UNIVERSITAT DE BARCELONA

Trabajo Final de Grado

GRADO EN INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

Intelligent Mobile Robot System: YOLOv8, SLAM, and ROS2 Integration in a Docker-Based Deployment on LIMO

Matthew Ayete Corrales

Tutores: Dr. Manel Puig i Vidal
Dr. Eloi Puertas i Prats

Departamento de Matemáticas e Informática
Universidad de Barcelona

Barcelona, 10 de junio de 2025

Abstract

This Final Degree Project presents the design, implementation, and evaluation of a human-aware autonomous navigation system using the AgileX LIMO robot and the ROS2 framework. The proposed system combines visual perception through a YOLOv8-based object detection model with SLAM-based navigation supported by LiDAR data. The complete development process was carried out within The Construct's virtualized environment, while the final deployment was dockerized and executed on the real robot. A custom dataset of traffic signs was generated and used to train a lightweight detection model capable of recognizing multiple sign types with high precision. The robot reacts to these signs in real-time by modifying its trajectory or stopping autonomously depending on the class detected. In addition, a modular software architecture was developed, enabling the system to operate either independently or in combination with the robot's navigation stack. The project demonstrates that it is possible to build a robust and scalable robotic system using open-source tools, contributing to the fields of autonomous navigation, vision-based robotics, containerized deployment, and ROS2-driven modular systems.

Resumen

Este Trabajo de Fin de Grado presenta el diseño, implementación y evaluación de un sistema de navegación autónoma consciente del entorno humano utilizando el robot LIMO de AgileX y el framework ROS2. El sistema propuesto combina percepción visual mediante un modelo de detección de objetos basado en YOLOv8 con navegación SLAM apoyada en datos LiDAR. Todo el desarrollo se llevó a cabo dentro del entorno virtual de The Construct, y posteriormente se dockerizó el sistema para su despliegue en el robot físico. Se generó un conjunto de datos personalizado de señales de tráfico, utilizado para entrenar un modelo ligero capaz de detectar múltiples tipos de señales con alta precisión. El robot reacciona ante estas señales en tiempo real, modificando su trayectoria o deteniéndose según la clase identificada. Además, se diseñó una arquitectura modular que permite el uso del sistema tanto de forma independiente como integrada dentro del stack de navegación del robot. El proyecto demuestra que es posible construir un sistema robótico robusto y escalable empleando herramientas de código abierto, contribuyendo así a los campos de la navegación autónoma, la robótica basada en visión, la dockerización de sistemas y el desarrollo modular con ROS2.

Resum

Aquest Treball de Fi de Grau presenta el disseny, implementació i avaluació d'un sistema de navegació autònoma conscient de l'entorn humà utilitzant el robot LIMO d'AgileX i el framework ROS2. El sistema proposat combina la percepció visual mitjançant un model de detecció d'objectes basat en YOLOv8 amb la navegació SLAM, recolzada en dades de LiDAR. Tot el desenvolupament es va realitzar en l'entorn virtual de The Construct, i posteriorment es va dur a terme la dockerització del sistema per al seu desplegament al robot físic. Es va generar un conjunt de dades personalitzat de senyals de trànsit, utilitzat per entrenar un model lleuger capaç de detectar múltiples tipus de senyals amb una gran precisió. El robot reacciona davant aquestes senyals en temps real, modificant la seva trajectòria o aturant-se segons la classe detectada. A més, es va desenvolupar una arquitectura modular que permet utilitzar el sistema de forma independent o integrat dins l'stack de navegació del robot. El projecte demostra que és possible construir un sistema robòtic robust i escalable amb eines de codi obert, contribuint als àmbits de la navegació autònoma, la robòtica basada en visió, la dockerització de sistemes i el desenvolupament modular amb ROS2.

Agradecimientos

Quiero dedicar unas líneas a agradecer a quienes han estado a mi lado durante este proyecto.

En primer lugar, a mi pareja. Aunque los detalles técnicos de este trabajo muchas veces le sonaban a otro idioma, nunca dejó de escucharme ni de interesarse. En los momentos más frustrantes, cuando algo fallaba sin explicación aparente, ella me daba ánimos para seguir adelante y dar con la solución.

También quiero agradecer al Dr. Manel Puig i Vidal, por acompañarme en todo el desarrollo técnico del proyecto y por su disponibilidad constante con el trabajo sobre el robot LIMO. Y al Dr. Eloi Puertas i Prats, por su ayuda valiosa con la parte escrita y por aportar ideas clave que dieron forma al proyecto tal como es ahora.

Por último, gracias a mi familia y a mis amigos. Sus ánimos, consejos y paciencia han sido parte esencial del proceso. Compartir con ellos lo que iba logrando me ayudó a ver el proyecto con otros ojos y a encontrar nuevas maneras de avanzar cuando me sentía estancado.

Índice

1. Introducción

1.1. Contexto

En la actualidad, dotar a un robot de capacidad de movimiento autónomo ya no constituye un reto suficiente. El verdadero desafío radica en conseguir que dicho desplazamiento se produzca de manera informada y contextualizada, es decir, que el robot sea capaz de percibir, interpretar y responder adecuadamente a los estímulos de su entorno. Esta necesidad se acentúa aún más en contextos donde interactúa con seres humanos, como ocurre en la robótica asistencial, cuyo objetivo va más allá de la movilidad, buscando construir sistemas capaces de comprender su entorno para actuar de manera coherente y segura.

En este contexto, la integración de sensores como el LiDAR junto con técnicas avanzadas de visión artificial ha abierto nuevas posibilidades para la percepción inteligente. El LiDAR proporciona al robot una representación tridimensional del entorno, mientras que modelos como YOLOv8 permiten detectar y clasificar señales, objetos u obstáculos con alta velocidad y precisión. Esta combinación de tecnologías se presenta como una base sólida para diseñar sistemas de navegación cognitiva.

Además, el uso de arquitecturas como ROS2 ha facilitado la implementación de estos sistemas en entornos tanto simulados como reales. Gracias a su naturaleza modular, su soporte a sistemas distribuidos y su capacidad para operar en tiempo real, ROS2 se ha consolidado como el framework estándar en el desarrollo de software para robótica avanzada.

Este Trabajo de Final de Grado se enmarca en esta línea de investigación y desarrollo, proponiendo la construcción de un sistema que combine percepción visual, navegación autónoma y respuesta contextual. Si bien el enfoque se plantea desde una perspectiva general y aplicable a múltiples plataformas robóticas, el robot LIMO de AgileX ha sido utilizado como plataforma de validación experimental. La mayor parte del desarrollo ha sido realizado sobre la plataforma The Construct, permitiendo iterar en simulación antes de desplegar en el robot físico. Para garantizar la portabilidad y reproducibilidad del sistema, se ha optado por la dockerización de todos los componentes, asegurando la consistencia del entorno tanto en simulación como en la ejecución final.

1.2. Motivación personal

Este proyecto no solo representa el cierre de una etapa académica, sino también la culminación de una inquietud personal que me acompaña desde pequeño. Mi interés por la robótica y la inteligencia artificial comenzó mucho antes de ingresar a la universidad. Siempre sentí una profunda curiosidad por entender cómo podíamos aprovechar la tecnología para mejorar la calidad de vida de las personas, especialmente aquellas con dificultades físicas o que necesitan asistencia en tareas cotidianas.

Desde niño observé cómo mi padre sufría con constantes dolores en sus rodillas. Esto me llevó a preguntarme continuamente: ¿Podría la tecnología algún día ofrecer una solución real a problemas como estos? La robótica, desde entonces, se convirtió en un camino lógico hacia la respuesta. Un día, viendo la película *Cómo entrenar a tu dragón*, me impresionó profundamente cómo el protagonista, tras perder una pierna, adaptaba una prótesis hecha a medida que le permitía llevar una vida plena. Esa imagen se quedó grabada en mi mente y me hizo reflexionar sobre las posibilidades reales que podría ofrecer la tecnología robótica no solo como entretenimiento o industria, sino como una herramienta al servicio de las personas.

Decidí emprender este proyecto utilizando el robot LIMO porque ofrecía precisamente esa capacidad: interactuar de forma segura y consciente con el entorno. Mi idea fue crear algo que no solo demostrara habilidades técnicas, sino que sirviera como punto de partida hacia aplicaciones reales y significativas. La integración de sistemas de navegación autónoma con detección visual y SLAM no solo proporciona una solución técnica robusta, sino que abre puertas hacia futuros desarrollos en robótica asistencial, logística inteligente y ciudades inteligentes.

Más allá de cumplir con los objetivos académicos, deseo que este proyecto sirva como punto de referencia para proyectos futuros que puedan continuar explorando estos caminos, aportando soluciones prácticas y mejorando la vida cotidiana de las personas mediante la robótica. Es mi intención seguir desarrollando sistemas que tengan un impacto positivo en la sociedad, ya sea ayudando a personas con movilidad reducida, facilitando tareas repetitivas o mejorando la interacción humano-robot en contextos cotidianos.

Figura 1: *Prótesis del protagonista de la película (DreamWorks, 2010)*

1.3. Objetivo

El objetivo general de este Trabajo de Final de Grado es el estudio, desarrollo e implementación de técnicas de navegación autónoma y percepción del entorno aplicadas a vehículos inteligentes, mediante el uso de sensores como el LiDAR y herramientas de Inteligencia Artificial para la interpretación visual. Todo el trabajo se ha llevado a cabo bajo el framework ROS2, siguiendo una metodología modular y escalable. Para validar los algoritmos desarrollados, se ha aplicado este enfoque a un caso práctico con el robot LIMO, utilizando entornos simulados en The Construct y posteriormente desplegando el sistema en el robot físico mediante contenedores Docker.

1.3.1. Objetivos específicos

Para alcanzar este objetivo general, se definen los siguientes objetivos particulares:

- Validar las técnicas de navegación autónoma mediante la implementación y prueba de los algoritmos desarrollados en un entorno simulado antes de su transferencia al robot físico.
- Aplicar los algoritmos al robot LIMO como caso de estudio para demostrar su viabilidad en un robot móvil real.
- Desarrollar y evaluar diferentes módulos funcionales basados en el sensor LiDAR del robot, incluyendo:
  - Un módulo de evitación de obstáculos, para mejorar la capacidad de desplazamiento en entornos dinámicos.
  - Un sistema de seguimiento de paredes que mantenga una distancia adecuada mediante reglas de control adaptativas.
  - Un sistema de navegación por coordenadas basado en mapas generados mediante SLAM.
- Diseñar e implementar un sistema de detección visual de señales utilizando modelos YOLOv8 entrenados específicamente para el caso, permitiendo al robot tomar decisiones en función de señales de tráfico u otras señales relevantes del entorno.
- Realizar la dockerización completa del sistema para facilitar su despliegue tanto en entornos simulados como en el robot físico, asegurando la portabilidad y la reproducibilidad del trabajo.
- Integrar todos los componentes en un sistema unificado basado en ROS2, estructurado en nodos independientes, favoreciendo su mantenimiento, pruebas y futuras ampliaciones.

2. Planificación del proyecto

2.1. Resumen de la planificación

La planificación inicial contempla una fase de exploración de ideas durante los primeros meses del curso académico. Posteriormente, se planificó una etapa de formación autodidacta en el entorno de simulación de The Construct y en el uso del framework ROS2, como preparación antes del inicio del desarrollo.

