



Escuela
Politécnica
Superior

SLAM con el robot articulado Unitree Go2



Máster Universitario en Inteligencia
Artificial

Trabajo Fin de Máster

Autor:

José Carlos Penalva Maciá

Tutor/es:

Félix Escalona Moncholí

Miguel Ángel Cazorla Quevedo

Carlos Zambrana Navajas



Universitat d'Alacant
Universidad de Alicante

Septiembre 2025

SLAM con el robot articulado Unitree Go2

Autor

José Carlos Penalva Maciá

Tutor/es

Félix Escalona Moncholí

Departamento de Ciencia de la Computación e Inteligencia Artificial

Miguel Ángel Cazorla Quevedo

Departamento de Ciencia de la Computación e Inteligencia Artificial

Carlos Zambrana Navajas

Departamento de Ciencia de la Computación e Inteligencia Artificial



Máster Universitario en Inteligencia Artificial



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Septiembre 2025

Preámbulo

La motivación principal para el desarrollo de este proyecto reside en la oportunidad de integrar los conocimientos adquiridos durante mi formación en el Máster de Inteligencia Artificial a la base obtenida en mi anterior titulación, el grado en Ingeniería Robótica. Esta convergencia de competencias permite abordar desafíos complejos desde una perspectiva multidisciplinar, aprovechando tanto los fundamentos teóricos como las aplicaciones tecnológicas más avanzadas.

El presente Trabajo Final de Máster (TFM) surge del interés por explorar las relaciones entre ambos campos del conocimiento, con el propósito de generar soluciones innovadoras que puedan servir como fundamento para futuras investigaciones y desarrollos en el ámbito de la inteligencia artificial aplicada a la robótica. De este modo, se busca contribuir al avance del estado del arte, facilitando el camino para posteriores ampliaciones y mejoras de proyectos relacionados con esta área de estudio.

Agradecimientos

La realización de este trabajo no habría sido posible sin el apoyo y la colaboración de numerosas personas a las que deseo expresar mi más sincero agradecimiento.

En primer lugar, quiero agradecer profundamente a mi familia —padres, hermanos y abuelos— por su respaldo inquebrantable a lo largo de todo este proceso. Desde mi infancia me inculcaron la importancia de perseverar en mis pasiones y nunca rendirme ante las dificultades. Su apoyo incondicional ante todos los desafíos que la vida me ha presentado, así como su comprensión durante los momentos más difíciles, han constituido un pilar fundamental para el desarrollo y culminación de este proyecto.

Asimismo, deseo expresar mi especial gratitud a Félix Escalona, Miguel Ángel Cazorla y Carlos Zambrana, del Departamento de Ciencia de la Computación e Inteligencia Artificial, cuya orientación ha sido esencial para la realización de este trabajo. Su disposición para compartir conocimientos, proporcionar las tecnologías necesarias y resolver todas las dudas que surgieron durante el desarrollo del proyecto ha sido invaluable para alcanzar los objetivos propuestos.

Finalmente, quiero reconocer y agradecer a todos mis amigos y compañeros del departamento que han mostrado interés en mi trabajo y me han brindado su apoyo constante. Sus contribuciones, comentarios y aliento han sido determinantes para que este proyecto se desarrollara de la mejor manera posible.

A todos ellos, mi más profundo agradecimiento.

*A mis padres, María Asunción Maciá y Jesús Carlos Penalva,
a mis hermanos, Jesús Penalva y María Penalva,
y a mis abuelos, María Asunción, María Teresa y José Antonio,
por su apoyo incondicional y por enseñarme, cada día, a ser mejor.*

*Allí donde reinan
la quietud y la meditación,
no hay lugar para las
preocupaciones ni para
la disipación.*

Francisco de Asís.

Índice general

1	Introducción	1
2	Marco Teórico	5
2.1	Simultaneous Localization and Mapping (SLAM)	5
2.2	Detección de objetos con CNN de región y YOLOv8	7
2.3	Transformers en visión por computador y Grounding DINO	8
2.4	Aprendizaje de embeddings por pares texto-imagen	9
2.5	Proyectos relacionados	10
2.5.1	Navegación autónoma	10
2.5.2	SLAM en robots móviles	11
2.5.3	Redes neuronales profundas en robots móviles	12
3	Objetivos	15
4	Metodología	17
4.1	Python	17
4.2	Unitree GO2 Edu	17
4.2.1	Características	18
4.2.2	Conexión mediante WebRTC	20
4.3	ROS2	20
4.4	Rclpy	21
4.5	OpenCV	22
4.6	Matplotlib	22
4.7	YOLOv8	23
4.8	Grounding DINO	24
4.9	PyTorch	27
4.10	SLAM	27
4.10.1	SLAM Toolbox	28
4.11	LiDAR	29
4.11.1	Unitree LiDAR 4D L1	30
4.12	RViz	30
4.13	Cámara integrada	31
4.14	Repositorio go2_ros2_sdk	32
4.15	Tkinter	32
4.16	Numpy	33
4.17	TF2	34
5	Desarrollo	35
5.1	Conexión con el robot	36

5.2	Navegación autónoma	37
5.3	Mapeado	40
5.4	Detección de objetos	41
5.4.1	YOLOv8	42
5.4.2	Grounding DINO	45
5.5	Seguimiento de Objetos	47
5.5.1	YOLOv8	47
5.5.2	Grounding DINO	49
5.6	Interfaz	50
6	Resultados	55
6.1	Entorno de pruebas	55
6.2	Experimentación	56
6.3	Resultados obtenidos	56
6.3.1	Navegación autónoma	56
6.3.2	Detección de objetos	58
6.3.3	Seguimiento de objetos	63
6.3.4	Interfaz	66
7	Conclusiones	69
Bibliografía		71
Lista de Acrónimos y Abreviaturas		75

Índice de figuras

1.1	Robot cuadrúpedo en aplicación industrial	1
1.2	Diferentes robots cuadrúpedos	2
2.1	Introducción a SLAM	6
2.2	R-CNN: Regiones con características CNN	7
2.3	Funcionamiento general de los transformers en visión	8
2.4	Diagrama algoritmos de navegación	10
2.5	Diagrama de localización y mapeo	11
2.6	Señalización de la detección de una persona	13
4.1	Logo Oficial Python	17
4.2	Robot Unitree Go2 Edu	18
4.3	Características Unitree Go2 Edu	19
4.4	Logo Oficial ROS2	21
4.5	Ejemplo Detección YOLOv8	24
4.6	Funcionamiento General Grounding DINO	25
4.7	Ejemplo Grounding DINO	26
4.8	Visualización del mapa en RViz	28
4.9	Unitree LiDAR 4D L1	30
4.10	Visualización RViz	31
5.1	Esquema visual del proyecto	35
5.2	Selección de modo en la aplicación Unitree	36
5.3	Visualización launch en RViz	37
5.4	Rangos LiDAR	38
5.5	Diagrama de flujo navegación autónoma	39
5.6	Mapa creado con slam_toolbox	41
5.7	Pipeline códigos detección de objetos	42
5.8	Introducción prompt para YOLOv8	43
5.9	Detección con YOLOv8	44
5.10	Introducción prompt para Grounding DINO	45
5.11	Detección con Grounding DINO	46
5.12	Controles básicos interfaz	51
5.13	Estado de procesos en la interfaz	52
5.14	Carga de mapas en la interfaz	52
5.15	Botones de cierre en la interfaz	53
6.1	Entorno de pruebas con obstáculos	55
6.2	Evitación de obstáculos	57
6.3	Comparativa de detección de personas	60

6.4	Comparativa de detección de sillas	61
6.5	Resultados de mapeado con detección de objetos	62
6.6	Ventanas mostradas	63
6.7	Pantalla de espera en YOLOv8	64
6.8	Ventanas de seguimiento de objetos	65
6.9	Interfaz gráfica final desarrollada para el control del robot	66

Índice de cuadros

6.1 Comparación entre YOLOv8 y Grounding DINO en la tarea de detección de objetos.	58
6.2 Resultados cuantitativos en la comparación de YOLOv8 y Grounding DINO.	59

1 Introducción

En las últimas décadas, la robótica ha evolucionado de ser un campo centrado en aplicaciones industriales preprogramadas, con robots destinados a entornos controlados y tareas repetitivas, a convertirse en un área interdisciplinaria en la que se combinan la Inteligencia Artificial (IA), la visión por computador, la ingeniería mecánica y la computación en tiempo real. Este progreso ha impulsado el desarrollo de sistemas cada vez más autónomos, capaces de desplazarse en entornos no estructurados, percibir su entorno con precisión y tomar decisiones en función de la información captada.



Figura 1.1: Robot cuadrúpedo en aplicación industrial
Obtenida de Invelon Technologies SL (2025)

El proyecto desarrollado tiene como objetivo integrar diversas tecnologías avanzadas para dotar a un robot cuadrúpedo articulado de capacidades de navegación autónoma en entornos desconocidos mientras realiza el mapeado de las zonas por las que se mueve. Por consiguiente, es fundamental introducir los conceptos de navegación autónoma, visión por computador y Simultaneous Localization and Mapping (SLAM) para comprender el funcionamiento de las distintas aplicaciones diseñadas para cumplir nuestro objetivo.

La navegación autónoma engloba el conjunto de procesos y tecnologías que permiten a un robot desplazarse de manera independiente, adaptándose a condiciones cambiantes y evitando obstáculos. Normalmente se estructura en tres fases:

- Percepción: el robot capta y procesa información del entorno mediante sensores.
- Planificación: se determinan trayectorias seguras y eficientes en función de objetivos y restricciones.
- Control: ejecuta los movimientos planificados manteniendo la estabilidad y corrigiendo

desviaciones en tiempo real.

Dentro de la fase de percepción, la visión por computador juega un papel fundamental. Esta disciplina agrupa un conjunto de técnicas y algoritmos que permiten a un sistema adquirir imágenes del mundo real, procesarlas y extraer información útil. En robótica móvil, su uso es clave para la detección y clasificación de objetos, seguimiento de elementos en movimiento y estimación de profundidad. Gracias a las redes neuronales profundas y librerías para clasificación de imágenes, la visión por computador puede ejecutarse en tiempo real, reduciendo el coste computacional y minimizando errores de detección incluso en entornos complejos. En este proyecto, esta tecnología permite identificar y localizar elementos relevantes del entorno, almacenando sus posiciones dentro del mapa generado por el robot.

