz Zubasti, Miguel[a,*], Colmenero Fernández, Alberto[a], Pérez Rastelli, Joshué[b]

[a] *Universidad de Navarra, Tecnun eRacing, Manuel Lardizabal 13, 20018 Donostia / San Sebastián, Spain*
[b] *CEIT-Basque Research and Technology Alliance (BRTA), Manuel Lardizabal 15, 20018 Donostia / San Sebastián, Spain.*
Universidad de Navarra, Tecnun, Manuel Lardizabal 13, 20018 Donostia / San Sebastián, Spain.

To cite this article: Oroz Zubasti, Miguel, Colmenero Fernández, Alberto, Pérez Rastelli, Joshué. 2025. Simulator for Functionality Validation in Automated Racing Vehicles. Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12075

Resumen

Los vehículos automatizados representan una de las áreas más innovadoras y desafiantes dentro del ámbito de la automoción y la automática. Sus aplicaciones van desde el transporte urbano, carreteras y hasta competiciones de alto rendimiento. En este contexto, la validación virtual de funcionalidades se ha convertido en una herramienta clave para acelerar el desarrollo y validar estos sistemas antes de probar en plataformas reales. En este trabajo se presenta MirenaSim, una implementación de un simulador de lazo completo de código abierto M. Oroz and A.Colmenero (2025) , desarrollado bajo licencia MIT, como herramienta de validación para el vehículo autónomo de competición de Tecnun eRacing. El simulador permite probar y depurar funcionalidades críticas del sistema autónomo, como la planificación de trayectorias, el control del vehículo y la percepción del entorno, en un entorno virtual seguro y reproducible. Gracias a esta herramienta, el equipo ha logrado avances significativos en el desarrollo y la fiabilidad del sistema autónomo, reduciendo tiempos de prueba y mejorando la calidad de algunas de las funcionalidades más importantes.

Palabras clave: Vehículos autónomos, Simulación, Integración de sensores y percepción, Sistemas de transporte inteligentes.

Simulator for Functionality Validation in Automated Racing Vehicles

Abstract

Automated vehicles represent one of the most innovative and challenging areas within the fields of automotive engineering and automation. Their applications range from urban transportation and highways to high-performance competitions. In this context, virtual validation of functionalities has become a key tool to accelerate development and validate these systems before testing them on real platforms.This work presents MirenaSim, the implementation of a full-loop open-source simulator M. Oroz and A.Colmenero (2025), developed under the MIT license, as a validation tool for Tecnun eRacing's autonomous racing vehicle. The simulator enables testing and debugging of critical functionalities of the autonomous system—such as trajectory planning, vehicle control, and environment perception—in a safe and reproducible virtual environment. Thanks to this tool, the team has achieved significant progress in the development and reliability of the autonomous system, reducing testing time and improving the quality of some of its most important functionalities.

Keywords: Autonomous Vehicles, Simulation, Sensor integration and perception, Intelligent transportation systems.

*Autor para correspondencia: morozzubast@alumni.unav.es

1. Introducción

La simulación aplicada a vehículos autónomos y automatizados ha experimentado una evolución notable desde sus primeras etapas, consolidándose como una herramienta fundamental en el desarrollo y validación de nuevas funcionalidades en entornos virtuales. Los primeros simuladores comenzaron a desarrollarse en las décadas de 1980 y 1990, en paralelo con los avances iniciales en inteligencia artificial y sistemas de control, que empezaban a aplicarse al ámbito automotriz. Desde entonces, la simulación ha pasado de ser una herramienta experimental a convertirse en un componente esencial para el diseño, prueba y optimización de algoritmos de conducción autónoma, permitiendo evaluar el comportamiento del vehículo en escenarios complejos.

Un hito clave en este proceso fue el desafío DARPA en 2004, que aceleró el desarrollo de algoritmos de navegación autónoma y la simulación de entornos complejos Newman (2007). En las dos décadas siguientes, tanto la industria como el ámbito académico han experimentado avances notables. Empresas como Waymo, Tesla y Uber han desarrollado simuladores avanzados que permiten entrenar vehículos autónomos en entornos virtuales, facilitando la validación de sistemas en una amplia variedad de escenarios sin necesidad de realizar costosas pruebas en el mundo real.