A partir de enero de 2025, el trabajo se centró en el diseño e implementación de las funcionalidades principales: un sistema básico de navegación autónoma basado en LiDAR, navegación guiada por coordenadas mediante mapas generados en simulación y finalmente un sistema de conducción autónoma con detección visual de señales usando YOLOv8. Todo ello se fue dockerizando progresivamente, siguiendo una estrategia tipo TDD (Test Driven Development), lo que permitió validar cada componente por separado en simulación y posteriormente en el robot físico LIMO.

Paralelamente al desarrollo, se fueron realizando pruebas prácticas y se redactó la memoria técnica del proyecto.

2.2. Planificación (Diagrama de Gantt)

A continuación se muestra la planificación, estructurada en bloques de tareas con sus respectivas fechas estimadas de inicio y finalización. Esta planificación ha guiado el desarrollo progresivo del proyecto desde su fase de idealización hasta la implementación final en el robot LIMO. En la Figura 2, se observa la distribución temporal de cada etapa, destacando cómo algunas de ellas se han solapado estratégicamente para optimizar el tiempo y facilitar una integración más ágil de los distintos módulos.

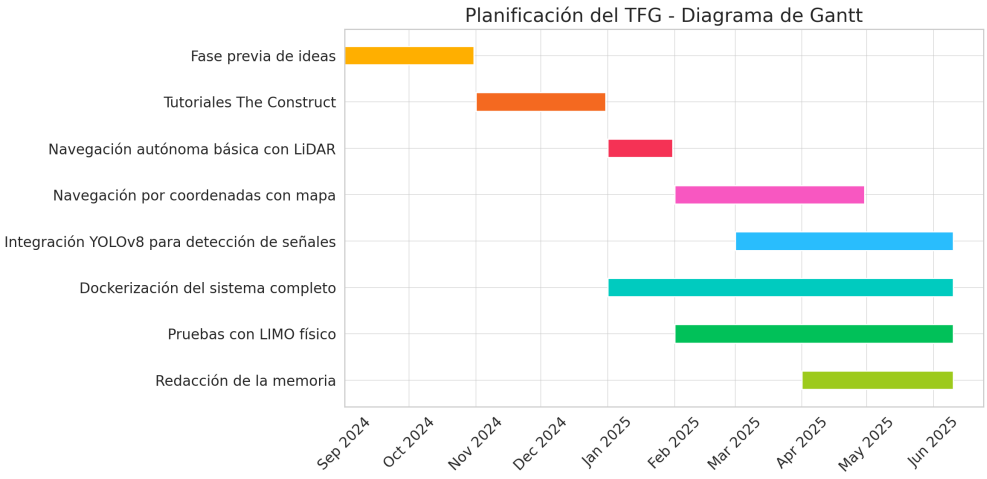

Figura 2: *Planificación temporal del proyecto: fases y tareas representadas en un diagrama de Gantt.*

2.2.1. Comparativa entre planificación inicial y final

Al inicio del proyecto se propuso una planificación estimada que, en términos generales, se ha respetado. No obstante, surgieron ciertas desviaciones que obligaron a reajustar algunas fases, sobre todo en lo referente a la integración de YOLOv8 con ROS2 y los problemas derivados de los tópicos de cámara y la dockerización del sistema de visión.

La estrategia de desarrollo tipo TDD (Test-Driven Development) permitió que la dockerización del entorno se desarrollara de forma progresiva a medida que se finalizaban los distintos módulos. A pesar de algunos contratiempos técnicos no previstos, la secuencia de trabajo se ha mantenido coherente y eficiente, alcanzando los objetivos previstos en los tiempos establecidos. Esto ha sido clave para asegurar la estabilidad del sistema en su ejecución tanto en simulación como en el robot real.

3. Costes y material

Se ha implicado el uso de diversos recursos materiales y servicios que, en un contexto empresarial, representarían una inversión económica significativa. A continuación, se detallan los principales componentes y servicios utilizados, junto con sus costes estimados:

- **Robot LIMO de AgileX**: Plataforma robótica utilizada para el desarrollo y pruebas del sistema de navegación. Precio estimado: 2.325,00 euros (IVA no incluido).
- **Ordenador portátil**: MacBook Pro de 14 pulgadas (2023) con chip Apple M2 Max y 32 GB de RAM. Precio estimado: 3.199,00 euros.
- **Licencia de The Construct**: Suscripción mensual para acceso a cursos y simuladores ROS2. Precio mensual: 39,97 euros. Se utilizó durante 2 meses, totalizando 79,94 euros.
- **Horas de trabajo**: Se estiman 450 horas de trabajo, correspondientes a 18 créditos ECTS. Asumiendo un coste de 13,08 euros por hora para un perfil de ingeniero junior, el coste total sería de 5.886,00 euros.

Recurso	**Coste estimado**
Robot LIMO	2.325,00
MacBook Pro 14”M2 Max, 32 GB RAM	3.199,00
Licencia The Construct	79,94
Horas de trabajo	5.886,00
Total estimado	**11.489,94**

Cuadro 1: Resumen de costes estimados del proyecto

Estos recursos, como el robot LIMO (Figura 3) y el ordenador portátil, fueron proporcionados por la universidad o eran de propiedad personal, por lo que no representaron un gasto directo para el desarrollo del proyecto. Sin embargo, se incluyen en esta estimación para reflejar el costo que tendría replicar este trabajo en un entorno empresarial.

La inversión en la licencia de The Construct (Figura 4) ha permitido acceder a cursos especializados y simuladores avanzados, facilitando el aprendizaje y la implementación de ROS2 en el proyecto. El cálculo de las horas de trabajo se basa en una estimación estándar y su coste se ha calculado considerando tarifas habituales en el sector para perfiles de ingeniero junior.

Figura 3: *Robot LIMO de AgileX Robotics, robot utilizado en el proyecto.*

Figura 4: *Entorno de simulación ROS2 proporcionado por The Construct, empleado para el desarrollo en software y pruebas.*

4. Análisis, Diseño e Implementación

4.1. Análisis de necesidades

El presente proyecto nace de una necesidad concreta: desarrollar un sistema de navegación autónoma que permita a un robot móvil desplazarse por entornos semiestructurados o dinámicos de manera segura, adaptativa y consciente de las señales que pueda encontrar en su camino. Este tipo de sistemas son fundamentales en contextos como la robótica asistencial, la logística automatizada o la investigación en interacción humano-robot. En particular, el reto técnico principal consistía en permitir que el robot no solo detectara su entorno mediante sensores clásicos como el LiDAR, sino que también pudiera interpretar señales visuales (en este caso, señales de tráfico), actuando en consecuencia.

4.2. Requisitos del sistema

Desde el punto de vista funcional, se definieron los siguientes requisitos:

- **Navegación autónoma segura**: el robot debe ser capaz de desplazarse por su entorno sin colisionar con obstáculos estáticos ni dinámicos, empleando sensores LiDAR como mecanismo principal de percepción del espacio.
- **Detección de señales visuales en tiempo real**: el sistema debe ser capaz de identificar señales relevantes en su entorno mediante técnicas de visión artificial.
- **Toma de decisiones basada en percepción visual**: el robot debe modificar su comportamiento (detenerse, girar, reducir la velocidad, etc.) en función de las señales detectadas.
- **Compatibilidad con ROS2**: el sistema debía desarrollarse íntegramente sobre el framework ROS2, estándar actual en robótica, que ofrece comunicación en tiempo real y escalabilidad modular.
- **Portabilidad y replicabilidad**: mediante el uso de Docker, el entorno debía poder desplegarse tanto en simuladores como en robots reales sin necesidad de replicar manualmente instalaciones complejas.
- **Funcionamiento validado en dos entornos**: el sistema debía poder desarrollarse y validarse primero en un entorno simulado (The Construct) y luego trasladarse al entorno real, utilizando el robot físico LIMO.

4.3. Limitaciones del sistema

Desde el punto de vista técnico, también se establecieron una serie de limitaciones y condicionantes que influenciaron el diseño:

- **Limitación de recursos en simulación**: el entorno The Construct, si bien es potente y versátil, impone restricciones de rendimiento, lo que obliga a optimizar las cargas de procesamiento, especialmente en tareas como la inferencia visual en tiempo real.
- **Interoperabilidad entre sensores y modelos**: no todos los modelos de cámara (Astra u Orbbec) ni todos los entornos virtuales presentaban los mismos tópicos o configuraciones, lo que requería una gestión cuidadosa del sistema de suscripciones y nombres de tópicos.
- **Tiempos de inferencia visual**: la detección de señales con modelos como YOLOv8 requiere una cierta capacidad computacional. Se optó por versiones ligeras del modelo y se priorizó la robustez en la clasificación frente a la cantidad de clases detectadas.
- **Fiabilidad del mapeo y localización**: para la navegación por coordenadas, era indispensable que el sistema de mapeo fuese lo suficientemente preciso como para generar un mapa útil y reutilizable entre simulación y robot físico.

Además, a nivel metodológico, el proyecto debía adaptarse a un enfoque de desarrollo iterativo y basado en pruebas (TDD), donde cada componente se diseñaba, implementaba, dockerizaba y validaba de manera incremental. Este enfoque favorecía la modularidad, la depuración rápida de errores y la validación progresiva en simulación antes del despliegue en el robot físico.

Estos requisitos funcionales y no funcionales del sistema se definieron cuidadosamente desde el principio, ajustándose posteriormente a medida que se identificaban nuevas necesidades técnicas, principalmente durante la integración de componentes o ante los problemas surgidos con los drivers de cámara o los tópicos de ROS.

4.4. Justificación de herramientas utilizadas

Durante el desarrollo de este trabajo, se llevó a cabo un riguroso análisis de las herramientas y tecnologías más adecuadas para afrontar el problema planteado. La elección de cada herramienta se ha hecho no sólo por su compatibilidad técnica, sino también por su madurez, estabilidad, documentación disponible y potencial de aprendizaje. A continuación, se detallan las principales herramientas empleadas y se justifica su uso frente a otras alternativas.

4.4.1. ROS2 (Robot Operating System 2)

ROS2 constituye el núcleo de la arquitectura del sistema. Se eligió por ser la evolución moderna de ROS1, con soporte nativo para comunicaciones en tiempo real (gracias a DDS), ejecución multihilo, y una mejor gestión de los recursos del sistema. A diferencia de ROS1, ROS2 permite una mayor escalabilidad y seguridad, lo cual es crucial para sistemas complejos y distribuidos.

Frente a otras soluciones middleware, ROS2 cuenta con una comunidad de desarrollo muy activa, amplia disponibilidad de paquetes específicos para robótica, y un ecosistema que favorece la integración de algoritmos de percepción, navegación y control. Además, ROS2 se considera ya el estándar emergente en la industria robótica, por lo que su elección no solo ha sido técnica, sino también estratégica a nivel profesional.