El SLAM constituye otro elemento central de la percepción, ya que permite al robot estimar su posición y construir un mapa del entorno de manera simultánea. Para lograrlo, se integran datos de múltiples sensores como LiDAR, cámaras o unidades iniciales, gestionando la incertidumbre y optimizando los cálculos en tiempo real para equilibrar precisión y eficiencia. Gracias a los avances recientes en procesamiento embebido, sensores de alta precisión y algoritmos de IA, el SLAM se ha convertido en un pilar fundamental de sistemas autónomos empleados en exploración espacial, operaciones de rescate, agricultura de precisión, logística y transporte inteligente.

En este contexto, el presente TFM aborda la implementación y evaluación de un sistema avanzado de navegación y percepción en el robot cuadrúpedo articulado Unitree Go2. A diferencia de las plataformas móviles con ruedas, los robots cuadrúpedos ofrecen una movilidad superior en terrenos irregulares, capacidad para superar obstáculos y mayor estabilidad en entornos complejos. Estas características, combinadas con la arquitectura sensorial y computacional del Unitree Go2, lo convierten en una plataforma idónea para integrar y evaluar soluciones avanzadas de SLAM, visión por computador y control autónomo. La aportación principal de este trabajo radica en la integración coordinada de estas funcionalidades sobre una plataforma robótica real, evaluando su rendimiento en escenarios reales y analizando cómo la combinación de percepción, mapeo y navegación influye en la eficacia global del sistema.



Figura 1.2: Diferentes robots cuadrúpedos
Obtenida de Invelon Technologies SL (2025)

Finalmente, a lo largo de este trabajo, se tratará de informar acerca del proceso de investigación y desarrollo de los diferentes objetivos. En primer lugar, en el Capítulo 2 se mostrarán diferentes trabajos ya realizados que se relacionan con el tema principal. El Capítulo 3 expondrá los objetivos a cumplir durante la realización de este proyecto. En el Capítulo 4 se apuntarán todas las herramientas necesarias para cumplir los objetivos del capítulo anterior. A continuación, en el Capítulo 5, describirá la utilidad de cada herramienta para las aplicaciones creadas, además de mostrar las características más concretas del proyecto. En el Capítulo 6, se expondrán los resultados obtenidos y las diferentes pruebas que se han realizado con el objetivo de mejorar estos resultados. Más tarde, en el Capítulo 7, se extraerán conclusiones basadas en los resultados obtenidos a lo largo del estudio. Finalmente, se mostrará la bibliografía necesaria para realizar el proyecto junto con los acrónimos y abreviaturas.

2 Marco Teórico

A lo largo de los años, los avances en los algoritmos de localización, la percepción del entorno y las técnicas de inteligencia artificial han transformado profundamente el campo de la robótica móvil. Este proceso de evolución ha convertido al SLAM en un área de investigación clave, ya que permite a los robots construir mapas del entorno mientras estiman su propia posición en tiempo real. La capacidad de desplazarse de manera autónoma en entornos dinámicos, evitando obstáculos y reconociendo elementos del entorno, representa un hito fundamental para el desarrollo de sistemas robóticos más inteligentes y versátiles.

La integración de visión artificial y aprendizaje profundo ha ampliado las posibilidades de la navegación autónoma, al dotar a los robots de la capacidad de identificar y clasificar objetos y personas de forma eficiente. Estas tecnologías no solo mejoran la precisión de los sistemas de mapeo, sino que también permiten una interacción más segura y contextual con el entorno, lo que abre la puerta a aplicaciones en ámbitos industriales, de exploración, logísticos y de asistencia.

En esta sección del marco teórico, exploraremos las tecnologías y conceptos clave que han impulsado el desarrollo de sistemas SLAM en robots móviles, así como las investigaciones previas y los avances en este campo que han sentado las bases para el presente proyecto. Para ello, se abordarán tres ejes principales: la navegación autónoma, los enfoques de SLAM en robots móviles y las técnicas de detección y clasificación de objetos usando redes neuronales profundas.

2.1 Simultaneous Localization and Mapping (SLAM)

El Simultaneous Localization and Mapping (SLAM) aborda el problema de que un robot móvil construya un mapa del entorno mientras estima su propia posición en él. La dificultad radica en la dependencia mutua entre ambos procesos: el robot necesita un mapa coherente para localizarse, y a su vez requiere una localización precisa para generar un mapa fiable. Esta interdependencia convierte al SLAM en un desafío central dentro de la robótica. Durrant-Whyte y Bailey (2006).

Los primeros enfoques se basaron en el Filtro de Kalman Extendido (EKF), que asumía distribuciones gaussianas, y en filtros de partículas para manejar incertidumbres más generales. Más tarde, los métodos de optimización en grafos se consolidaron como una solución eficiente y escalable, formulando la trayectoria como una red de poses unidas por restricciones sensoriales y resolviendo el problema como mínimos cuadrados no lineales.

En cuanto a la representación del entorno, se ha pasado de mapas de características puntuales a mapas de ocupación, extendidos con la popularización del LiDAR. Actualmente, sensores como cámaras RGB-D o LiDAR 3D permiten reconstrucciones densas, mientras que los mapas semánticos añaden etiquetas de objetos para enriquecer la navegación.

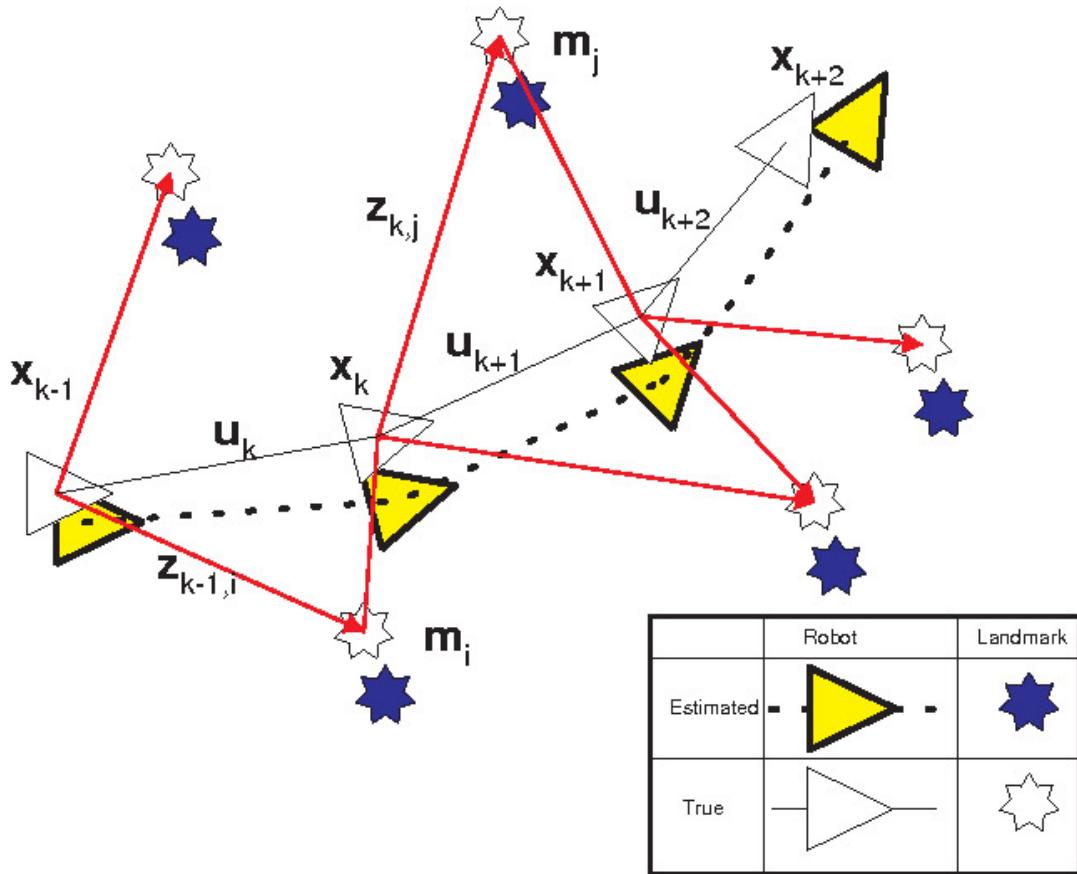


Figura 2.1: Introducción a SLAM
Obtenida de Durrant-Whyte y Bailey (2006)

Dos aspectos clave del SLAM son la asociación de datos, decidir qué observación corresponde a qué parte del mapa y el cierre de bucle, que corrige errores acumulados al reconocer lugares previamente visitados. Asimismo, la elección de sensores influye en las prestaciones: los sistemas visuales son económicos pero sensibles a la iluminación, mientras que los basados en LiDAR son más robustos aunque demandan mayor coste computacional.

En la actualidad, SLAM constituye la base de la navegación autónoma, evolucionando desde métodos probabilísticos hacia formulaciones gráficas y, más recientemente, incorporando aprendizaje profundo y semántica. Esto ha permitido desarrollar sistemas capaces de mapear la geometría y, a la vez, reconocer objetos, ampliando notablemente las capacidades de los robots móviles. Cadena y cols. (2016).

2.2 Detección de objetos con CNN de región y YOLOv8

La detección de objetos es un área fundamental en visión por computador que combina dos tareas: la clasificación de qué objeto aparece en una imagen y la localización de dicho objeto mediante una caja delimitadora. Los primeros enfoques modernos se apoyaron en Redes Neuronales Convolucionales (CNN), dando lugar a la familia de las Redes Neuronales Convolucionales de Región (R-CNN), que marcaron un antes y un después en la precisión de los sistemas de detección. Girshick y cols. (2014).