En este contexto, las competiciones de Formula Student (FSD) están consideradas entre las más importantes para estudiantes de escuelas de ingeniería a nivel universitario. Equipos de toda Europa diseñan, construyen y compiten con monoplazas tipo fórmula, que son evaluados mediante pruebas estáticas y dinámicas.

Desde su expansión en 1998 con la creación de Formula Student UK, y el posterior auge de eventos como Formula Student Germany, la competición se ha consolidado como un referente en innovación, formación práctica y excelencia técnica dentro del ámbito automotriz.

En 2017 se añadió la categoría de Conducción Autónoma, complementando a las existentes combustión y electrico. Desde entonces muchos equipos de estudiantes se han lanzado a robotizar sus vehículos existentes para que sean capaces de realizar las pruebas tradicionales de la competición sin necesidad de piloto. En este trabajo se presenta un simulador de código abierto, diseñado para validar funcionalidades críticas del vehículo autónomo de competición de Tecnun eRacing. La herramienta permite probar planificación, control y percepción en un entorno virtual seguro. Su uso ha mejorado la eficiencia del desarrollo y la fiabilidad del sistema autónomo.

El desarrollo de este tipo de sistemas se ve limitado, en muchos casos, por la imposibilidad de disponer de acceso exclusivo al vehículo, que se rediseña cada año, y por los riesgos asociados a probar funcionalidades autónomas directamente sobre el vehículo de competición. Por ello, muchos equipos han optado por desarrollar sus propios simuladores de uso interno, que les permiten validar los sistemas de percepción, planificación y control mediante un gemelo virtual del vehículo.

Simuladores prexistentes como CARLA Dosovitskiy et al. (2017), de uso más general en sistemas autónomos, son demasiado pesados y complejos para nuestro caso de uso. Otros simuladores específicos para las competiciones de FSD como FSDS Community (2021), o eufs Student (2023), carecen de capacidades de simulación completas y no ofrecen la flexibilidad necesaria para nuestro trabajo, especialmente en contextos donde no se dispone de hardware real. En la Tabla 1 se presenta una comparativa de las características más relevantes de algunos de los simuladores del estado del arte. Dado que en la mayoría de los casos las licencias restrictivas o las limitaciones de hardware representaban un obstáculo para su adopción, se optó por desarrollar el simulador que se presenta en este artículo, adaptado específicamente a las necesidades del equipo y con un enfoque en la flexibilidad y accesibilidad.

Tabla 1: Comparación abreviada de simuladores FSD

Sim	Enfoque	Motor	ROS	Lic.
FSDS	Realismo FSD	Unreal	Sí	OSS
eufs	Ligero ROS2	Gazebo	Sí	OSS
CARLA	Autónomos	Unreal	Sí	MIT
Cogn.	ADAS/FSD ind.	Prop.	Parc.	Com.
CarSim	Dinámica veh.	Prop.	Parc.	Com.

El resto del artículo se divide como sigue. En la sección 2 se dará un resumen técnico de las distintas funcionalidades y detalles de la implementación. Posteriormente, en la sección 3 se expondrán algunos casos de uso reales que han sido validados por el sistema propuesta. Finalmente, se presentan las conclusiones en la sección 4, las cuales están obtenidas a partir del trabajo en el simulador.

2. Simulador

El simulador está diseñado específicamente para vehículos automatizados y proporciona una plataforma robusta para el desarrollo y la validación de sistemas autónomos. Su objetivo principal es ofrecer un entorno de simulación realista, integrando una amplia variedad de sensores virtuales (como cámaras, LiDAR y GPS) que permiten replicar con precisión las condiciones del mundo real.