4.4.2. The Construct

Para la fase de simulación y formación, se optó por la plataforma The Construct. Esta decisión se fundamentó en varios motivos clave:

- **Facilidad de uso y puesta en marcha**: The Construct permite iniciar simulaciones completas sin necesidad de instalar localmente Gazebo, RViz o controladores.
- **Acceso a cursos especializados**: la suscripción a esta plataforma incluye acceso a formaciones avanzadas en ROS2, detección de objetos y SLAM, lo cual fue de gran utilidad en las primeras fases del proyecto.
- **Compatibilidad con Docker y ROS2**: al ofrecer imágenes preconfiguradas, se eliminan muchos de los problemas de configuración típicos en entornos robóticos.

Alternativas como simuladores locales fueron descartadas debido al alto coste temporal que suponía replicar entornos similares a The Construct desde cero. El uso de esta plataforma permitió dedicar más tiempo al desarrollo funcional del sistema que a la infraestructura subyacente.

4.4.3. Docker

Desde el inicio se adoptó una estrategia basada en contenedores para asegurar la reproducibilidad del entorno y facilitar el despliegue entre simulación y robot físico. Docker permitió encapsular todas las dependencias (versiones de Python, paquetes de ROS, librerías de visión, modelos entrenados, etc.) dentro de imágenes ligeras pero funcionales.

La modularidad de Docker también resultó esencial en el enfoque de desarrollo TDD (Test-Driven Development), permitiendo construir, probar y desplegar cada módulo del sistema por separado. Además, se aprovechó una imagen base proporcionada por The Construct (`theconstructai/limo`), lo que redujo significativamente el tiempo de configuración inicial y garantizó compatibilidad con el robot físico LIMO.

4.4.4. YOLOv8 (You Only Look Once, versión 8)

La detección visual se implementó con YOLOv8, modelo de última generación en tareas de detección de objetos en tiempo real. Inicialmente se consideró el uso de modelos basados en Keras, pero tras una fase de pruebas preliminares, se constató que YOLOv8 ofrecía una precisión superior, una velocidad de inferencia más adecuada y un soporte activo por parte de la comunidad.

Además, YOLOv8 ofrece una interfaz de entrenamiento sencilla, permite realizar inferencias directamente desde scripts Python integrables en ROS2, y soporta múltiples formatos de exportación.

4.4.5. Robot LIMO de AgileX Robotics

La plataforma robótica LIMO fue elegida por su versatilidad y compatibilidad con ROS2. Este robot compacto y potente integra un sistema de navegación basado en LiDAR, cámara RGB y una IMU, lo que lo convierte en una excelente opción para experimentar con navegación autónoma.

A nivel práctico, LIMO ofrecía:

- Soporte oficial y documentación para ROS2.
- Modo de simulación compatible con The Construct.
- Interfaz física robusta para realizar pruebas reales sin riesgo de dañar el hardware.

En comparación con otras opciones como TurtleBot3 o robots simulados genéricos, LIMO permitía una experiencia mucho más cercana a un entorno profesional, sin sacrificar accesibilidad para pruebas.

4.4.6. Otras herramientas complementarias

A lo largo del desarrollo también se utilizaron herramientas auxiliares que complementaron el proceso:

- **VSCode**: como entorno de desarrollo principal por su integración con extensiones de Python y ROS2.
- **Git**: para el control de versiones y seguimiento del progreso del código.

4.5. Diseño de la arquitectura del sistema

La arquitectura del sistema desarrollado se ha diseñado con un enfoque modular, escalable y orientado a pruebas, con el fin de asegurar una correcta integración de todos los componentes, tanto en simulación como en el robot físico LIMO. La Figura 5 representa un diagrama de alto nivel con los módulos principales y su interacción.

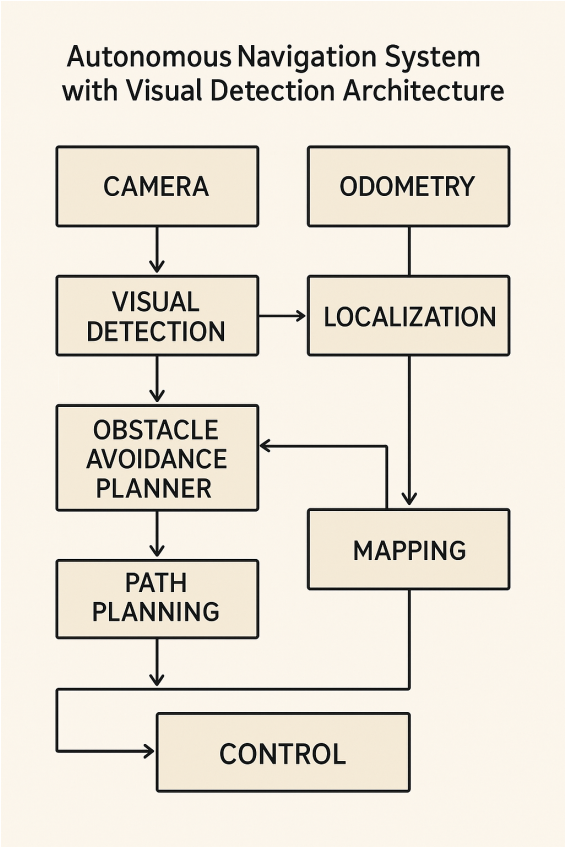

Figura 5: *Diagrama funcional del sistema desarrollado. Los nodos se comunican entre sí mediante tópicos ROS2.*

4.5.1. Subsistemas principales

La arquitectura se compone de los siguientes bloques funcionales:

- **Subsistema de percepción**: Encargado de captar el entorno mediante sensores integrados en el robot, principalmente el LiDAR y la cámara RGB. La información obtenida de estos sensores se publica en tópicos ROS estándar como `/scan` (LiDAR) y `/camera/color/image_raw` (imagen de la cámara).
- **Subsistema de visión artificial**: Implementado mediante un nodo que utiliza el modelo YOLOv8 entrenado sobre señales de tráfico. Este nodo suscribe

las imágenes de la cámara y publica resultados de inferencia (clase, posición, bounding box) en un mensaje personalizado de tipo `Yolov8Inference`. Además, sobre este nodo se ejecuta lógica condicional que determina cómo debe reaccionar el robot ante ciertas señales detectadas, modificando directamente la velocidad con comandos sobre `/cmd_vel`.

- **Subsistema de navegación**: Este bloque agrupa la lógica de navegación autónoma clásica basada en el stack `Navigation2`. Incluye la localización mediante SLAM (con Cartographer), planificación global sobre el mapa y planificación local para evitar obstáculos. Se integran nodos de control como `nav_target0.py` y el uso de `Simple Commander API` para guiar al robot hacia coordenadas objetivo.

- **Subsistema de control reactivo**: Complementario al subsistema de navegación, este bloque incluye nodos específicos para comportamientos reactivos basados en LIDAR. Entre ellos: detección de obstáculos frontales (`my_robot_selfcontrol`) y seguimiento de pared (`my_robot_wallfollower`). Estos comportamientos permiten responder a eventos inmediatos que pueden no estar reflejados aún en el mapa global.

- **Subsistema de integración y despliegue (Docker)**: Todo el sistema se ha encapsulado progresivamente en contenedores Docker. Se parte de una imagen base proporcionada por The Construct (`theconstructai/limo:humble`), sobre la que se añaden los paquetes personalizados y las dependencias necesarias (por ejemplo, Ultralytics YOLO, cv_bridge, nav2, etc.). Esta estrategia modular facilita el despliegue en múltiples entornos y asegura la replicabilidad del comportamiento entre simulación y robot real.

- **Simulación y validación en The Construct**: Todo el desarrollo ha sido testeado y validado en simulación mediante escenarios personalizados en la plataforma de The Construct. Esta herramienta provee entornos virtuales realistas, lo que permite probar algoritmos de navegación y visión antes de desplegarlos físicamente en el robot.

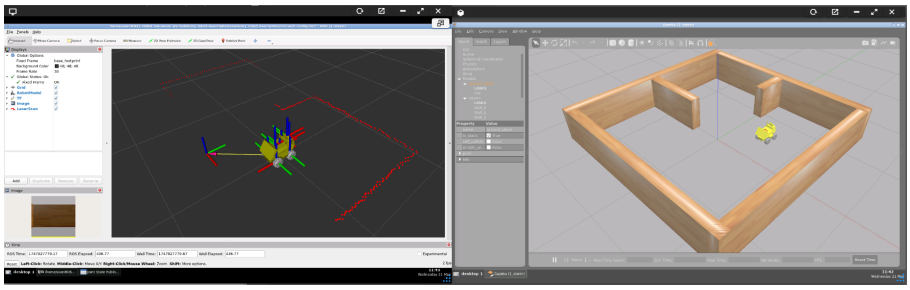

Figura 6: *Entorno de simulación (TheConstruct)*

4.5.2. Flujo de información

El flujo de datos sigue una estructura jerárquica con dos grandes bucles:

1. El **bucle de percepción y reacción**, compuesto por la cámara y el LiDAR, cuyas salidas son procesadas en tiempo real por los nodos de visión y control reactivo. Este bucle responde directamente a estímulos del entorno, permitiendo paradas, giros o maniobras evasivas.

2. El **bucle de navegación y planificación**, que construye y mantiene un mapa del entorno, localiza al robot en él y determina trayectorias óptimas hacia objetivos definidos. Este bucle funciona de forma complementaria al primero, priorizando la eficiencia y coherencia global del movimiento del robot.

4.5.3. Justificación del diseño

Esta arquitectura responde a la necesidad de disponer de un sistema flexible y desacoplado, donde cada componente pueda desarrollarse, testearse y sustituirse de manera independiente. El uso de ROS2 como middleware permite una comunicación robusta entre nodos, y la dockerización asegura que todo el ecosistema puede reproducirse en distintos entornos sin necesidad de una reconfiguración manual.

El diseño modular permitió aplicar una metodología TDD (Test-Driven Development), ya que cada funcionalidad se validó de forma aislada en simulación antes de pasar a ser parte del sistema completo y desplegarse en el LIMO físico.

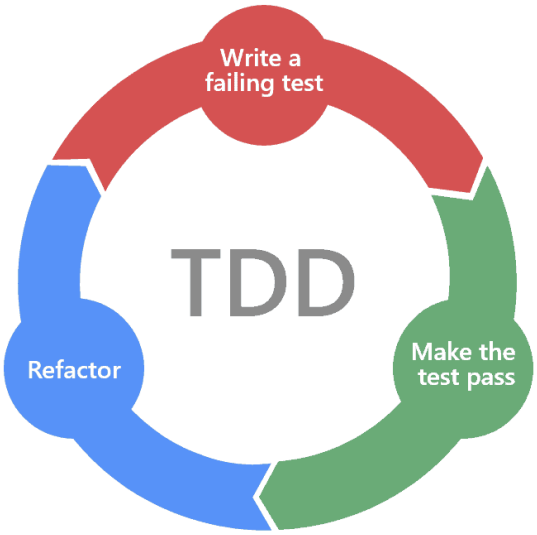

Figura 7: *Metodología TDD*

5. Implementación

La implementación del sistema ha seguido una metodología *TDD (Test-Driven Development)*, lo cual ha permitido validar por separado cada una de las funcionalidades antes de integrarlas en el entorno completo. Este enfoque no solo favoreció una mayor robustez del sistema, sino que también facilitó la resolución incremental de problemas complejos, sobre todo en la parte de integración con sensores reales y el entorno dockerizado.