El modelo R-CNN original proponía regiones de interés mediante algoritmos selectivos, sobre las que se aplicaba una CNN para extraer características y posteriormente clasificar con un SVM. A pesar de su alta precisión, el proceso resultaba computacionalmente costoso. Fast R-CNN mejoró la eficiencia al compartir las convoluciones en toda la imagen y aplicar un agrupamiento de regiones de interés, reduciendo significativamente los tiempos de inferencia. Posteriormente, Faster R-CNN integró una Region Proposal Network (RPN), eliminando la dependencia de métodos externos para generar propuestas y alcanzando un equilibrio entre precisión y velocidad.

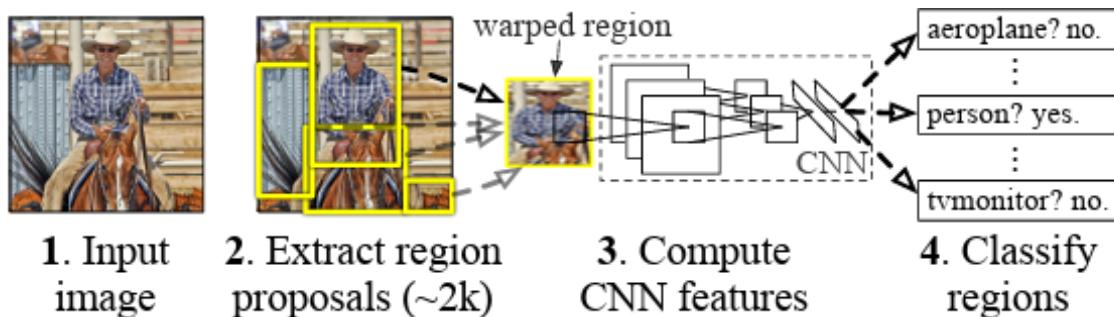


Figura 2.2: R-CNN: Regiones con características CNN
Obtenida de Girshick y cols. (2014)

Frente a estos enfoques de dos etapas, surgieron arquitecturas de detección de una sola etapa, entre las cuales destaca la familia You Only Look Once (YOLO). En lugar de separar la generación de regiones y la clasificación, YOLO reformula la detección como un problema de regresión directa sobre una cuadrícula, prediciendo simultáneamente las cajas delimitadoras y las probabilidades de clase en una única pasada de la red. Este enfoque permitió alcanzar velocidades en tiempo real, aunque inicialmente sacrificaba algo de precisión en comparación con Faster R-CNN. Redmon y cols. (2016).

Las versiones más recientes de YOLO han reducido esa brecha de precisión al tiempo que mantienen una gran eficiencia. En particular, YOLOv8 introduce mejoras sustanciales en el *backbone* mediante arquitecturas más ligeras, una cabeza de detección libre de *anchors* y técnicas de entrenamiento optimizadas para un equilibrio entre exactitud y velocidad. Gracias a estas características, YOLOv8 se ha convertido en una de las arquitecturas más utilizadas en aplicaciones prácticas, desde sistemas de vigilancia hasta robótica móvil, donde la capacidad de detección en tiempo real es crítica.

2.3 Transformers en visión por computador y Grounding DINO

Los transformers surgieron inicialmente en el campo del procesamiento de lenguaje natural como una arquitectura basada en mecanismos de autoatención, capaces de modelar dependencias a largo plazo en las secuencias de datos. Su éxito en tareas como la traducción automática impulsó su adopción en visión por computador, donde demostraron ser capaces de superar a las redes convolucionales tradicionales en varias tareas de clasificación y detección. La clave de esta arquitectura radica en el mecanismo de autoatención, que permite que cada elemento de la entrada, como un token de texto o un parche de imagen, se relacione con todos los demás, asignando pesos de atención que destacan las regiones más relevantes para la tarea. De esta forma, se captura de manera eficiente el contexto global de la imagen.

La adaptación de transformers a imágenes dio lugar a los Vision Transformer (ViT), en los que una imagen se divide en parches y se procesa como si se tratara de una secuencia de tokens. Este enfoque eliminó la necesidad de convoluciones y permitió aprender representaciones visuales más ricas y globales, aunque con un alto coste computacional. Posteriormente, surgieron variantes optimizadas que mejoraron la eficiencia y la escalabilidad de estos modelos, facilitando su aplicación en tareas de visión complejas. Alexey Dosovitskiy y Houlsby (2021).

En el ámbito de la detección de objetos, los transformers introdujeron un cambio de paradigma con modelos como DETR, que reformularon el problema como una tarea de predicción directa de cajas delimitadoras y clases a través de un mecanismo de atención. Esto permitió prescindir de las etapas tradicionales de generación de propuestas de regiones y anclas, simplificando la arquitectura a costa de mayores exigencias de entrenamiento. Sin embargo, el uso de transformers en detección abrió la puerta a integrar visión y lenguaje en un mismo marco.

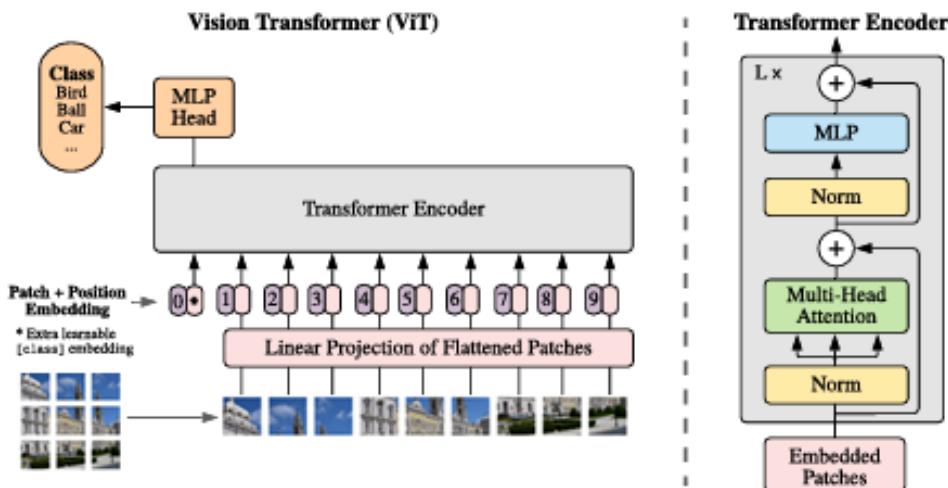


Figura 2.3: Funcionamiento general de los transformers en visión
Obtenida de Alexey Dosovitskiy y Houlsby (2021)

Sobre esta base, surgieron modelos más avanzados como Grounding DINO, que extienden las capacidades de los transformers hacia escenarios abiertos. A diferencia de los detectores tradicionales, limitados a un conjunto fijo de categorías predefinidas, Grounding DINO combina el poder de la atención visual con descripciones textuales libres, lo que le permite detectar y localizar objetos en función de instrucciones expresadas en lenguaje natural. De esta forma, el sistema no solo identifica instancias de categorías conocidas, sino que también puede adaptarse dinámicamente a nuevos conceptos sin necesidad de reentrenamiento específico. Esta característica lo convierte en una herramienta especialmente potente para aplicaciones donde la flexibilidad y la capacidad de generalización son críticas, como la robótica móvil, la interacción hombre-robot y la búsqueda de información visual en entornos no estructurados. Liu y cols. (2023).

2.4 Aprendizaje de embeddings por pares texto-imagen

El aprendizaje de embeddings por pares texto-imagen busca representar descripciones textuales e imágenes dentro de un mismo espacio vectorial, de forma que la proximidad entre vectores refleje su correspondencia semántica. Así, una imagen y su descripción deben generar vectores cercanos, mientras que las combinaciones incorrectas se mantienen alejadas. Este principio permite a los sistemas establecer relaciones complejas entre información visual y lingüística, habilitando tareas como la búsqueda de imágenes a partir de texto, la clasificación basada en descripciones o la detección de objetos condicionada por instrucciones verbales.

El procedimiento habitual consiste en emplear dos redes neuronales: una encargada de procesar las imágenes, generalmente una CNN o un ViT, y otra para el texto, a menudo un transformer de lenguaje. Ambas proyectan sus entradas a embeddings de dimensión fija, entrenados mediante funciones de pérdida contrastiva. Esta pérdida fuerza que los pares correctos se acerquen en el espacio vectorial y los incorrectos se distancien, reforzando así la alineación semántica.

Los embeddings cumplen un papel clave no solo por su capacidad de capturar significados, sino también por su eficiencia, ya que reducen la dimensionalidad de los datos y proporcionan representaciones densas que facilitan el procesamiento y el almacenamiento. Además, permiten operaciones vectoriales útiles para modelar relaciones entre conceptos, lo cual se extiende naturalmente al contexto multimodal. OpenWebinars (2023).

Modelos recientes como CLIP y Grounding DINO aplican este enfoque para permitir la detección abierta de objetos a partir de descripciones textuales. Gracias al aprendizaje de embeddings conjuntos, estos sistemas pueden reconocer conceptos no vistos durante el entrenamiento y adaptarse a nuevas categorías de forma flexible, ampliando de manera significativa las capacidades de la visión por computador en ámbitos como la robótica, la recuperación de información y las aplicaciones multimodales.

2.5 Proyectos relacionados

2.5.1 Navegación autónoma

El primer proyecto estrechamente relacionado con nuestro trabajo es el desarrollado por García Ulloa y Nieves Guerrero (2024), en el que se diseñó e implementó un robot móvil autónomo con un sistema de mapeo y detección de obstáculos basado en un sensor LiDAR. El objetivo de este proyecto era dotar a la plataforma de la capacidad de desplazarse de manera independiente en entornos interiores, explorando el espacio, evitando colisiones y alcanzando destinos previamente definidos sin intervención humana.

En lo referente a la planificación de trayectorias, el sistema integraba el algoritmo Rapidly-exploring Random Tree (RRT), ampliamente utilizado en navegación robótica por su capacidad de explorar eficientemente el espacio de búsqueda. Este planificador generaba rutas factibles desde un punto de partida hasta un objetivo definido, ajustando dinámicamente la trayectoria cuando aparecían obstáculos en el camino. Gracias a esta integración, el robot podía desplazarse de forma autónoma entre distintas ubicaciones del entorno, corrigiendo su trayectoria y garantizando un movimiento seguro incluso en escenarios parcialmente desconocidos.