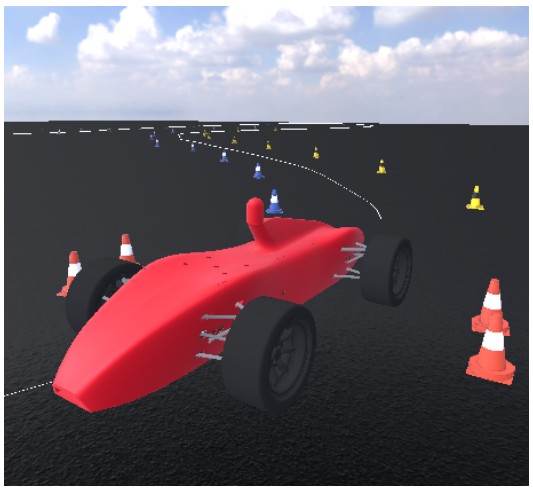

Figura 1: Vista del Simulador con un vehículo y conos

Entre sus ventajas destacan la flexibilidad en la instalación y configuración, la cual se realiza mediante las herramientas colcon y ROS 2 Macenski et al. (2022), junto con la integración de Godot Community (2014) para una simulación preci-

sa. Además, el simulador cuenta con comandos y controles intuitivos para cargar pistas, cambiar modos de control y personalizar el entorno, permitiendo así modificaciones detalladas que se adaptan a las necesidades específicas de cada proyecto.

2.1. *Implementación*

Todo simulador necesita integrar un motor físico que permita representar el comportamiento dinámico del entorno y del vehículo; otros simuladores, como CARLA o FSSIM Racing (2018), hacen uso de motores de videojuegos como Unreal Engine. En nuestro caso, hemos decidido utilizar Godot Community (2014), un motor de código abierto desarrollado en 2014 y que ofrece una API de programación en C++ que permite la integración con el entorno de ROS 2. El simulador implementa una serie de sensores y actuadores virtuales sobre las físicas del módulo de simulación de vehículos que el propio motor provee. Además, ofrece la posibilidad de cargar pistas generadas y generar datos sintéticos para el entrenamiento de sistemas de aprendizaje profundo. Todo esto es convenientemente distribuido como un paquete de ROS 2. La Figura 1 muestra una imagen del simulador con un modelo del coche.

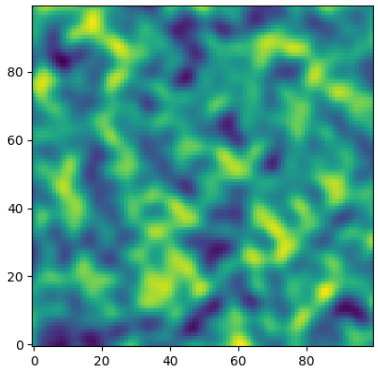

Figura 2: Mapa de ruido

2.2. *Generación de Mapas*

Generar circuitos de automoción proceduralmente consiste esencialmente en generar una curva cerrada simple de clase C^1. Bajo la idea de que una pista de carreras adopta su forma debido a su necesidad en origen de adaptarse a la orografía del terreno, el problema se simplifica de manera notable si generamos un mapa bidimensional de ruido Simplex Perlin (2002), como el que se muestra en la Figura 2.

A partir de este mapa realizamos un corte de nivel y extraemos todos los contornos mediante un algoritmo *Marching Squares*, como se muestra en Razavikia et al. (2020), analizando los contornos y descartando aquellos con pocos puntos o curvaturas muy elevadas (por ejemplo más de 0,5 m^{-1}). Elegimos uno de los restantes aleatoriamente, como el que se muestra en la figura 3.

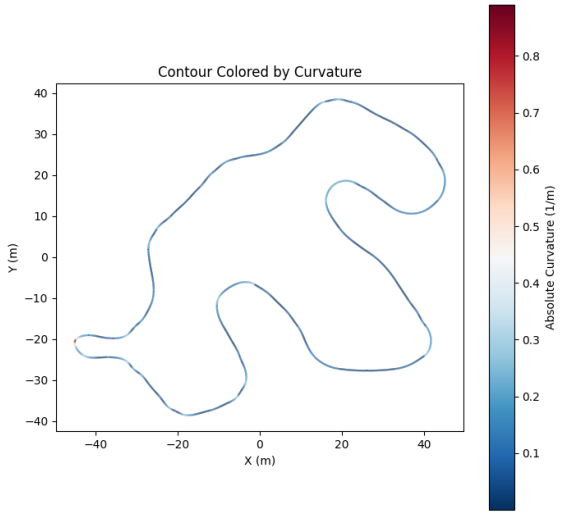

Figura 3: Análisis de suavidad del trazado

Tomando esa curva y aplicándole una transformación lineal para que se ajuste al área de la pista, se procede a interpolar mediante una *B-Spline* Saillot et al. (2024), sobre dicha curva se recorre integrando numéricamente el camino y generando las puertas de conos a la distancia requerida. La pista y su curva generatriz son exportadas en un formato json estándar para su uso en simulador.