5.1. Organización del proyecto

El proyecto ha sido estructurado en distintos paquetes ROS2, agrupando los scripts por funcionalidad:

- **Paquetes de percepción**: contienen los nodos responsables de adquirir y procesar datos sensoriales, como las imágenes de la cámara RGB o las lecturas del LiDAR.
- **Paquetes de navegación**: incluyen scripts asociados a la planificación de trayectorias y el control del movimiento, así como interfaces con la API de `Nav2`.
- **Paquete de detección de señales**: contiene el nodo de inferencia visual basado en YOLOv8, entrenado con un conjunto de imágenes personalizadas.
- **Paquetes de control reactivo**: scripts como `wall_follower` y `self_control` permiten reacciones rápidas a obstáculos y seguimiento de paredes en entornos estrechos.
- **Paquete de integración**: en este módulo se incluye el archivo `Dockerfile`, scripts de lanzamiento (launch files) y configuraciones específicas de entorno (por ejemplo, `params.yaml` y `config.rviz`).

Esta estructura permite que cada componente pueda mantenerse, testearse y ampliarse de forma independiente, lo cual es clave para sistemas robóticos con múltiples subsistemas concurrentes.

5.2. Nodo de detección con YOLOv8

Uno de los bloques centrales de la implementación ha sido el nodo de visión artificial, basado en un modelo YOLOv8 personalizado y entrenado específicamente para reconocer señales de tráfico como "STOP", "Prohibido", "Ceda", "Derecha" e "Izquierda". El nodo se ejecuta en ROS2 y utiliza la librería `Ultralytics` para la inferencia, así como `cv_bridge` para convertir las imágenes ROS a formato OpenCV.

El script realiza las siguientes tareas:

- Suscripción al tópico `/camera/color/image_raw`.
- Redimensionamiento de la imagen a 640x640 píxeles para ajustarse a la entrada del modelo.
- Inferencia y extracción de los resultados (nombre de la clase, coordenadas del *bounding box*).
- Publicación de resultados personalizados en el tópico `/Yolov8_Inference` mediante mensajes ROS2 definidos en un tipo customizado.
- Reacciones inmediatas (giro, parada, velocidad reducida) mediante publicación en `/cmd_vel`, en función de la clase detectada.

Este nodo fue el más complejo de implementar debido a múltiples problemas técnicos relacionados con los paquetes de cámara (diferencias entre Orbbec y Astra), IDs de The Construct y sincronización entre tópicos de ROS2. No obstante, su diseño modular permitió aislar y depurar cada incidencia por separado.

El modelo empleado fue entrenado específicamente para este proyecto y su rendimiento fue evaluado de forma exhaustiva. Los detalles del proceso de entrenamiento, así como las métricas obtenidas y su análisis crítico, se presentan en la sección 6.5.

5.3. Navegación basada en coordenadas

Otro de los módulos clave es el sistema de navegación por coordenadas, basado en la API `nav2_simple_commander`. A través de esta herramienta, el robot puede recibir objetivos (*waypoints*) definidos manualmente o generados en el entorno de simulación y desplazarse hacia ellos evitando obstáculos.

El flujo típico de este nodo es:

- Inicializar el sistema de navegación con la posición del robot.
- Cargar un mapa previamente generado mediante SLAM (utilizando `Cartographer`).
- Publicar una secuencia de objetivos en el espacio (por ejemplo: [(1.5, 0.0), (1.5, 1.5), (0.0, 1.5)]).
- Supervisar la ejecución y comprobar si los *waypoints* se alcanzan correctamente.
- Detectar condiciones de fallo y ejecutar maniobras de recuperación.

Este módulo ha sido probado extensivamente en The Construct antes de su despliegue en el LIMO físico, logrando una integración fluida con los módulos de detección y control.

5.4. Control reactivo y comportamientos de emergencia

Como complemento a la navegación planificada, se implementaron varios scripts de control reactivo que permiten respuestas inmediatas ante condiciones imprevistas. Entre ellos:

- **Wall follower**: sigue una pared lateral a una distancia constante.
- **Self-control LIDAR**: detiene el robot si se detecta un obstáculo frontal demasiado cercano.
- **Reacciones a señales visuales**: control de velocidad o dirección ante la detección de señales.

Estos scripts permiten dotar al sistema de un comportamiento robusto en entornos dinámicos, especialmente cuando las condiciones cambian rápidamente y el planificador no puede reaccionar a tiempo.

5.5. Dockerización y despliegue

Uno de los aspectos más cuidados del proyecto ha sido la integración continua mediante Docker. Se ha utilizado una imagen base proporcionada por The Construct (`theconstructai/limo:humble`), a partir de la cual se añadieron capas con los scripts y paquetes ROS2 personalizados. El `Dockerfile` define:

- Instalación de dependencias como `Ultralytics`, `OpenCV`, `rclpy`, `cv_bridge` y mensajes personalizados.
- Copia del *workspace* ROS2 y su compilación con `colcon`.
- Configuración de entorno para visualizar imágenes y lanzar nodos.

Este enfoque garantiza que todo el sistema puede reproducirse exactamente igual en cualquier entorno compatible con Docker, lo cual resulta vital tanto para el desarrollo como para su posible uso independientemente del entorno.

6. Pruebas, evaluación y resultados

6.1. Evitación de obstáculos

Este nodo implementa una lógica básica de navegación reactiva basada en el análisis del sensor LiDAR. Su objetivo principal es evitar colisiones detectando obstáculos cercanos y tomando decisiones de movimiento en consecuencia.

6.1.1. Entorno de simulación

Durante las pruebas en Gazebo, se desplegó el nodo junto con el entorno del robot en un mundo cerrado con obstáculos. El robot:

- Se activó mediante el comando de `bringup` en modo simulación.
- Publicaba mensajes de tipo `Twist` para ajustar su dirección y velocidad.
- Se observó cómo respondía eficazmente a la presencia de obstáculos en el entorno, deteniéndose o girando según la proximidad del objeto detectado.

Los logs mostraban la distancia mínima detectada y el giro aplicado, lo cual permitió afinar los umbrales de seguridad (frenar si un objeto está a menos de 0.6 metros).

6.1.2. Entorno real

En las pruebas físicas con el robot LIMO:

- El script se ejecutó dentro del contenedor Docker del robot (`bringup`).
- Los obstáculos fueron simulados con cajas y mobiliario en interiores.
- El robot fue capaz de detectar obstáculos de manera precisa gracias a su sensor LiDAR, adaptando su velocidad y dirección en tiempo real.
- Se validó que el nodo era lo suficientemente rápido como para evitar colisiones incluso con desplazamientos a velocidad media.

En este entorno real se observó una mejora significativa al limitar el rango de visión a una región frontal de $-45°$ a $45°$, lo que redujo las decisiones erróneas por detecciones laterales no relevantes.

6.2. Seguimiento de paredes

Este nodo implementa un comportamiento reactivo para mantener al robot siguiendo una pared lateral, utilizando exclusivamente datos del sensor LiDAR. El robot analiza continuamente las distancias en diferentes regiones angulares y ajusta su movimiento para mantenerse a una distancia deseada de la pared.

6.2.1. Entorno de simulación

Las pruebas en Gazebo se realizaron en un mundo con pasillos delimitados por muros. El nodo:

- Estaba suscrito al tópico `/scan` y configurado con política QoS `RELIABILITY_BEST_EFFORT`.
- Identificaba las distancias mínimas en tres regiones: frente ($-45°$ a $45°$), izquierda ($45°$ a $110°$) y derecha ($-110°$ a $-45°$).
- Ajustaba su trayectoria según la proximidad de la pared izquierda, intentando mantener una distancia constante (0.5 metros).
- Mostraba en los logs las decisiones tomadas y el número de lecturas válidas por región.

El comportamiento fue estable en trayectorias rectas y al enfrentar esquinas, donde priorizaba girar manteniéndose junto a la pared izquierda.

6.2.2. Entorno real

En el entorno físico con el robot LIMO:

- Se ejecutó el nodo dentro del contenedor con ROS2 Humble y el sensor LiDAR correctamente activado.
- El robot fue colocado junto a una pared lisa y se le permitió avanzar libremente.
- El sistema detectaba la pared y ajustaba su orientación para seguirla sin colisionar.
- En zonas sin paredes laterales, el robot continuaba recto hasta encontrar un nuevo obstáculo lateral o frontal.

Este nodo demostró ser especialmente útil para tareas de navegación semiestructurada, como pasillos o recorridos delimitados por muros.

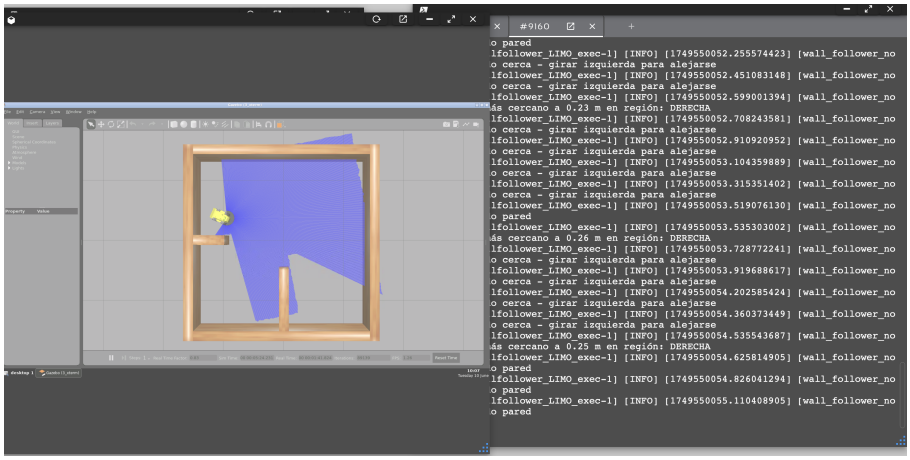

Figura 8: *Simulación del nodo Wallfollower (TheConstruct))*

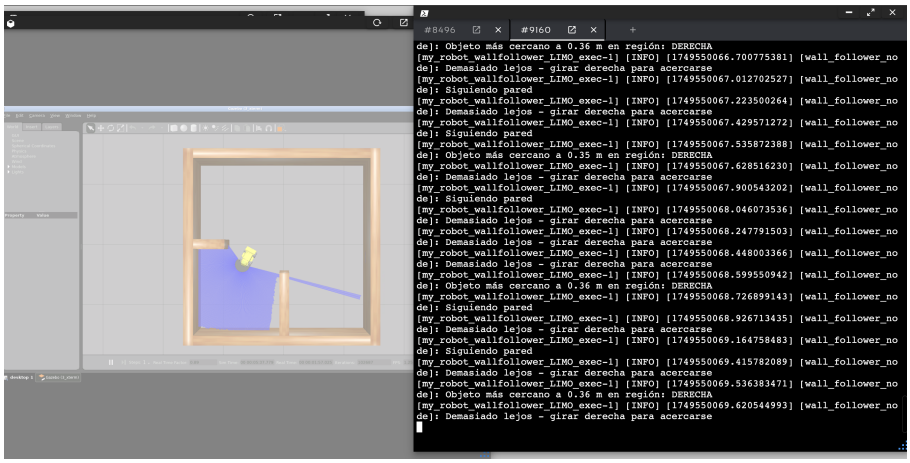

Figura 9: *Simulación del nodo Wallfollower (TheConstruct))*

6.3. Navegación SLAM y seguimiento de waypoints

Los paquetes `my_robot_cartographer`, `my_robot_navigation2` y `my_robot_nav_control` fueron desarrollados de forma modular, permitiendo su uso independiente para tareas específicas como mapeado, navegación y control por waypoints. No obstante, su integración coordinada es lo que permite alcanzar un comportamiento de navegación autónoma robusto y preciso. A continuación, se describe su funcionamiento y los resultados obtenidos en los distintos entornos.