Durante la fase experimental, se llevaron a cabo pruebas en entornos interiores controlados donde el robot debía explorar el espacio, construir un mapa del mismo y alcanzar puntos de destino específicos. Los resultados mostraron que el sistema era capaz de evitar colisiones en tiempo real, modificar su trayectoria ante la aparición de obstáculos imprevistos y completar satisfactoriamente las misiones de desplazamiento. Los autores concluyeron que la combinación de percepción LiDAR, algoritmos de SLAM y planificación con RRT constituye una solución robusta para la navegación autónoma de robots móviles. No obstante, su implementación se limitó a escenarios controlados en interiores, lo que abre la puerta a nuevas investigaciones que extiendan este enfoque a contextos más complejos. En este sentido, el trabajo aporta una base metodológica valiosa para nuestro proyecto, en el que buscamos trasladar dichas capacidades al caso de un robot articulado.



Figura 2.4: Diagrama algoritmos de navegación
Obtenida de García Ulloa y Nieves Guerrero (2024)

En resumen, el trabajo de García Ulloa y Nieves Guerrero (2024) constituye un aporte significativo al desarrollo de robots móviles autónomos en entornos interiores, integrando percepción LiDAR, algoritmos de SLAM y planificación con RRT en un marco ROS. Aunque su enfoque se limita a robots de arquitectura diferencial y a escenarios controlados, sus

resultados validan la robustez de esta combinación tecnológica.

2.5.2 SLAM en robots móviles

Otro trabajo relevante para nuestro proyecto es el desarrollado por Ricardo Urvina Córdova y Álvaro Prado Romo (2023), en el que se propone un enfoque de SLAM basado en el procesamiento de nubes de puntos generadas por un sensor LiDAR, combinando métodos de aprendizaje de máquina no supervisados con un Filtro de Kalman Extendido (EKF). El objetivo principal era dotar al robot móvil de la capacidad de localizarse y mapear su entorno en escenarios donde el acceso a sistemas de posicionamiento global es limitado o inexistente, como en minas subterráneas o interiores industriales.

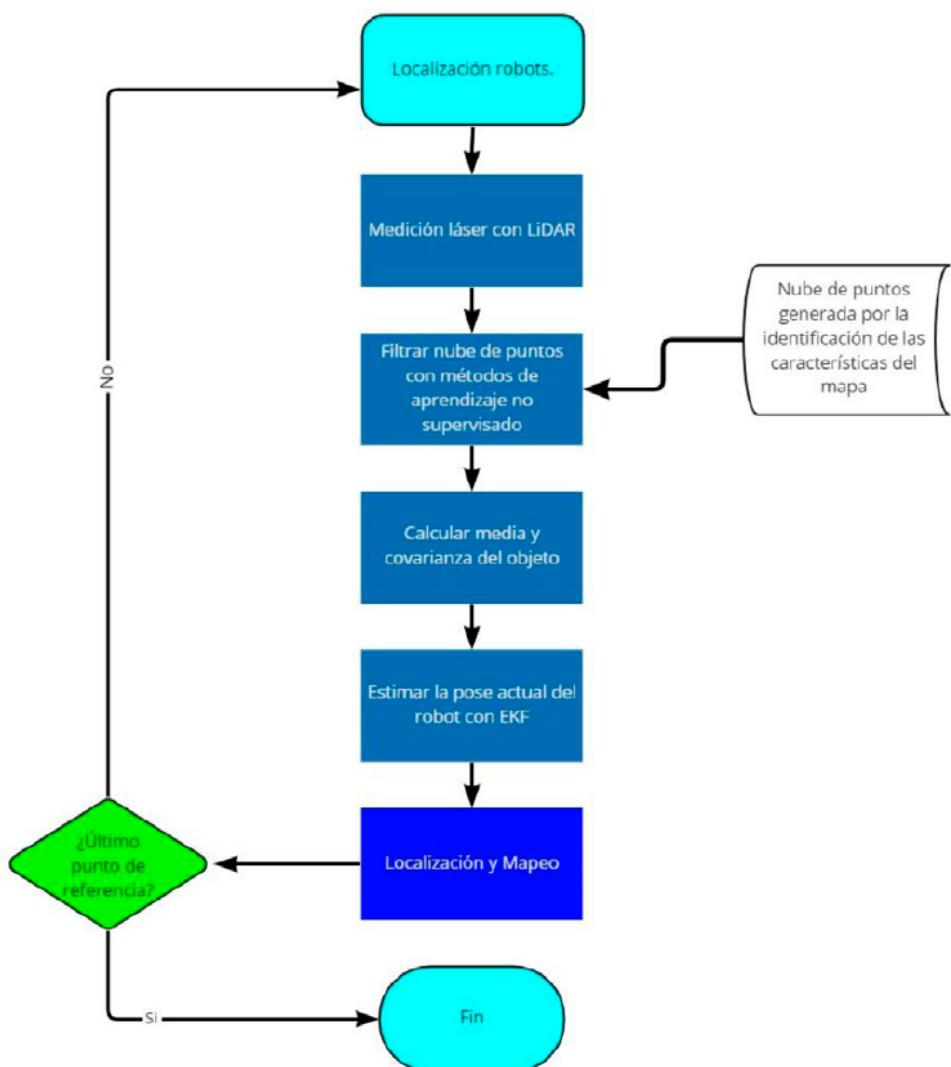


Figura 2.5: Diagrama de localización y mapeo
Obtenida de Ricardo Urvina Córdova y Álvaro Prado Romo (2023)

La metodología consistió en implementar cuatro enfoques distintos de agrupamiento de datos: un algoritmo heurístico basado en distancias, K-Means, Modelos de Mezcla Gaussiana (GMM) y DBSCAN. Estos algoritmos se emplearon para segmentar y caracterizar las características del entorno a partir de la nube de puntos, mientras que el EKF integraba la información de los sensores para estimar de forma robusta la pose del robot a lo largo del tiempo. El uso de estas técnicas permitió manejar de forma más efectiva la incertidumbre inherente a los datos de odometría y percepción.

Los resultados experimentales mostraron que, entre los métodos evaluados, DBSCAN ofreció un mejor rendimiento tanto en la precisión de la localización como en el tiempo de ejecución, reduciendo en aproximadamente un 5% el error acumulado en comparación con los otros algoritmos probados. Además, la integración con EKF demostró ser fundamental para corregir errores de odometría y garantizar la consistencia del sistema de localización, incluso en situaciones donde el sensor LiDAR no detectaba referencias temporales en el entorno.

En conclusión, el trabajo de Ricardo Urvina Córdova y Álvaro Prado Romo (2023) aporta una propuesta innovadora al combinar técnicas de clusterización no supervisadas con estimación probabilística, ofreciendo una alternativa robusta para la localización y el mapeo de robots móviles en entornos complejos y sin GPS. Esta aproximación representa un punto de partida relevante para investigaciones que buscan extender las capacidades de SLAM hacia plataformas más avanzadas.

2.5.3 Redes neuronales profundas en robots móviles

Otro proyecto estrechamente relacionado con nuestro trabajo es el desarrollado por Gordon y cols. (2019), en el que se combinó un sistema de SLAM con un detector de objetos basado en redes neuronales profundas para la navegación autónoma de un robot móvil KUKA You-Bot. El objetivo principal de este trabajo era que el robot no solo pudiera desplazarse en un entorno evitando obstáculos, sino también que fuese capaz de reconocer e incorporar información semántica al mapa generado durante la navegación. De esta forma, el robot obtenía una representación más rica del espacio, integrando tanto la geometría como la identificación de objetos presentes en él.

En cuanto a la arquitectura del sistema, se utilizó el paquete *Gmapping* de ROS como núcleo para la localización y el mapeo, mientras que la detección de objetos se realizó mediante el algoritmo *Single Shot Multibox Detector* con arquitectura *VGG*, entrenado sobre la librería Caffe e implementado en Python con el apoyo de OpenCV. La información proveniente del sensor LiDAR y de la cámara integrada en el robot se procesaba de manera complementaria: el LiDAR se encargaba de capturar la geometría del entorno, mientras que las imágenes de la cámara eran analizadas por la red neuronal para identificar objetos con diferentes niveles de confianza. Cuando se lograba una detección fiable, las coordenadas aproximadas del objeto se estimaban combinando la posición del robot con los datos del láser, y finalmente se representaban como marcas visuales en el mapa generado en RViz. Gracias a este procedimiento, el sistema era capaz de señalar elementos como personas, animales, mobiliario o señales de advertencia en el entorno.

Durante las pruebas realizadas en un entorno simulado con Gazebo y visualizado en RViz, se observó que el robot era capaz de desplazarse hacia metas definidas por el usuario, construir el mapa del espacio en tiempo real y, de manera simultánea, detectar y marcar los objetos encontrados durante su trayecto. El aporte más destacado radica en que el mapa generado no se limitaba a la representación geométrica, sino que incorporaba información semántica útil para enriquecer las decisiones de navegación futuras. No obstante, los autores identificaron una limitación relevante, la dificultad de proyectar detecciones realizadas en imágenes 2D sobre coordenadas reales del mapa. Este problema fue parcialmente mitigado utilizando las medidas del LiDAR, aunque aún persistía cierta incertidumbre en la localización exacta de los objetos detectados.