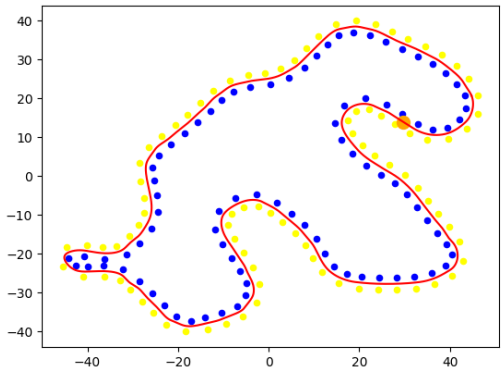

Figura 4: Puertas generadas

2.3. *Interfaz y Uso*

Al usar Godot Community (2014) la ventaja radica en que podemos implementar los sensores como nuevos nodos del motor, por medio de una librería dinámica disponemos de todos los sensores que implementa este simulador que podremos posicionar sobre cualquier objeto o instancia del juego. Godot utiliza un lenguaje de scripting interno llamado GDscript, un lenguaje semejante a python con tipado dinámico, del que podemos programar de manera sencilla nuestro caso de uso que es un vehículo. Sin embargo, estos sensores pueden ser incorporados en cualquier otro tipo de simulador.

Una vez que se ejecuta el simulador, se podrá entrar en el editor para ajustar distintas configuraciones de los sensores como su posición, frecuencia de envío, resolución, entre otros. El simulador se encarga automáticamente de convertir entre sistemas de coordenadas y enviar las posiciones de los nodos que implementa mediante **TF2** Quigley et al. (2025).

El simulador también ofrece la posibilidad de lanzarlo con una serie de argumentos que permiten la carga de pistas o la generación automática de datasets, facilitando así su uso en sistemas de integración continua.

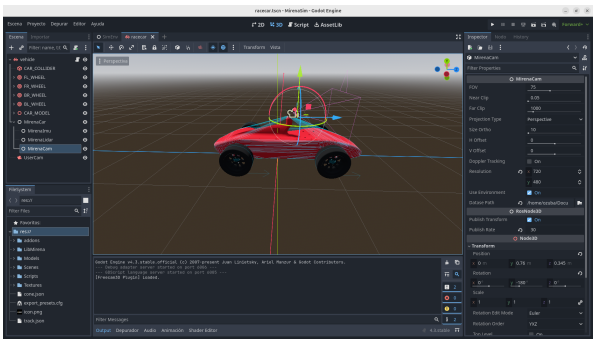

Figura 5: Editor del Simulador

2.4. *Sensores*

El simulador implementa los distintos sensores que tradicionalmente se encuentra en un vehículo autónomo, y se describen a continuación.

2.4.1. *LIDAR*

El simulador implementa un sensor LiDAR usando la api de *raycasting* que godot expone. Se parte de un punto foco situado en el centro del sensor, sobre este se calcula una dirección $\vec{d_{\phi,\theta}}$ definida por los valores de azimut 1 y elevación 2 para cada punto $(i, j) \in \mathbb{N}^2$ teniendo en cuenta la densidad angular horizontal y vertical de puntos (h_{step}, v_{step}),la cantidad de capas v_{res} y resolución horizontal h_{res}, sobre ella se define un punto final 4 en base a la distancia máxima de detección *max_range* desde el centro del sensor $\vec{origin}$, el motor se encargará de devolvernos la colisión con el primer punto entre ambos por medio de la función *ray_query* con el primer solido que encuentre a la que añadiremos un ruido gausiano 6 para simular un entorno real de medición:

$$\phi_i = (h_{\text{step}} \cdot i - \frac{h_{res}}{2}) \cdot \frac{\pi}{180} \tag{1}$$