6.3.1. Entorno de simulación

Fase de mapeado: Esta fase consiste en explorar un entorno desconocido mediante sensores, para construir un mapa 2D navegable que se usará posteriormente como referencia.

- Se utilizó el paquete `my_robot_cartographer` con la instrucción de lanzamiento para iniciar el SLAM en el mundo simulado `square4m_sign.world`, con tiempo simulado habilitado (`use_sim_time = true`).
- El robot fue teleoperado para explorar el entorno y generar el mapa.
- Una vez finalizada la exploración, se guardó el mapa en la carpeta `my_robot_navigation2/map` con el nombre `my_map.yaml`.

Fase de navegación: En esta fase se usa el mapa previamente generado para planificar y ejecutar trayectorias hacia objetivos definidos por el usuario, evitando obstáculos en tiempo real.

- Se utilizó el paquete `my_robot_navigation2` para cargar el mapa y lanzar el stack de navegación.
- Se configuró el archivo `limo_sw.yaml`, destacando los siguientes parámetros ajustados para mejorar la precisión:
  - `controller_frequency`: 10.0 Hz
  - `planner_patience`: 5.0 s
  - `motion_model_type`: omni
  - `max_rotational_velocity`: 0.8 rad/s
  - `amcl.base_frame_id`: base_link
- Se utilizó RViz para establecer la posición inicial (`2D Pose Estimate`) y los objetivos de navegación, tanto individuales como múltiples (modo "Nav Through Poses").

Control programado con my_robot_nav_control: Este módulo permite controlar al robot de forma automática mediante programación, sin intervención manual en RViz, mediante la definición de waypoints.

- Se desarrolló un nodo en Python que interactúa con `Nav2` mediante la API `nav2_simple_commander`.
- Se definieron waypoints en coordenadas absolutas del mapa y se enviaron secuencialmente al action server.
- En caso de pérdida de localización, se implementó una lógica de reintento tras un pequeño retardo.

6.3.2. Entorno real

Bringup y mapeado: Aquí el robot explora físicamente el entorno real utilizando sus sensores, construyendo un mapa navegable mediante SLAM, sin uso de simuladores.

- El bringup del robot LIMO se realiza automáticamente al encender el dispositivo.
- Se lanzó `my_robot_cartographer` con `use_sim_time = false` para registrar el entorno real.
- El mapa generado se almacenó localmente para su reutilización.

Navegación y ajuste: Utilizando el mapa generado, el robot navega de forma autónoma por el entorno real. Para mejorar la estabilidad, se ajustaron parámetros específicos del sistema de localización.

- Se utilizó el paquete `my_robot_navigation2` con el archivo de parámetros `limo_real.yaml`, adaptado para el entorno físico:
  - `base_frame_id`: odom
  - `motion_model_type`: differential
  - Se ajustaron los parámetros de `AMCL` para reducir la frecuencia de actualización y evitar sobrecarga de cómputo en tiempo real.
- Se empleó RViz para inicializar la localización y lanzar la navegación hacia los puntos objetivo.

Control programado: De igual manera que en simulación, el módulo de control automático permite al robot seguir una secuencia de waypoints sin necesidad de intervención directa.

- Una vez completada la localización manual en el mapa, se ejecutó el nodo `nav_target0_exec` para iniciar la navegación secuencial entre waypoints.
- El comportamiento fue satisfactorio, sin pérdida de trayectoria y con adaptación dinámica a obstáculos inesperados.

Los tres paquetes son independientes, pero en conjunto ofrecen una solución completa y flexible para navegación SLAM con ROS2. Si bien pueden emplearse por separado según las necesidades del proyecto, su integración maximiza la eficiencia y fiabilidad del sistema.

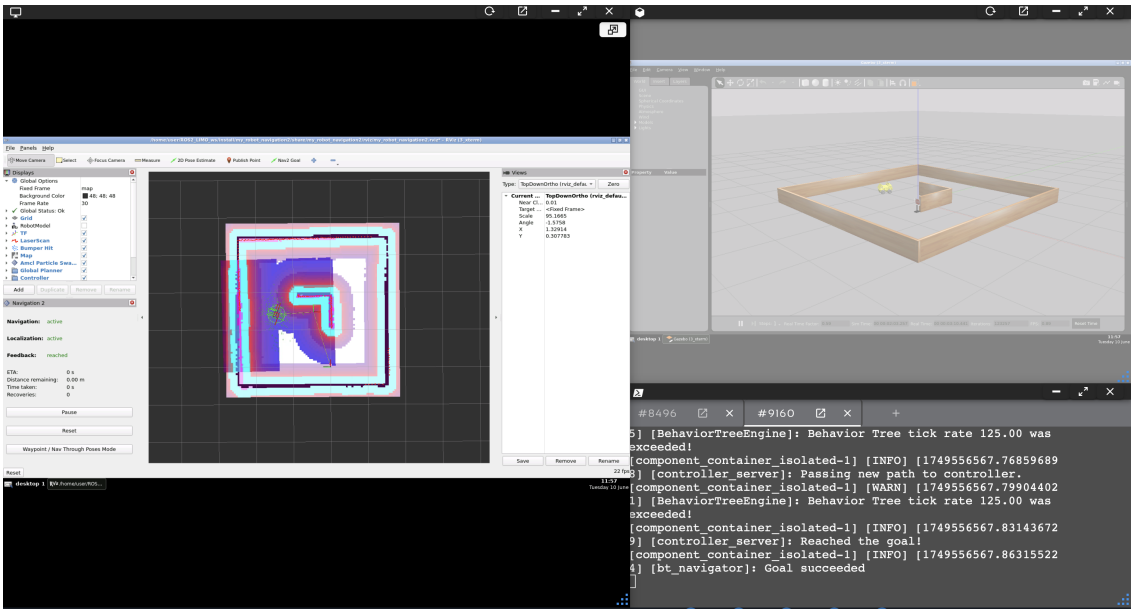

Figura 10: *Simulación del nodo NAV por coordenadas (TheConstruct))*

6.4. Comparativa entre Keras y YOLOv8

Durante las primeras fases del proyecto se contempló el uso de `Keras` como enfoque inicial para implementar un clasificador visual de señales de tráfico. Mediante una red neuronal convolucional tradicional (CNN), el objetivo era tomar imágenes completas como entrada y clasificarlas en una de las categorías predefinidas (por ejemplo, *STOP*, *Ceda*, *Prohibido*, etc.). Esta solución resultó sencilla de desarrollar inicialmente, con tiempos de entrenamiento reducidos y un conjunto de herramientas muy accesible. Sin embargo, pronto se hicieron evidentes sus limitaciones estructurales.

En primer lugar, el enfoque con Keras obligaba a centrar la señal dentro del encuadre, ya que el sistema solo proporcionaba una clase por imagen sin conocer la ubicación del objeto. En entornos reales, esto implicaba que el robot necesitaba ver una única señal bien centrada, sin elementos de distracción, lo cual no era realista. Además, incluso cuando no había ninguna señal en la imagen, el modelo emitía siempre una predicción, lo que derivaba en falsos positivos constantes durante la navegación.

Ante esta situación, se optó por explorar alternativas más robustas dentro del campo de la detección de objetos. En este contexto, **YOLOv8** (You Only Look Once) emergió como una solución ideal. Este modelo permite detectar simultáneamente múltiples objetos dentro de una misma imagen, junto con sus respectivas coordenadas espaciales, representadas mediante *bounding boxes*. Además, si no detecta ninguna clase relevante, el modelo no devuelve resultados, lo que reduce significativamente los errores en la toma de decisiones.

La elección de YOLOv8 también estuvo motivada por su compatibilidad con ROS2 y su disponibilidad a través de la librería `Ultralytics`, lo que permitió una integración más fluida dentro de un sistema dockerizado. Además, su alto rendimiento incluso en dispositivos con recursos limitados, como el robot LIMO, lo convirtió en la opción óptima para el sistema final.

6.4.1. Comparativa de ventajas y desventajas:

- **Keras (clasificación con CNN):**
  - **Ventajas:**
    - Fácil de implementar para prototipos rápidos.
    - Curva de aprendizaje baja.
    - Menor tiempo de entrenamiento con datasets reducidos.
  - **Desventajas:**
    - No detecta múltiples objetos ni sus posiciones.
    - Falsa predicción en ausencia de señales.
    - Poco robusto frente a ruido, ángulos variables o fondos complejos.
    - Difícil integración directa con ROS2.

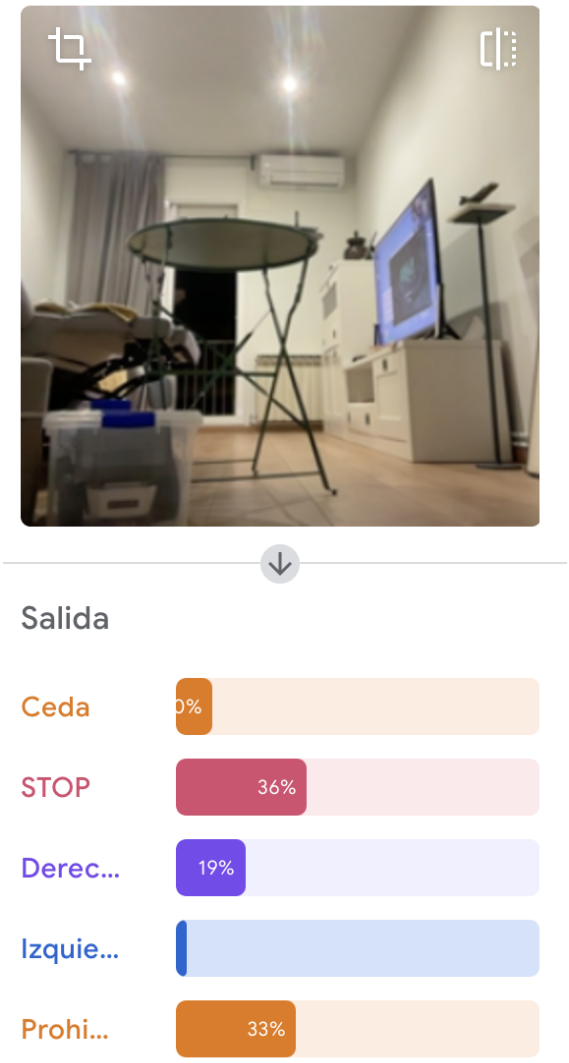

Figura 11: *Predicción de Keras en entorno sin señales con respuestas erróneas*

- **YOLOv8 (detección de objetos):**
  - **Ventajas:**
    - Detección de múltiples señales con localización precisa.
    - Omite predicciones si no detecta objetos válidos.
    - Gran precisión incluso en condiciones variables.
    - Fácil integración en ROS2 con nodos personalizados.
  - **Desventajas:**
    - Requiere más recursos para entrenamiento e inferencia.
    - Necesita datasets más extensos y bien etiquetados.
    - Curva de configuración más técnica.