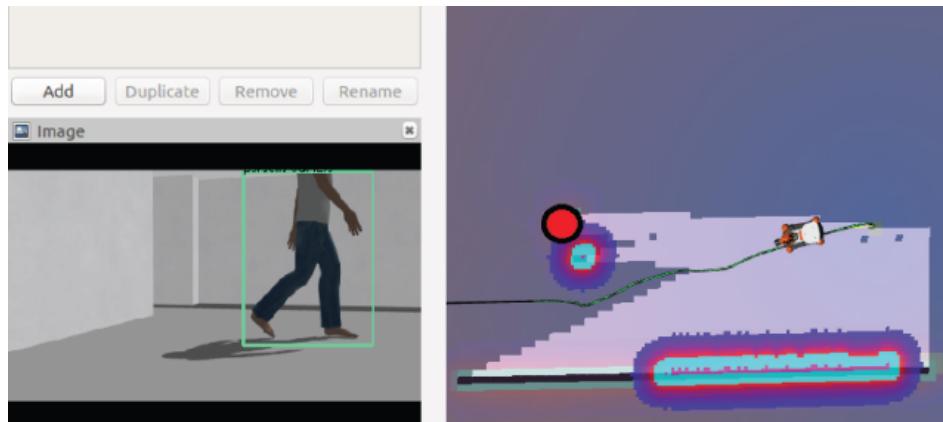


Figura 2.6: Señalización de la detección de una persona
Obtenida de Gordon y cols. (2019)

En conclusión, el trabajo de Gordon y cols. (2019) constituye un claro ejemplo de cómo la integración de redes neuronales profundas con algoritmos de SLAM amplía las capacidades de los robots móviles, pasando de mapas puramente geométricos a representaciones más completas y semánticas. En la Figura 2.6 se puede ver como el sistema realiza el mapeado y en caso de detectar un objeto lo marca en el mapa.

3 Objetivos

Para realizar este Trabajo Final de Máster (TFM) se propone el desarrollo de un sistema autónomo de navegación y mapeado para el robot Unitree Go2, integrando técnicas de visión artificial y aprendizaje profundo para el reconocimiento de objetos en entornos dinámicos. La investigación se centrará en la creación de un sistema Simultaneous Localization and Mapping (SLAM) eficiente, que permita al robot construir mapas precisos del entorno mientras se desplaza. Además, se implementarán modelos de detección, clasificación y seguimiento de objetos, utilizando redes neuronales profundas para el procesamiento de imágenes obtenidas a través de las cámaras del robot.

Por lo que los objetivos de este proyecto vienen definidos por:

- Diseñar e implementar un sistema SLAM eficiente.
- Desarrollar modelos de detección, clasificación y seguimiento de objetos mediante redes neuronales profundas.
- Integrar en un mismo sistema la navegación autónoma, el mapeado y la percepción visual.
- Crear una interfaz gráfica que facilite la interacción y el control del robot.

Estos objetivos se pueden concretar en:

1. Implementar un prototipo funcional en el robot Unitree Go2, capaz de navegar de manera autónoma en entornos desconocidos mientras se crea el mapa de la zona por donde avanza el robot.
2. Validar la detección, clasificación y seguimiento de objetos en tiempo real, evaluando el comportamiento del robot en función de sus resultados.
3. Desarrollar una interfaz gráfica intuitiva que permita gestionar de manera sencilla la navegación y la percepción del robot.

Finalmente, como objetivo personal, este proyecto permitirá profundizar en conceptos clave de la robótica autónoma, tales como:

- Aplicar y reforzar conocimientos de visión por computador y aprendizaje profundo en un caso práctico real.
- Explorar la integración entre algoritmos de SLAM y técnicas de percepción para robots móviles.

- Desarrollar competencias en el uso de plataformas robóticas avanzadas como Unitree Go2, con vistas a su aplicación en investigación y en cualquier tipo de entorno.
- Familiarizarse con el diseño e implementación de interfaces gráficas aplicadas a sistemas robóticos.

4 Metodología

La realización de este proyecto ha brindado la oportunidad de explorar y aplicar diversas tecnologías y metodologías, algunas de ellas previamente desconocidas. En este capítulo se describen en detalle las herramientas, técnicas y enfoques utilizados durante el desarrollo del presente TFM.

4.1 Python

Entre los diversos lenguajes de programación disponibles en la actualidad, se ha optado por utilizar Python para el desarrollo principal del proyecto, debido a su amplia adopción en ámbitos como el desarrollo de software, las aplicaciones web, la ciencia de datos y, especialmente, el aprendizaje automático.

Python es un lenguaje de programación de código abierto, interpretado y orientado a objetos, que destaca por su sintaxis clara y legible. Cuenta con una extensa biblioteca estándar y un ecosistema muy consolidado, lo que facilita el uso de múltiples librerías especializadas para tareas de visión por computador, interfaces gráficas y robótica. Actualmente, se considera uno de los lenguajes de programación más utilizados a nivel mundial. Python (2023).



Figura 4.1: Logo Oficial Python
Obtenida de Python (2023)

4.2 Unitree GO2 Edu

El Unitree Go2 es un robot cuadrúpedo de última generación desarrollado por Unitree Robotics, diseñado para ofrecer una combinación equilibrada entre agilidad, estabilidad y capacidades de percepción avanzada. El robot cuenta con 12 grados de libertad, distribuidos en tres por cada extremidad, y un sistema de control de fuerza en las articulaciones que le

permite adaptarse de forma dinámica a distintos tipos de terreno y obstáculos. Gracias a su diseño compacto, resistencia mecánica y algoritmos de control de locomoción, el Go2 es capaz de operar en entornos tanto interiores como exteriores, manteniendo un desplazamiento estable incluso en superficies irregulares.

La serie Go2 incluye tres versiones: Go2 Air, Go2 Pro y Go2 Edu. Estas variantes se diferencian principalmente en el tipo de procesador, la capacidad de cálculo embarcada y la dotación de sensores que integran, lo que las orienta a diferentes perfiles de uso, desde aplicaciones recreativas y de inspección hasta investigación avanzada y desarrollo de inteligencia artificial.



Figura 4.2: Robot Unitree Go2 Edu
Obtenida de Robotics (2025)

4.2.1 Características

En el presente proyecto se ha empleado el modelo Go2 Edu, versión orientada a educación e investigación, que incorpora una amplia gama de módulos y dispositivos diseñados para potenciar sus capacidades. Entre ellos se encuentra un módulo de seguimiento, que permite operar en modo remoto o automático, así como una cámara frontal con resolución de transmisión de 1280×720 píxeles, campo de visión de 120° y lente gran angular para ofrecer una visión clara y detallada. El robot también dispone de una lámpara frontal que ilumina eficazmente el camino y un micrófono con sistema de intercomunicación, pensado para mantener una comunicación fluida sin restricciones de entorno.

Para facilitar su transporte, el Go2 Edu incorpora una correa autorretráctil, mientras que su conectividad incluye tarjeta eSIM 4G, WiFi6 de doble banda y Bluetooth 5.2 para man-

tener una conexión estable y un control remoto fiable. En el núcleo de procesamiento, el robot cuenta con un controlador de movimiento y un procesador ARM de alto rendimiento, junto con un procesador optimizado para algoritmos de inteligencia artificial y soporte para módulos externos como ORIN NX/NANO.

En cuanto a percepción y navegación, incorpora un LiDAR 4D L1 con escaneo omnidiereccional de $360^\circ \times 90^\circ$, que permite evitar obstáculos de forma automática con una mínima zona ciega, y doce motores en las articulaciones, potentes y precisos, que garantizan una locomoción estable. Dispone además de sensores de fuerza en las patas para medir en tiempo real la presión ejercida en el terreno, y una batería de larga duración de 15000 mAh, con sistemas de protección frente a sobrecalentamiento, sobrecarga y cortocircuitos. Por último, cuenta con un altavoz integrado que permite reproducir audio o música según las necesidades del usuario. Robotics (2025).

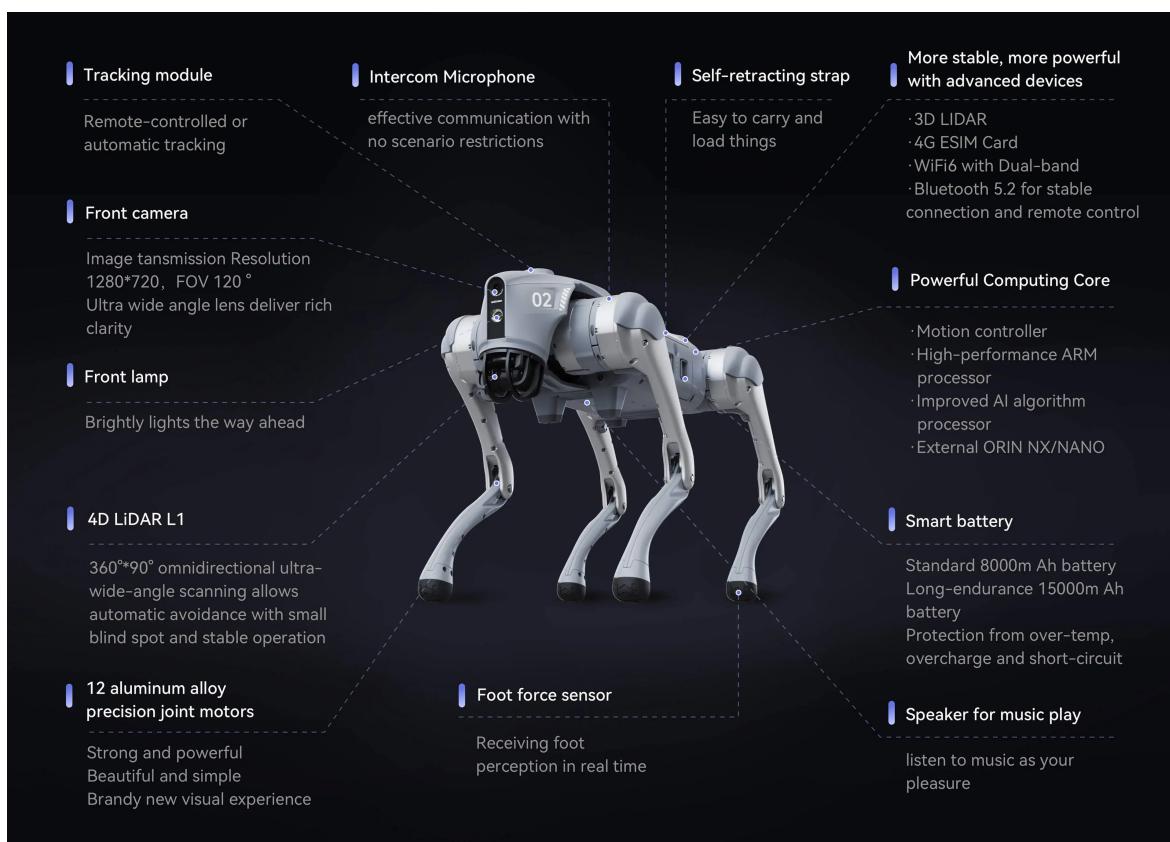


Figura 4.3: Características Unitree Go2 Edu
Obtenida de Robotics (2025)

4.2.2 Conexión mediante WebRTC

Web Real-Time Communication (WebRTC) es un conjunto de protocolos y APIs de código abierto diseñado para permitir la transmisión de audio, vídeo y datos en tiempo real entre dos o más dispositivos conectados a través de Internet. Su principal característica es que establece enlaces directos entre pares, reduciendo la latencia y evitando que el flujo principal de datos pase por servidores intermedios, lo que lo hace ideal para aplicaciones que requieren respuesta inmediata, como videollamadas, teleoperación o monitorización remota. Google WebRTC Team (2025).