$$\theta_j = (v_{\text{step}} \cdot j - \frac{v_{res}}{2}) \cdot \frac{\pi}{180} \tag{2}$$

$$\vec{d_{\phi,\theta}} = (\cos(\theta) \cdot \sin(\phi),\ \sin(\theta),\ \cos(\theta) \cdot \cos(\phi)) \tag{3}$$

$$\vec{to} = \vec{origin} + \vec{d_{\phi,\theta}} \cdot \text{max\_range} \tag{4}$$

$$\text{ray\_query(from, to)} = \left(\vec{origin},\ \vec{to}\right) \tag{5}$$

$$\vec{w} = \mathcal{N}(0, \sigma^2) \cdot \vec{d_{\phi,\theta}} \tag{6}$$

$$\text{Hitpoint} = \text{intersect\_ray(ray\_query)} + \vec{w} \tag{7}$$

El motor nos devolverá el primer punto de intersección entre ambos puntos o el rango máximo si este no existe. El coste computacional de esta operación puede ser elevado si intentamos llegar al numero de puntos que nos ofrece un **LiDAR** comercial. Al realizarse este paso con el motor de físicas congelado, podemos optimizarlo fácilmente haciendo uso de **OpenMP** paralizando el escaneo en distintos hilos. Al resultado final le añadiremos un ruido gaussiano blanco aditivo para simular el ruido real.

2.4.2. *IMU*

La unidad de medida inercial deriva numéricamente los valores de posición a lo largo del tiempo y aplica una transformación linear para obtener las componentes de aceleración y velocidad angular sobre el marco de coordenadas del vehículo.

2.4.3. *Cámara*

Para la cámara se utiliza el sistema de cámaras de godot que permite renderizar los viewports con la resolución y deformación de lente requeridas, para posteriormente convertir el formato a uno que ROS 2 soporta de manera estandar.

2.4.4. *GNSS*

El GNSS hace uso del sistema de coordenadas del motor y utiliza una proyección geodésica para convertir las coordenadas cartesianas (*engine_x*, *engine_y*,*engine_z*) en latitud y longitud, a partir de las coordenadas internas del motor, las ecuaciones 8 y 9 computan la proyección de las coordenadas internas del motor sobre un plano en la superficie terrestre, se obtienen las coordenadas finales (*origin*,*origin*,*origin*) 10. por medio de un desfase de origen (*origin_latitude*,*origin_longitude*,*origin_altitude*).

$$\Delta\text{lat} = \left(\frac{\text{engine\_x}}{R_{\text{earth}}}\right) \times \left(\frac{180{,}0}{\pi}\right) \tag{8}$$

$$\Delta\text{lon} = \left(\frac{\text{engine\_y}}{R_{\text{earth}} \cdot \cos(\text{origin\_latitude} \cdot \frac{\pi}{180})}\right) \times \left(\frac{180{,}0}{\pi}\right) \tag{9}$$

$$\begin{pmatrix} \text{latitude} \\ \text{longitude} \\ \text{altitude} \end{pmatrix} = \begin{pmatrix} \text{origin\_latitude} \\ \text{origin\_longitude} \\ \text{origin\_altitude} \end{pmatrix} + \begin{pmatrix} \Delta\text{lat} \\ \Delta\text{lon} \\ \text{engine\_z} \end{pmatrix} \tag{10}$$

2.4.5. *WSS*

El sensor de velocidad en rueda simplemente hace broadcast de la velocidad de las ruedas del vehículo que tiene en cuenta el motor de físicas.

2.5. *Datos sintéticos*

El simulador permite en estos momentos generar 2 tipos de datasets sintéticos:

2.5.1. YoloV8 Dataset

Durante la conducción se almacenará en la carpeta del editor un par de ficheros, una imagen y sus correspondientes anotaciones de *Bounding Boxes* en formato YOLO8 Ultralytics (2023), como se muestra en la Figura 6