Figura 12: *Predicción de YOLO en entorno sin señales*

```
image 1/1 /Users/matt/Downloads/Signals Detection/test/images/test_YOLO.jpg: 448x640 (no detections), 35.7ms
Speed: 2.0ms preprocess, 35.7ms inference, 0.3ms postprocess per image at shape (1, 3, 448, 640)
```

Figura 13: *Respuesta de YOLO en entorno sin señales (no detections)*

6.5. Entrenamiento del modelo YOLOv8

El sistema de visión artificial del robot se apoya en un modelo YOLOv8 entrenado específicamente para identificar señales de tráfico. El proceso de entrenamiento ha sido uno de los pilares más relevantes del desarrollo del proyecto, no solo por la precisión alcanzada, sino también por la flexibilidad del sistema ante condiciones reales.

6.5.1. Preparación del dataset

Las imágenes utilizadas para el entrenamiento fueron capturadas manualmente con el propio robot en distintos escenarios controlados. Se recopilaron más de **600 imágenes**, correspondientes a cinco clases: *STOP*, *Ceda*, *Prohibido*, *Derecha* e *Izquierda*. Estas imágenes fueron etiquetadas de forma manual utilizando la plataforma Roboflow, la cual permitió crear conjuntos balanceados y exportarlos en formato compatible con YOLOv8 (`data.yaml` más una estructura de carpetas con información relevante del entrenamiento).

6.5.2. Entrenamiento del modelo

El modelo fue entrenado utilizando la versión ligera `YOLOv8n`, ideal para dispositivos embebidos o con recursos limitados, como es el caso del robot LIMO. El proceso de entrenamiento se llevó a cabo en CPU, dentro de un contenedor Docker configurado específicamente para este propósito. Se utilizó la librería `ultralytics` con una configuración de 100 épocas, tamaño de imagen de 640 píxeles, optimización por `SGD` y una partición de los datos de 80

El flujo seguido puede resumirse con el siguiente pseudocódigo:

Algorithm 1 Entrenamiento del modelo YOLOv8

1: Cargar modelo base `YOLOv8n`
2: Cargar archivo `data.yaml` generado por Roboflow
3: Establecer parámetros de entrenamiento:
4: `epochs` ← 100
5: `tamaño_imagen` ← 640
6: `dispositivo` ← CPU
7: Ejecutar proceso de entrenamiento
8: Evaluar el modelo sobre el conjunto de validación
9: Probar el modelo con imágenes reales del entorno
10: Guardar pesos finales del modelo como `yolov8n_custom.pt`
11: Exportar modelo a formato ONNX para compatibilidad futura

6.5.3. Evaluación del rendimiento

El modelo entrenado alcanzó un **mAP@0.5 de 0.995**, lo que indica un rendimiento sobresaliente. A continuación se detallan las métricas obtenidas y algunas visualizaciones:

- **Matrices de confusión:** tanto en valores absolutos como normalizados, muestran una clasificación perfecta de las instancias del conjunto de test (Figuras 14 y 15).

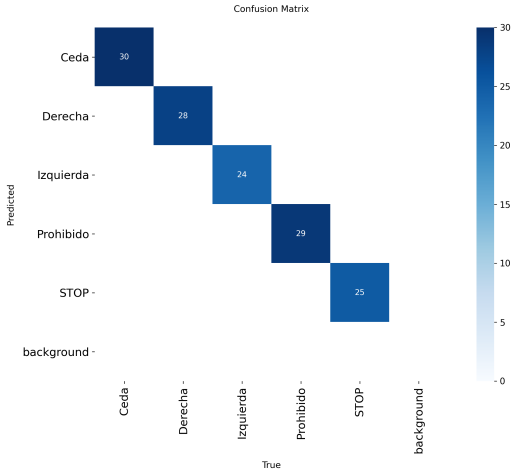

Figura 14: Matriz de confusión absoluta del modelo YOLOv8

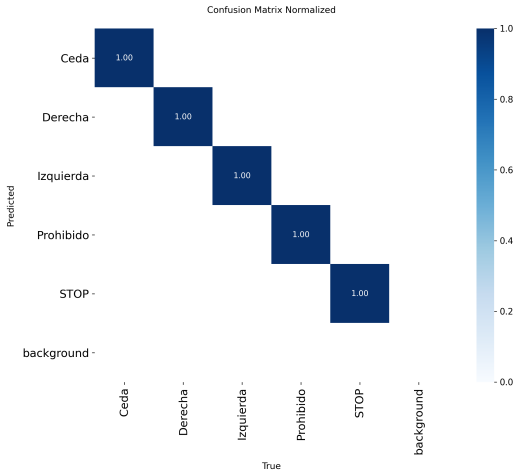

Figura 15: Matriz de confusión normalizada del modelo YOLOv8

- **Curva de precisión-recall:** se alcanzó una precisión y exhaustividad cercana a 1.0 en todas las clases (Figura 16).

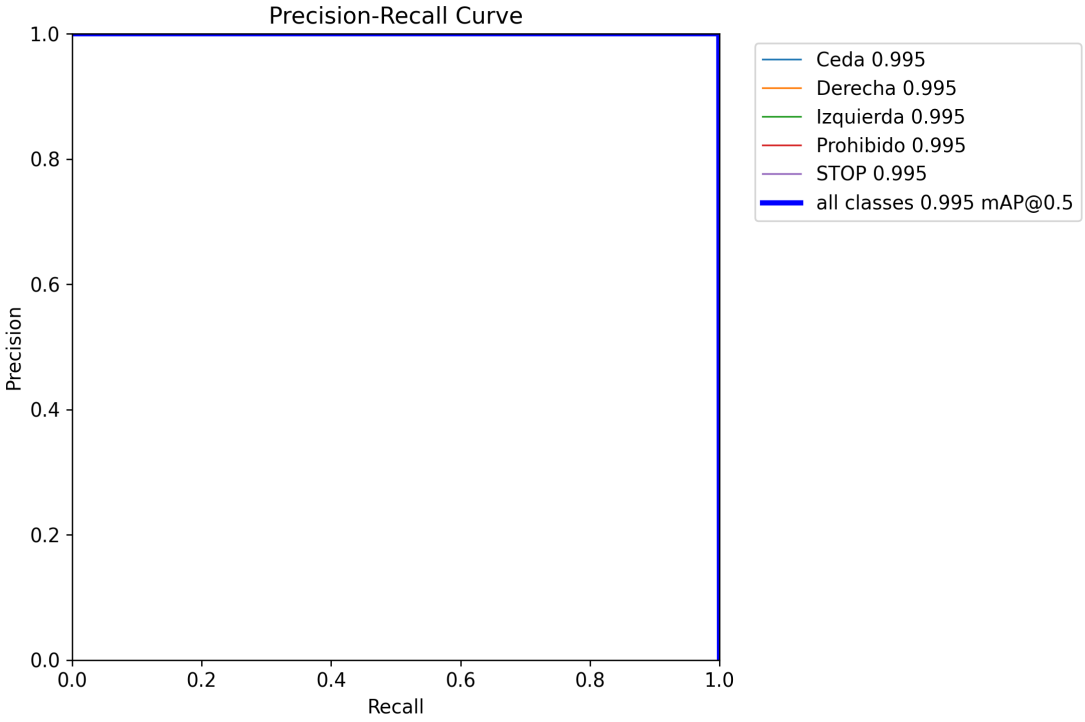

Figura 16: Curva Precisión-Recall para cada clase

- **Curvas precisión/confianza y F1/confianza:** muestran una respuesta estable del modelo en distintos umbrales de confianza, con valores óptimos superiores al 0.9 (Figuras 17 y 18).

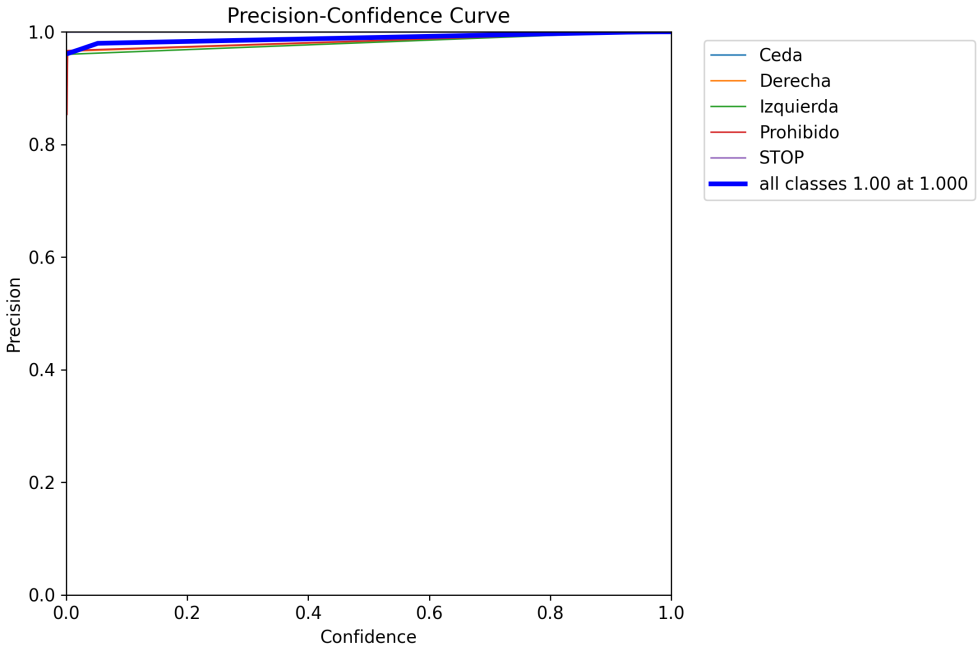

Figura 17: Curva Precisión vs Confianza

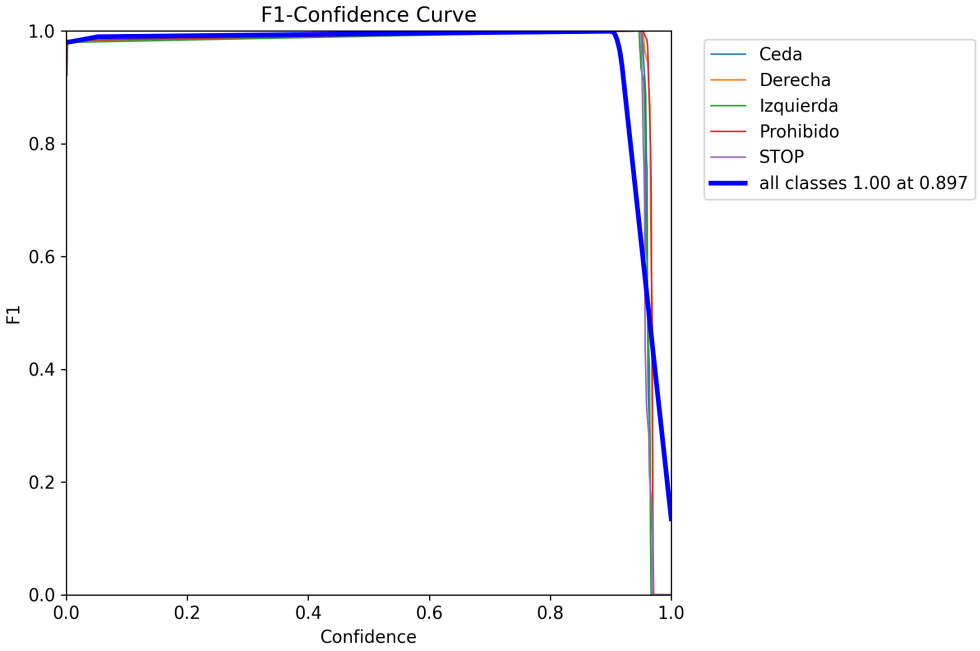

Figura 18: Curva F1 vs Confianza

- **Ejemplos visuales:** se incluyen predicciones correctas del modelo sobre imágenes reales, con alta confianza y correcta identificación de clases y *bounding boxes* (Figuras 19 y 20).