El funcionamiento básico de WebRTC puede entenderse como un sistema que primero establece un canal de comunicación directo entre los dispositivos y, una vez conectado, envía los datos sin pasar por intermediarios. Para conseguirlo, se apoya en tres elementos clave:

1. Canales de medios, que transmiten audio y vídeo en tiempo real, asegurando que la imagen y el sonido lleguen de forma fluida.
2. Canales de datos, que permiten enviar información estructurada o mensajes de control casi de inmediato, como si fuera una conversación en directo.
3. Protocolos de seguridad, como Datagram Transport Layer Security (DTLS) y Secure Real-time Transport Protocol (SRTP), que se encargan de cifrar toda la comunicación para que los datos no puedan ser interceptados o alterados. 3CX (2025).

En este proyecto, WebRTC ha sido el medio de comunicación permanente entre el robot Unitree Go2 y el sistema desde el que se lanzaron todos los procesos implementados. A través de esta conexión se transmitieron la señal de vídeo de la cámara frontal, los datos generados por los sensores y el estado operativo del robot, al mismo tiempo que se enviaban desde el sistema de control las instrucciones y nodos de ROS2 necesarios para la ejecución del SLAM y otras tareas.

Esta arquitectura permitió operar el Go2 de manera remota y en tiempo real, manteniendo una sincronización constante entre el *software* embarcado y el sistema de control, incluso en redes inalámbricas no dedicadas. Gracias a la baja latencia inherente a WebRTC, fue posible tomar decisiones de navegación de forma inmediata, un factor crítico para el correcto funcionamiento del sistema.

4.3 ROS2

Robot Operating System (ROS) es un conjunto de bibliotecas de *software* y herramientas de código abierto que facilita la creación de aplicaciones robóticas complejas. Aunque no constituye un sistema operativo en sí mismo, proporciona servicios típicos de uno, como la abstracción del hardware, el control de dispositivos de bajo nivel, el paso de mensajes entre procesos, la implementación de funcionalidades de uso común y la gestión de paquetes. Su arquitectura se basa en un grafo distribuido de nodos que intercambian información mediante mensajes, lo que permite una separación clara entre los diferentes componentes del sistema

y fomenta la modularidad y reutilización del código.

ROS está compuesto por dos grandes partes: por un lado, el núcleo del sistema que maneja la comunicación entre procesos; y por otro, *ros-pkg*, una colección de paquetes desarrollados y compartidos por la comunidad, que implementan funcionalidades avanzadas como Simultaneous Localization and Mapping (SLAM), la planificación de trayectorias, la percepción visual, o la simulación. Gracias a esta estructura y al respaldo de una amplia comunidad de desarrolladores, ROS se ha consolidado como un estándar en el ámbito de la robótica, utilizado tanto en entornos académicos como industriales.

En este proyecto se ha trabajado sobre ROS2 Humble, una de las versiones con Soporte a Largo Plazo (LTS) de ROS2. Esta versión representa una evolución significativa respecto a la arquitectura original de ROS, al introducir un sistema de comunicación más robusto basado en Data Distribution Service (DDS), mayor soporte para sistemas distribuidos, mejoras en la gestión del tiempo real y una arquitectura más adecuada para entornos productivos. Humble ha sido especialmente diseñada para proporcionar estabilidad, compatibilidad a largo plazo y un entorno de desarrollo más maduro y seguro.

La documentación oficial de ROS2 Humble ha sido utilizada de forma sistemática durante todo el proceso de desarrollo. Esta ha servido como referencia principal para la configuración del entorno de trabajo, la creación y compilación de paquetes personalizados, la gestión de nodos, la implementación de sistemas de comunicación entre procesos, así como para la integración con herramientas como RViz para visualización. Open Robotics (2022).

El uso riguroso de esta documentación ha permitido garantizar una implementación alineada con los estándares actuales de la robótica, facilitar la reproducibilidad del sistema y establecer una metodología de desarrollo sólida, fundamentada técnicamente y coherente con los objetivos del proyecto.



Figura 4.4: Logo Oficial ROS2
Obtenida de Open Robotics (2022)

4.4 Rclpy

En el desarrollo de los módulos *software* del sistema se ha utilizado *rclpy*, que es la biblioteca cliente de ROS2 para el lenguaje de programación Python. Esta herramienta proporciona las interfaces necesarias para la creación de nodos y la gestión de los distintos mecanismos

de comunicación que ofrece ROS2, como la publicación y suscripción a tópicos, el uso de servicios y la ejecución de acciones. Open Robotics (2025).

Al estar escrita en Python, *rclpy* facilita una implementación más rápida y flexible de componentes de alto nivel, especialmente útil en etapas de desarrollo, prueba e integración de funcionalidades. Su uso ha permitido desarrollar nodos que interactúan con el resto del sistema de forma transparente, manteniendo la modularidad y escalabilidad que caracteriza a ROS2.

4.5 OpenCV

OpenCV es una biblioteca de visión por computador de *software* de aprendizaje automático y código abierto. OpenCV fue desarrollada por la compañía Intel para proporcionar una infraestructura común para aplicaciones de visión por computadora y acelerar el uso de la percepción artificial en los productos comerciales. En la actualidad, ha ganado gran popularidad debido a su extenso conjunto de funciones y por contar con un soporte multiplataforma para Linux, Windows, IOS, entre otros.

La biblioteca tiene más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de aprendizaje automático y visión por computadora, tanto clásicos como de última generación. Estos algoritmos se pueden utilizar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estéreo, unir imágenes para producir una alta resolución, a partir de una imagen de una escena completa, buscar imágenes similares en una base de datos de imágenes, eliminar los ojos rojos de imágenes tomadas con flash, seguir los movimientos oculares, reconocer paisajes, establecer marcadores para superponerlos con realidad aumentada, etc. OpenCV (2024).

En el marco de este trabajo, OpenCV se ha empleado específicamente para la detección de objetos y el seguimiento de su movimiento a lo largo del tiempo, con el fin de dotar al sistema de percepción visual en tiempo real. Estas funcionalidades han resultado esenciales para permitir que el sistema identifique elementos relevantes del entorno y reaccione de manera coherente ante su desplazamiento o aparición dentro del campo de visión.

4.6 Matplotlib

Matplotlib es una biblioteca de código abierto de Python diseñada para la creación de visualizaciones estáticas, animadas e interactivas de alta calidad en entornos de programación científicos y académicos. Al ser una biblioteca *open source*, ha tenido significativos avances, hasta el punto de que actualmente por medio de ella se pueden construir gráficos como líneas, histogramas, diagramas de barras, gráficos de dispersión, gráficos tridimensionales y más, con apenas unas líneas de código.

Matplotlib dispone de dos interfaces principales:

- Una API funcional, pyplot, que emula el estilo de MATLAB y es ideal para crear gráficos de manera rápida.
- Una API orientada a objetos, que proporciona un control más granular sobre los elementos visuales como figuras, ejes, leyendas o anotaciones, adecuada para aplicaciones más complejas o integradas en GUI.

Matplotlib se integra perfectamente con bibliotecas fundamentales como NumPy y pandas, facilitando la visualización directa de estructuras de datos comunes en ciencia y análisis de datos. Matplotlib (2025).

A lo largo de este proyecto se ha hecho uso de esta herramienta en los códigos que hacen uso de Grounding DINO para:

- Visualización del mapeado generado por SLAM: representa el mapa creado por SLAM con todos los obstáculos que se encuentra mientras marca los objetos detectados indicados por el usuario.
- Representación de detecciones: muestra los objetos o regiones detectadas a partir del prompt introducido mediante la GUI, ofreciendo un feedback visual inmediato sobre si la red ha interpretado correctamente la instrucción.

Para cada gráfico se han incluido títulos descriptivos, etiquetas de ejes, leyendas y anotaciones para garantizar una presentación clara y funcional. Esta estrategia permite que el equipo evalúe en tiempo real los resultados del sistema SLAM y la detección de objetos deseados desde una perspectiva visual, superando la interpretación mediante simples métricas o logs, y aporta una manera intuitiva para detectar posibles mejoras en cada iteración.

4.7 YOLOv8

Para la detección y seguimiento de objetos en este trabajo se ha empleado YOLOv8, la última versión de la familia You Only Look Once (YOLO) reconocida por su alta eficiencia y precisión en tareas de visión por computador. YOLOv8 es un modelo de detección de objetos de aprendizaje profundo desarrollado por Ultralytics, que combina arquitecturas avanzadas con técnicas optimizadas para lograr un rendimiento superior en tiempo real.

Este modelo utiliza una arquitectura de CNN que procesa imágenes completas en una sola pasada, permitiendo identificar múltiples objetos simultáneamente y en diferentes escalas. Además, YOLOv8 integra mejoras en el preprocesamiento, anclajes automáticos y técnicas de aumento de datos, lo que contribuye a una mayor robustez frente a variaciones en las condiciones de iluminación y entorno.