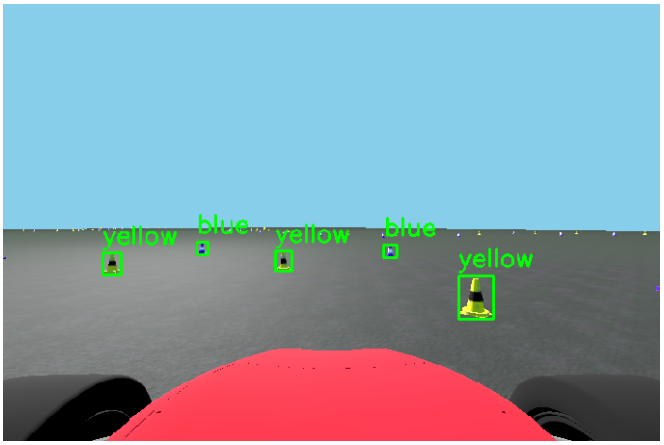

Figura 6: Anotaciones generadas en YoloV8

Esto es posible ya que el simulador es capaz de realizar una proyección de perspectiva de los vértices de la AABB de cada obstáculo sobre el plano de la cámara.

2.5.2. Keypoints Dataset

Además, durante la conducción se puede añadir todos los conos disponibles en pantalla con sus respectivos ficheros de posiciones de los puntos clave dentro del cono, como se muestra en la Figura 7. En la carreras de Formula Student, los conos son importantes para delimitar el circuito a seguir.

Figura 7: Puntos clave del cono

3. Casos de uso

Uno de los primeros retos del desarrollo de un sistema de conducción autónoma es la percepción, para lo cual el uso de este simulador personalizado ha permitido un avance rápido y con pocos medios (sin vehículo real).

3.1. Localizador y Clasificador

El primer problema a resolver es segmentar y clasificar los obstáculos de la pista (Conos). El desarrollo reciente de redes convolucionales como **YOLOV8** Ultralytics (2023) permite realizar esta tarea con un coste computacional mucho menor que otros sistemas clásicos. Se presenta la Tabla 2 con algunas de las redes populares evaluadas en terminos de fotogramas por segundo y la media del promedio de precisión (**mAP**). Como puede observarse los modelos de tipo **YOLO** lideran la eficiencia computacional.

Tabla 2: Comparativa de modelos de deep learning para clasificación y detección

Modelo	mAP@0.5	FPS
YOLOv8 Ultralytics (2023)	53.9	100+
YOLOv5 Ultralytics (2020)	50.1	80–100
SSD (MobileNet)	23.2	50–60
Faster R-CNN	42.0	5–10
EfficientDet-D0	33.8	30–35
ResNet-50	76.0	–
EfficientNet-B0	77.1	–

Gracias al set de datos sintéticos es posible entrenar este modelo con buena fidelidad en poco tiempo. La fidelidad visual del simulador permite que la red de inferencia funcione también sobre imágenes reales de datasets públicos como FSOCO FSOCO Team (2021).

3.2. Filtrado y detección LIDAR

El módulo LiDAR implementa un pipeline basado en la librería PCL Rusu and Cousins (2011) para la detección de conos a partir de una nube de puntos 3D. El proceso comienza con la eliminación de *outliers* mediante un filtro esférico basado en distancia. Posteriormente, se elimina el plano del suelo aplicando segmentación RANSAC Fischler and Bolles (1981). La nube filtrada es entonces segmentada en clústeres mediante agrupamiento euclidiano. Finalmente, se computan los centroides de cada clúster y se clasifican como conos con alta confianza, asumiendo una única clase. Esta aproximación permite identificar obstáculos relevantes para la navegación autónoma sobre pista, como se ve en la Figura 8.