Figura 19: Predicciones del modelo sobre imágenes del conjunto de validación

Figura 20: Predicciones del modelo sobre imágenes del conjunto de validación

6.5.4. Resolución del resultado del modelo

El entrenamiento del modelo demuestra ser altamente efectivo para el caso de uso planteado, con una precisión muy elevada en el conjunto de test. Esto se debe en parte a la alta calidad del dataset, a la simplicidad visual de las señales utilizadas y al enfoque modular de integración del nodo de inferencia. Como mejora futura, podría considerarse la generalización del modelo con señales en distintos contextos de iluminación, distancia, orientación o mayor ruido visual.

6.6. Detección e identificación visual

Una vez entrenado el modelo YOLOv8, se diseñó el nodo `my_robot_ai_identification`, encargado de realizar inferencias en tiempo real con imágenes recibidas del robot y actuar en función de las señales detectadas. Este nodo fue implementado en Python y adaptado a ROS2, haciendo uso de la librería `Ultralytics` para cargar el modelo y realizar predicciones, y `cv_bridge` para convertir imágenes del tipo `sensor_msgs/Image` en objetos OpenCV manipulables.

6.6.1. Entorno de simulación

Durante la fase de pruebas en The Construct se verificó que el nodo pudiera detectar correctamente las señales en tiempo real desde el entorno virtual y emitir mensajes con la clase detectada.

- Se lanzó el nodo `my_robot_ai_identification` dentro del contenedor Docker donde estaba instalado el modelo.
- El tópico de entrada configurado fue `/camera/color/image_raw`.
- Las inferencias se realizaban a aproximadamente 4–5 fps sobre CPU, suficiente para una detección estable.
- El nodo publicaba los resultados en el tópico `/Yolov8_Inference`, permitiendo que otros nodos del sistema (como los de control) pudieran reaccionar ante las señales detectadas.
- Se validó la detección de todas las clases entrenadas (*STOP*, *Ceda*, *Prohibido*, *Derecha*, *Izquierda*) y se comprobó que, en ausencia de señales, no se generaban predicciones falsas.
- Cabe destacar que el nodo revisaba la presencia de señales cada **20000 ms**, lo cual resultó perceptiblemente lento durante la navegación. Esta latencia se debe a las limitaciones propias del entorno de simulación de The Construct, y no al modelo en sí. Aunque se identificó como una posible área de mejora, en este trabajo no se ha abordado dicha optimización, dado que el foco del proyecto no estaba centrado en el rendimiento computacional ni en la optimización de infraestructura.

6.6.2. Entorno real

En el entorno físico, el nodo mostró un comportamiento consistente con la simulación. Las pruebas se realizaron en espacios reales controlados con impresiones en papel de señales reales.

- Se conectó el robot LIMO mediante SSH y se ejecutó el nodo con el modelo `yolov8n_custom.pt` ya entrenado.
- Las imágenes capturadas por la cámara RGB del robot eran enviadas automáticamente al nodo de inferencia.
- Las predicciones se representaban visualmente en tiempo real para depuración, mostrando las cajas delimitadoras, etiquetas y nivel de confianza.
- Se midió la latencia entre la detección y la publicación en el tópico `/Yolov8_Inference`, siendo inferior a 0.5 s.
- Se verificó el funcionamiento en distintas condiciones de iluminación, distancia y ángulo, manteniendo una alta tasa de acierto.

Este nodo de inferencia actúa como un componente autónomo, pero totalmente integrable con el resto del sistema robótico, especialmente con el módulo de control de navegación. Su diseño modular y compatible con ROS2 garantiza su adaptabilidad a futuros sistemas o nuevos modelos de detección.

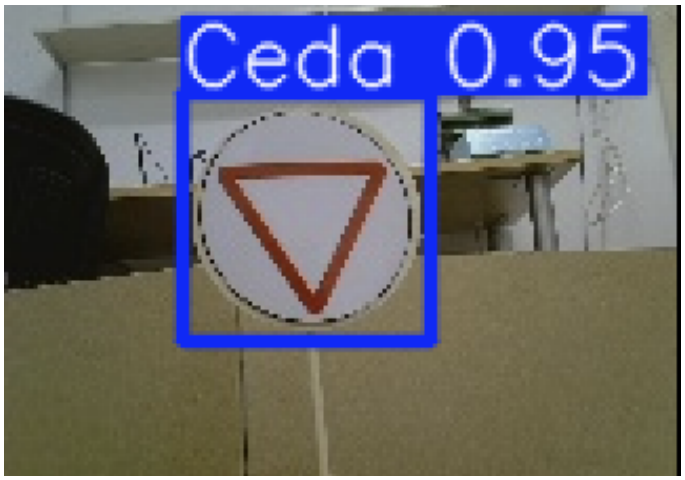

Figura 21: Predicción del modelo sobre imagen de señal real (Ceda)

6.7. Reacción del robot ante señales detectadas

Una vez identificada una señal de tráfico mediante el nodo `my_robot_ai_identification`, el sistema activa un mecanismo de decisión y control que modifica la trayectoria del robot en función de la clase detectada. Esta lógica se implementa de forma acoplada a la navegación autónoma basada en coordenadas, integrando de manera inteligente la percepción visual con el movimiento autónomo en el mapa.

6.7.1. Entorno de simulación

- En Gazebo, tras lanzar el nodo de navegación junto al nodo de inferencia visual, el robot comienza a moverse hacia una serie de objetivos definidos mediante `Waypoints`.
- Durante el trayecto, el nodo de predicción analiza la imagen de cámara y, al detectar una señal, publica su clase.
- Otro nodo, suscrito al tópico `/Yolov8_Inference`, recibe dicha clase y ejecuta una orden:
  - `STOP`: detener el robot durante un tiempo definido.
  - `Ceda el paso`: reducir la velocidad durante unos segundos.
  - `Prohibido el paso`: detener completamente el robot, cancelar todos los movimientos y finalizar el nodo con `rclpy.shutdown()`. Esta acción puede entrar en conflicto con la navegación activa, pero se consideró necesaria para evitar avanzar en zonas restringidas.
  - `Gira derecha / izquierda`: modificar su sentido hacia el siguiente punto objetivo de la ruta.
- Esta lógica permite una navegación adaptable, en la que el robot puede modificar su comportamiento sin intervención externa.
- Los tiempos de actuación están ajustados a un entorno simulado, con latencias añadidas por el entorno The Construct. La integración de todos los módulos (detección, inferencia, control y navegación) permite una ejecución coherente incluso con dichas limitaciones.

6.7.2. Entorno real

- En condiciones reales, se realizaron pruebas en circuitos cerrados donde se colocaron señales impresas en papel.
- El nodo de inferencia detectaba las señales con gran fiabilidad, y la lógica de control reaccionaba en función de la detección.
- Por ejemplo, al encontrar una señal de STOP a menos de 1 metro, el robot se detenía durante 3 segundos antes de reanudar su movimiento.
- Ante una señal de `Prohibido`, el sistema reaccionaba de forma drástica: detenía el robot y finalizaba la ejecución del nodo. Esta decisión impide que el robot continúe navegando, por lo que es una interrupción total de la operación en curso.
- La detección en el entorno real se realizó también con una cadencia reducida (cada 7 segundos), como en la simulación. Esta limitación, heredada del entorno de desarrollo y pruebas, ha sido identificada como una mejora futura pero no fue abordada en el marco de este proyecto.

Cada uno de estos módulos —la inferencia visual, la lógica de control y la navegación autónoma— puede utilizarse de forma independiente. Sin embargo, su integración coordinada ha demostrado ser una solución robusta, flexible y eficaz para la navegación autónoma basada en percepción visual.

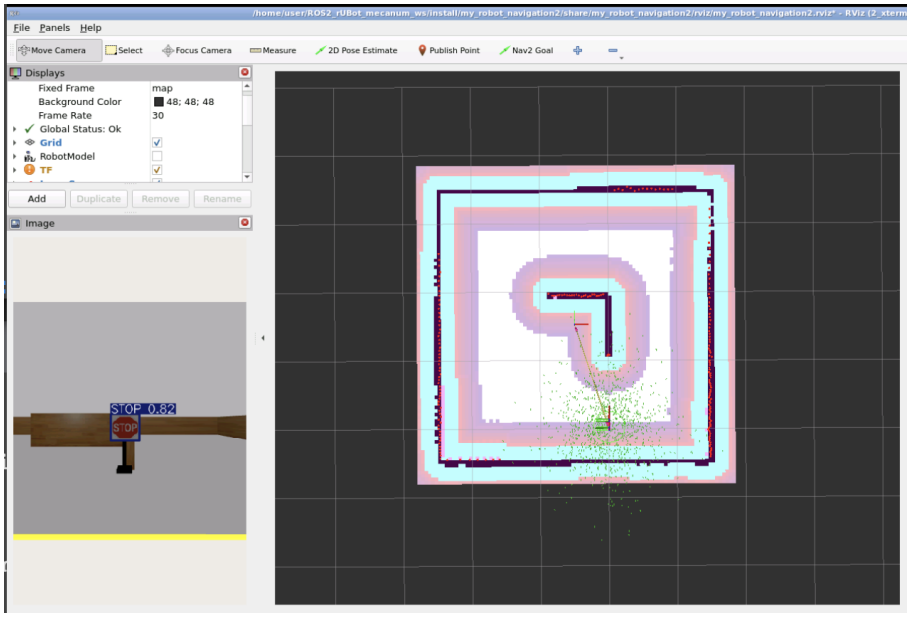

Figura 22: *Entorno de simulación con reacción ante señal STOP (TheConstruct)*

7. Conclusiones y trabajo futuro

Este proyecto ha tenido como propósito fundamental el estudio, desarrollo e implementación de un sistema de navegación autónoma y percepción visual, aplicando técnicas de inteligencia artificial sobre una plataforma robótica. Para ello, se ha utilizado el robot LIMO como base experimental y se han desplegado los distintos componentes mediante una arquitectura modular en ROS2, facilitando su integración mediante contenedores Docker y validando previamente su funcionamiento en el entorno simulado de The Construct.