Debido a su diseño modular y su soporte para inferencia rápida en dispositivos con recursos limitados, YOLOv8 ha sido implementado en este proyecto para proporcionar capacidades avanzadas de detección de objetos en tiempo real. Su uso ha facilitado la identificación precisa y el seguimiento eficiente de los elementos de interés en las escenas analizadas, aportando una base sólida para las funcionalidades de percepción visual requeridas. Ultralytics (2025).

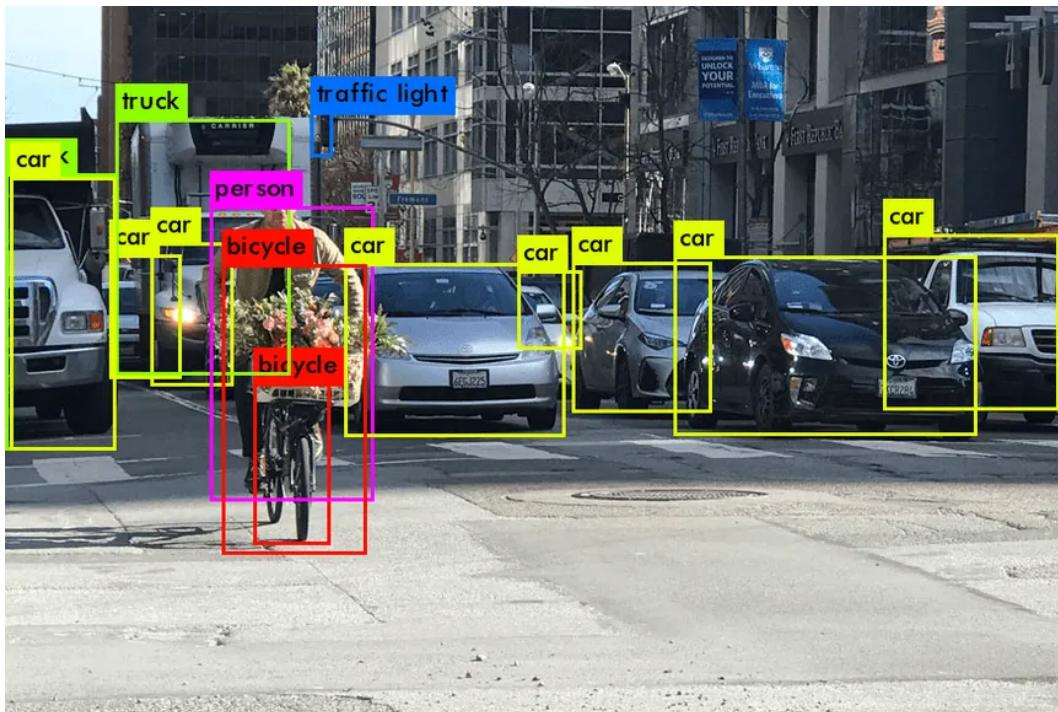


Figura 4.5: Ejemplo Detección YOLOv8
Obtenida de VisionPlatform.ai (2025)

4.8 Grounding DINO

Grounding DINO es un detector de objetos basado en técnicas avanzadas de aprendizaje profundo multimodal, capaz de procesar simultáneamente información visual y textual para realizar detecciones precisas en función de palabras clave o frases en lenguaje natural. Este enfoque permite que, al recibir una imagen junto con una consulta textual, el modelo identifique y localice únicamente los objetos que cumplen la descripción, superando las limitaciones de los detectores tradicionales que dependen de clases predefinidas.

Desde un punto de vista técnico, Grounding DINO se basa en la arquitectura DINO, una evolución de DETR que incorpora técnicas de entrenamiento de eliminación de ruido y un refinamiento iterativo de *anchor boxes*. Estas mejoras aceleran la convergencia del modelo y aumentan su precisión en la detección.

Para la parte textual, el modelo utiliza un Text Encoder basado en CLIP, que combina un codificador de texto tipo Transformer, similar a BERT, con un codificador visual como ViT o Swin Transformer. De esta forma, se generan representaciones ricas tanto del texto como de la imagen. La conexión entre ambas modalidades se realiza mediante un Cross-Modal Transformer, que aplica *cross-attention* para fusionar las características visuales y semánticas. Esto permite que las consultas de objetos del decoder estén directamente condicionadas por la información textual.

En cuanto al entrenamiento, el modelo sigue un enfoque mixto: primero se realiza un pre-entrenamiento multimodal sobre grandes colecciones de pares imagen–texto y, posteriormente, un fine-tuning supervisado en conjuntos de datos de detección. Este proceso le permite detectar conceptos no vistos previamente y adaptarse con facilidad a nuevos dominios con poca supervisión adicional.

El proceso de anotación automática comienza con la selección de un subconjunto reducido de imágenes, anotadas manualmente para servir como referencia de evaluación. Sobre este conjunto se mide el desempeño del modelo mediante métricas estándar como la Media de Precisión Promedio (MAP). Una vez alcanzado un nivel satisfactorio, se emplea el modelo para generar anotaciones automáticas en el resto del conjunto de datos. Este procedimiento produce etiquetas de alta calidad de manera eficiente, reduciendo significativamente el esfuerzo manual y acelerando el ciclo de desarrollo y entrenamiento de sistemas de visión por computador. Liu y cols. (2023).

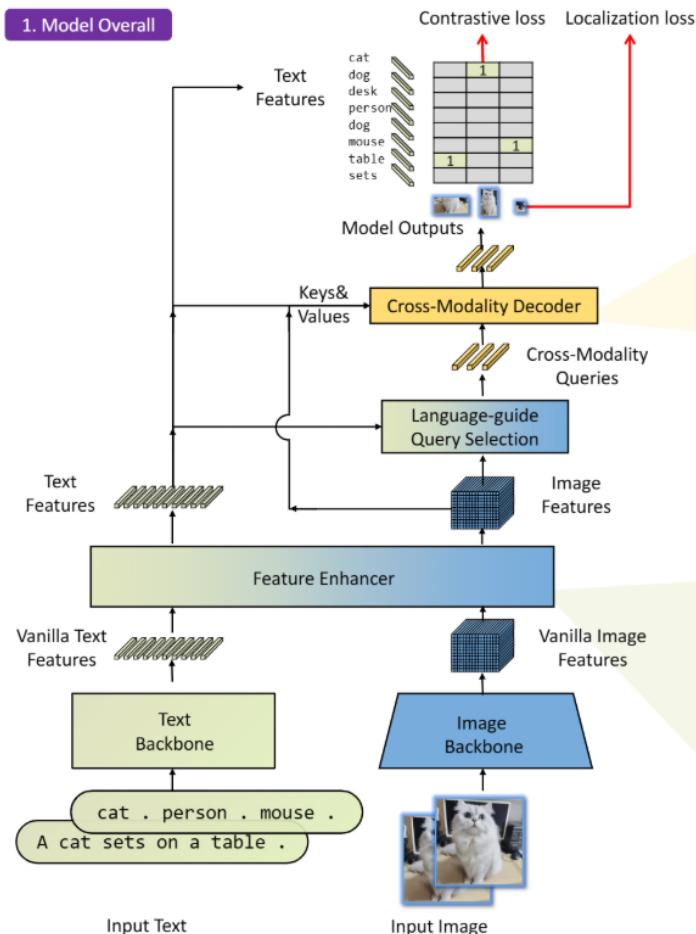


Figura 4.6: Funcionamiento General Grounding DINO
Obtenida de Liu y cols. (2023)

En este proyecto, Grounding DINO se ha utilizado para identificar cualquier objeto a partir de comandos dados por el usuario, permitiendo al robot llegar hasta el objeto deseado o en caso de ser un objeto móvil realizar el seguimiento del objeto.

El uso de Grounding DINO presenta varias ventajas relevantes para proyectos de visión por computador. En primer lugar, su capacidad de detección condicionada por consultas en lenguaje natural permite trabajar con conceptos abiertos, evitando la necesidad de clases predefinidas y facilitando la adaptación a nuevos dominios y escenarios no vistos durante el entrenamiento. Asimismo, ofrece capacidades avanzadas de comprensión de expresiones referenciales, lo que le permite localizar objetos concretos descritos mediante texto con gran precisión.

No obstante, también existen limitaciones a considerar. El modelo requiere un alto poder computacional para el entrenamiento y, en muchos casos, para la inferencia. En este trabajo se ha observado que, aunque su precisión es excelente, la velocidad de inferencia sigue siendo insuficiente para aplicaciones completamente en tiempo real.

En la Figura 4.7 se puede observar una comparación en caso de introducir *chair* como prompt, resultado a la izquierda de la imagen, o *dog's tail* como prompt, a la derecha de la imagen.

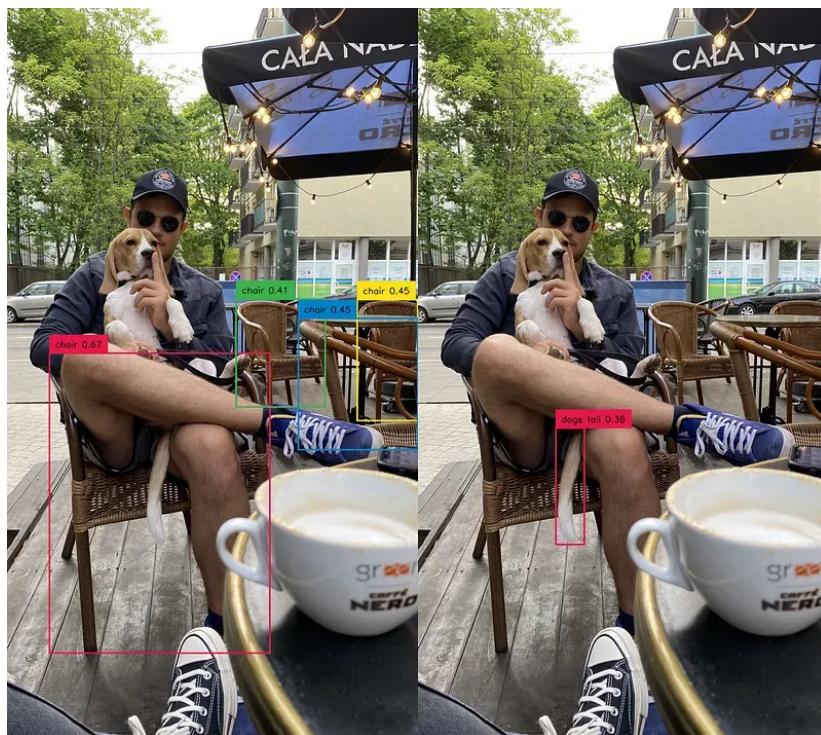


Figura 4.7: Ejemplo Grounding DINO
Obtenida de Skalski (2023)

4.9 PyTorch

PyTorch es un marco de deep learning de código abierto ampliamente adoptado en la comunidad científica. Basado en Python, PyTorch proporciona una interfaz intuitiva y flexible para construir, entrenar e implementar redes neuronales profundas, permitiendo un flujo de trabajo dinámico que resulta especialmente útil en entornos de investigación y prototipado rápido.