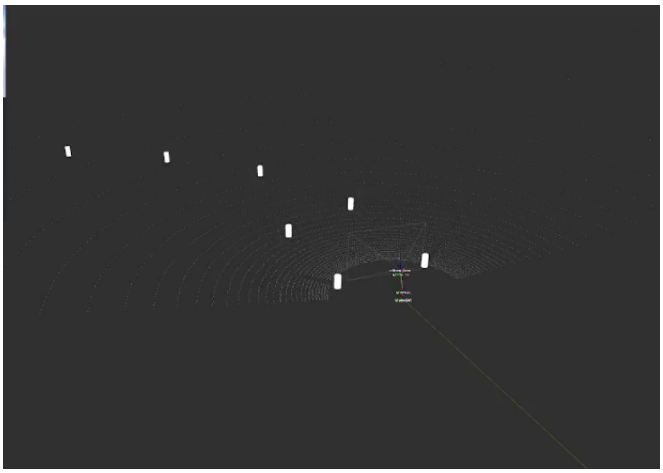

Figura 8: Conos detectados por el LiDAR

4. Conclusiones

El desarrollo de un simulador propio ha sido clave para lograr avances significativos durante el primer año del proyecto de conducción autónoma del equipo de formula de TECNUN. La posibilidad de personalizar la herramienta frente a otras alternativas existentes nos ha proporcionado la flexibilidad necesaria para adaptarla de forma eficiente a nuestros casos de uso específicos. Además, el uso de un motor de videojuegos como base, junto con la implementación de sensores virtuales, no solo ha facilitado la validación del sistema en un entorno digital, sino que también ha contribuido a una comprensión más profunda del funcionamiento de dichos sensores. Esta plataforma ha permitido validar de forma segura y controlada gran parte del desarrollo posterior, acelerando el proceso de integración y prueba del sistema autónomo en el futuro.

Agradecimientos

A la Universidad de Navarra por permitir estos espacios de trabajo e investigación al alumnado, a mis compañeros de equipo Tecnun eRacing por acompañar e inspirar este desarrollo. A todos los contribuidores del proyecto Godot que de manera desinteresada crean una herramienta tan versatil. Financiado en parte por por el proyecto iEXODDUS, bajo el Grant 101146091.

Referencias

Community, F.-D., 2021. Formula student driverless simulator (fsds). https://github.com/FS-Driverless/Formula-Student-Driverless-Simulator, accedido el 5 de mayo de 2025.

Community, G. E., 2014. Godot engine. Motor de videojuegos de código abierto.
URL: https://godotengine.org

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V., 2017. Carla: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. PMLR.

Fischler, M. A., Bolles, R. C., 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24 (6).
DOI: 10.1145/358669.358692

FSOCO Team, 2021. Fsoco: Formula student objects in context dataset. https://www.fsoco-dataset.com, dataset público para Formula Student.
URL: https://www.fsoco-dataset.com

M. Oroz and A.Colmenero, 2025. Mirena simulator. https://github.com/Tecnun-eRacing/mirena_sim.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W., May 2022. Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics 7 (66).
URL: http://dx.doi.org/10.1126/scirobotics.abm6074
DOI: 10.1126/scirobotics.abm6074

Newman, W. S., 2007. Team case and the 2007 darpa urban challenge. In: Proceedings of the 2007 DARPA Urban Challenge. Último acceso: 24 de junio de 2025.
URL: https://api.semanticscholar.org/CorpusID:18147969

Perlin, K., 2002. Improving noise. In: ACM Transactions on Graphics (TOG). Vol. 21. ACM.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A., 2025. Ros: an open-source robot operating system. https://github.com/ros/geometry2, Último acceso: 24 de junio de 2025.

Racing, A., 2018. Fssim: Formula student simulator.
URL: https://github.com/AMZ-Driverless/fssim

Razavikia, S., Amini, A., Daei, S., 2020. Reconstruction of binary shapes from blurred images via hankel-structured low-rank matrix recovery. IEEE Transactions on Image Processing 29.
DOI: 10.1109/TIP.2019.2950512

Rusu, R. B., Cousins, S., May 9-13 2011. 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA). Shanghai, China.
DOI: 10.1109/ICRA.2011.5980567

Saillot, M., Michel, D., Zidna, A., 2024. B-spline curve approximation with transformer neural networks. Mathematics and Computers in Simulation 223.
URL: https://www.sciencedirect.com/science/article/pii/S0378475424001368
DOI: https://doi.org/10.1016/j.matcom.2024.04.010

Student, E. U. F., 2023. eufs_sim: Gazebo simulation for formula student driverless. https://github.com/eufs/eufs_sim, accedido el 5 de mayo de 2025.

Ultralytics, 2020. Yolov5.
URL: https://github.com/ultralytics/yolov5

Ultralytics, 2023. Yolov8: Real-time object detection framework.
URL: https://github.com/ultralytics/ultralytics