7.1. Evaluación de objetivos

Los objetivos planteados al inicio del proyecto se han alcanzado en su mayoría:

- **Dockerización del sistema:** se ha conseguido empaquetar todas las dependencias y componentes necesarios en contenedores, permitiendo su portabilidad y replicabilidad tanto en simulación como en el robot físico.
- **Desarrollo funcional con sensores LiDAR:** se han implementado tres módulos clave: evitación de obstáculos, seguimiento de paredes y navegación basada en SLAM y waypoints, todos verificados en entornos virtuales y reales.
- **Integración de visión artificial:** se ha entrenado un modelo YOLOv8 personalizado y se ha desplegado en el sistema de navegación, permitiendo al robot reconocer señales de tráfico durante su recorrido.
- **Validación progresiva:** cada módulo ha sido validado de forma independiente y posteriormente integrado, ajustando sus parámetros y controlando el comportamiento general del sistema.
- **Estructura modular en ROS2:** el sistema se ha organizado en nodos especializados, facilitando su mantenimiento, prueba unitaria y futura escalabilidad.

Algunas limitaciones operativas han obligado a aplicar estrategias de contingencia. Por ejemplo, la baja frecuencia de inferencia en el entorno virtual fue mitigada mediante pruebas más específicas con inputs controlados y posterior comprobación en el robot real. Asimismo, algunos procesos de inicialización requerían pasos manuales (como lanzamientos en RViz), lo que limitó la automatización completa. Estos elementos han sido tenidos en cuenta como puntos de mejora en el análisis posterior.

7.2. Análisis crítico

- **Fortalezas:**
  - La integración entre percepción y navegación ha sido efectiva, permitiendo al robot adaptarse al entorno y responder a estímulos visuales y espaciales.
  - La arquitectura del sistema, al ser modular, facilita futuras mejoras o sustitución de componentes.
  - La dockerización ha sido clave para asegurar la portabilidad del sistema entre entornos, minimizando errores por diferencias en configuraciones locales.
  - La metodología TDD ha ayudado a mantener una alta cohesión y control sobre el desarrollo, mejorando la fiabilidad general del código.
- **Limitaciones:**
  - La baja tasa de inferencia visual en simulación limitó los experimentos más dinámicos durante las primeras pruebas.
  - La integración de nodos aún requiere secuencias específicas de lanzamiento y no está completamente automatizada.
  - El modelo de planificación utilizado es determinista y no incorpora planificación dinámica ni aprendizaje adaptativo.
  - El conjunto de datos visuales no contempla condiciones adversas o de baja visibilidad.

7.3. Líneas de trabajo futuro

Este trabajo abre múltiples líneas de investigación y mejora técnica:

- **Optimización del rendimiento en tiempo real:** reestructurar los nodos de inferencia y aprovechar GPU externas permitiría reducir drásticamente la latencia de detección.
- **Ampliación del dataset:** introducir escenarios nocturnos, condiciones meteorológicas adversas y nuevos objetos (como peatones o vehículos) enriquecería el modelo y lo acercaría a casos de uso reales.
- **Planificación inteligente:** explorar mecanismos como árboles de comportamiento o planificación dinámica multiobjetivo permitiría un control más sofisticado de la navegación.
- **Despliegue en entornos más complejos:** probar el sistema en espacios exteriores o urbanos reales pondría a prueba su robustez y capacidad de adaptación.
- **Colaboración multi-robot:** implementar arquitecturas distribuidas donde varios robots compartan información (mapas, detecciones, estados) en tiempo real mediante ROS2 DDS.
- **Entorno simulado propio:** desarrollar una infraestructura de simulación personalizada basada en Gazebo o Ignition, con despliegue Dockerizado, permitiría mayor control sobre los escenarios de prueba y reduciría la latencia operativa frente a plataformas externas como The Construct.

7.4. Reflexión final

Este proyecto ha demostrado que es posible construir, desde cero, un sistema robótico inteligente y operativo combinando múltiples tecnologías avanzadas en un entorno académico. A través de la integración de visión artificial, navegación basada en SLAM, comunicación entre nodos mediante ROS2 y despliegue mediante contenedores Docker, se ha logrado implementar una solución funcional, modular y fácilmente replicable tanto en simulación como en el robot real.

Más allá de los logros técnicos, este trabajo ha supuesto un auténtico reto personal y profesional. El proceso no solo ha requerido conocimientos de programación y robótica, sino también habilidades de investigación, gestión del tiempo, resolución de problemas, y sobre todo, capacidad de adaptación constante ante imprevistos técnicos y limitaciones del entorno.

El uso de plataformas virtuales como The Construct, combinado con la transferencia del sistema al robot físico LIMO, ha aportado una visión muy cercana a los flujos de trabajo reales en entornos industriales o de investigación. El hecho de dockerizar cada componente ha sido clave para entender la importancia de la reproducibilidad, la portabilidad de soluciones y la robustez ante cambios en el entorno de ejecución.

Durante el desarrollo de este TFG también se han reforzado competencias transversales como la toma de decisiones fundamentadas, el análisis crítico del propio trabajo, la documentación clara y ordenada, y el trabajo autónomo dentro de un proyecto de alta complejidad. Ha sido, en definitiva, una experiencia que ha puesto a prueba no solo mis conocimientos, sino también mi capacidad para aprender, adaptarme y sacar adelante una solución técnica realista, funcional y aplicable.

Este proyecto no ha sido simplemente una demostración técnica, sino una validación personal de todo lo aprendido durante la carrera, proyectado hacia un escenario real y con una proyección directa hacia futuros retos profesionales.

Referencias

[1] AgileX Robotics, *LIMO Mobile Robot*. Disponible en: `https://www.agilex.ai/education/18`, consultado el 20 de septiembre de 2024.

[2] Apple Inc., *MacBook Pro de 14 pulgadas*. Disponible en: `https://www.apple.com/es/shop/buy-mac/macbook-pro`, consultado el 22 de septiembre de 2024.

[3] Glassdoor, *Salario medio de Ingeniero Junior en España*. Disponible en: `https://www.glassdoor.es/Sueldos/ingeniero-junior-sueldo-SRCH_KO0,16.htm`, consultado el 30 de septiembre de 2024.

[4] The Construct, *Planes y precios de suscripción*. Disponible en: `https://www.theconstructsim.com/pricing/`, consultado el 15 de noviembre de 2024.

[5] The Construct, *Teachable Machine*. Disponible en: `https://www.theconstruct.ai`, consultado el 1 de diciembre de 2024.

[6] Python Software Foundation, *Python Documentation*. Disponible en: `https://docs.python.org/3`, consultado el 12 de diciembre de 2024.

[7] Lentin Joseph, Jonathan Cacace, *ROS Robotics Projects, Packt Publishing, 2017*. consultado el 14 de diciembre de 2024.

[8] ROS2, *Documentación oficial ROS2*. Disponible en: `https://docs.ros.org/en/humble/index.html`, consultado el 14 de diciembre de 2024.

[9] Nigel Poulton, *Docker Deep Dive, Independently published, 2020*. consultado el 22 de diciembre de 2024.

[10] ROS2, *Middleware DDS ROS2*. Disponible en: `https://docs.ros.org/en/humble/Concepts/About-DDS-and-ROS2.html`, consultado el 15 de enero de 2025.

[11] Open Robotics, *Gazebo Simulator*. Disponible en: `https://gazebosim.org`, consultado el 18 de enero de 2025.

[12] Docker Inc., *Docker Documentation*. Disponible en: `https://docs.docker.com`, consultado el 18 de enero de 2025.

[13] ROS Wiki, *RViz*. Disponible en: `https://wiki.ros.org/rviz`, consultado el 25 de enero de 2025.

[14] ROS2, *colcon documentation*. Disponible en: `https://colcon.readthedocs.io/en/released/`, consultado el 30 de enero de 2025.

[15] Aaron Martinez Romero, Enrique Fernández, *Learning ROS for Robotics Programming, Packt Publishing, 2015*. consultado el 8 de febrero de 2025.

[16] Enrique Fernández, et al., *Learning ROS for Robotics Programming - Second Edition, Packt Publishing, 2018*. consultado el 10 de febrero de 2025.

[17] Google Cartographer, *Google Cartographer documentación*. Disponible en: https://google-cartographer-ros.readthedocs.io, consultado el 20 de febrero de 2025.

[18] Docker Hub, *Imagen de Docker*. Disponible en: https://hub.docker.com/r/theconstructai/limo, consultado el 22 de febrero de 2025.

[19] Navigation2, *ROS2 Navigation Stack*. Disponible en: https://navigation.ros.org, consultado el 25 de febrero de 2025.

[20] César Cadena, et al., *Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age, IEEE Transactions on Robotics, 2016*. consultado el 1 de marzo de 2025.

[21] Ultralytics, *Documentación oficial YOLOv8*. Disponible en: https://docs.ultralytics.com, consultado el 15 de marzo de 2025.

[22] Joseph Redmon, et al., *You Only Look Once: Unified, Real-Time Object Detection, IEEE CVPR, 2016*. consultado el 22 de marzo de 2025.

[23] Ultralytics, *YOLOv8 Technical Report, Ultralytics, 2023*. consultado el 30 de marzo de 2025.

[24] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning, MIT Press, 2016*. consultado el 1 de abril de 2025.

[25] Roboflow, *Plataforma de etiquetado para entrenamiento de modelos YOLO*. Disponible en: https://roboflow.com, consultado el 5 de abril de 2025.

[26] Richard Szeliski, *Computer Vision: Algorithms and Applications, Springer, 2010*. consultado el 10 de abril de 2025.

[27] TensorFlow, *Keras functional API*. Disponible en: https://www.tensorflow.org/guide/keras/functional?hl=es-419, consultado el 12 de abril de 2025.

[28] YouTube, *Curso de YOLO*. Disponible en: https://www.youtube.com/watch?v=svn9-xV7wjk, consultado el 15 de abril de 2025.

[29] Berthold K.P. Horn, *Robot Vision, MIT Press, 1986*. consultado el 15 de abril de 2025.

[30] YouTube, *Clase de YOLO con ROS y Darknet*. Disponible en: https://www.youtube.com/watch?v=HRxPnfCAmRw, consultado el 20 de abril de 2025.

[31] Ultralytics, *Repositorio YOLOv8 GitHub*. Disponible en: https://github.com/ultralytics/ultralytics, consultado el 25 de abril de 2025.

[32] TeachableMachine, *Teachable Machine*. Disponible en: https://teachablemachine.withgoogle.com, consultado el 20 de mayo de 2025.

Anexo

El manual de uso completo del sistema puede consultarse en el repositorio público del proyecto en GitHub, disponible en la siguiente dirección:

- `https://github.com/Mattyete/ROS2_LIMO_ws/blob/main/Documentation/LIMO_Manual.md`

Este mismo documento también se encuentra incluido en el código fuente entregado, accesible desde la ruta local:

- `/Documentation/LIMO_Manual.md`

Este manual proporciona instrucciones detalladas sobre cómo lanzar el entorno de simulación y el robot real, así como el uso de los distintos nodos implementados, su configuración y los parámetros relevantes para su correcto funcionamiento.