Uno de los componentes fundamentales de PyTorch es su sistema de diferenciación automática, que permite calcular gradientes de forma automática a través del seguimiento dinámico de las operaciones realizadas sobre los tensores. Esto facilita la optimización de los modelos mediante algoritmos de descenso del gradiente sin necesidad de derivar manualmente las expresiones matemáticas. Además, PyTorch integra una infraestructura eficiente para el manejo de datos, entrenamiento en GPU y despliegue en entornos de producción. IBM (2025).

Su papel ha sido esencial en este trabajo, ya que constituye la base sobre la que se han construido y ejecutado modelos como YOLOv8 y Grounding DINO, que aprovechan la arquitectura modular de PyTorch para integrar redes convolucionales, mecanismos de atención y modelos de detección de objetos de última generación.

4.10 SLAM

La localización y el mapeo simultáneos (Simultaneous Localization and Mapping (SLAM)) son problemas centrales en robótica móvil. Consisten en que un robot construya, mientras se desplaza, un mapa de su entorno y, a la vez, estime su propia posición dentro de ese mapa. Este doble objetivo elimina la dependencia de mapas previos o sistemas de posicionamiento externos como el GPS, siendo imprescindible en escenarios no estructurados o interiores donde dichos servicios no son fiables.

Para alcanzar este objetivo, los sistemas SLAM integran información de múltiples sensores, como escáneres LiDAR, cámaras, odometría y Unidad de Medición Inercial (IMU), junto con los registros de movimiento del robot. Estas medidas sensoriales se combinan mediante técnicas probabilísticas o de optimización para modelar la incertidumbre y minimizar el error acumulado derivado de la odometría y el ruido de los sensores.

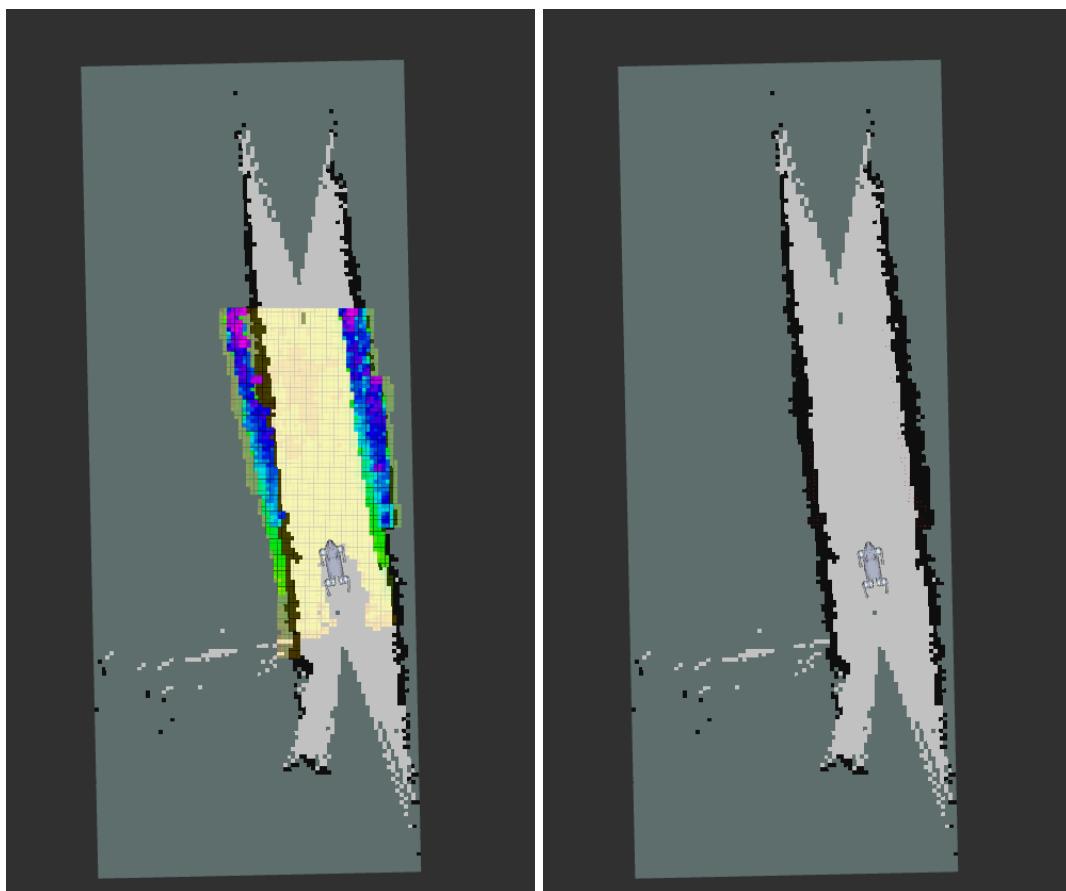
Una fase crítica es el cierre de bucle, que permite al sistema reconocer cuando el robot regresa a una zona previamente explorada. Esto posibilita corregir desplazamientos y errores acumulados en la trayectoria, obteniendo mapas globalmente más coherentes. Además, los enfoques SLAM se dividen en métodos en línea, que realizan mapeo y localización en tiempo real, y métodos fuera de línea, que procesan posteriormente datos registrados; aunque en la práctica, muchos sistemas combinan ambos modos según los recursos y objetivos.

ROS ofrece varios paquetes para implementar y probar algoritmos SLAM, como gmapping, cartographer, rtabmap y slam_toolbox. En nuestro caso, hemos utilizado slam_toolbox para la generación de mapas. Durrant-Whyte y Bailey (2006).

4.10.1 SLAM Toolbox

SLAM Toolbox es un paquete de ROS2 orientado a la implementación de SLAM en entornos bidimensionales mediante datos de sensores LiDAR. Su salida principal es un mapa de ocupación representado como una rejilla, donde cada celda se clasifica en tres estados:

- Celda libre: son las celdas representadas en color blanco ya que no se ha detectado ningún obstáculo en esa celda por el sensor.
- Celda ocupada: son las celdas representadas en color negro, esto nos indica la presencia confirmada de un obstáculo en esa celda.
- Celda desconocida: son las celdas que se muestran en color gris en los mapas. Estas celdas son regiones de las que no se tienen datos suficientes para determinar si están libres u ocupadas.



(a) Mapa y nube de puntos de obstáculos

(b) Mapa

Figura 4.8: Visualización del mapa en RViz

El núcleo de `slam_toolbox` se basa en la optimización de grafos de poses, donde cada nodo representa una estimación de la posición del robot y las aristas son restricciones derivadas

de la odometría y observaciones sensoriales. Cuando se detecta un cierre de bucle, el grafo se ajusta para corregir errores acumulados y mejorar la precisión del mapa.

El paquete ofrece varios modos de operación adaptados a distintas necesidades. En modo síncrono, el mapa se actualiza en tiempo real conforme el robot se desplaza; en modo asincrónico, las actualizaciones se realizan con menor frecuencia para reducir la carga computacional; el modo de localización permite utilizar un mapa preexistente para calcular la posición sin modificarlo; y el modo fuera de línea se encarga de procesar datos registrados previamente para construir el mapa sin exploración activa. Macenski y Jambrecic (2021).

Además, `slam_toolbox` se integra nativamente con herramientas como RViz para la visualización en tiempo real y es compatible con el sistema de navegación Nav2, facilitando el control autónomo del robot.

4.11 LiDAR

El sistema Light Detection And Ranging (LiDAR) es una tecnología de detección remota que permite medir distancias con gran exactitud mediante el uso de pulsos de luz láser. Este sistema es capaz de escanear superficies, detectar objetos y generar representaciones tridimensionales del entorno con un alto nivel de detalle. Su aplicación se ha extendido a ámbitos como la agricultura, la automoción, la robótica y, más recientemente, la logística y la intralogística, donde se emplea para optimizar procesos, mejorar la navegación autónoma y aumentar la seguridad operativa.

LiDAR forma parte de un conjunto de tecnologías de percepción espacial que incluye también sensores de ultrasonidos, cámaras con tecnología de navegación SLAM y sistemas basados en visión artificial. Entre todas ellas, LiDAR destaca por su alta resolución, elevada velocidad de captura y capacidad de operar en condiciones de baja iluminación o entornos adversos, lo que lo convierte en una herramienta versátil para aplicaciones tanto en interiores como en exteriores.

El principio de funcionamiento del LiDAR se basa en la emisión de pulsos de luz láser y la medición del tiempo que tardan en rebotar en los objetos y regresar al sensor. Aplicando la fórmula clásica de cálculo de distancias, que utilizando la velocidad de la luz y el tiempo de ida y vuelta del pulso, el sistema determina la posición relativa de cada punto detectado. El proceso se desarrolla en tres etapas principales:

1. Emisión del pulso láser, que en los sistemas actuales puede alcanzar millones de pulsos por segundo.
 2. Reflexión del pulso, cuando este impacta sobre superficies u objetos del entorno, como paredes, estanterías, vehículos o el suelo.
 3. Recepción y cálculo, donde el sensor registra el tiempo transcurrido y genera, a partir de estos datos, una nube de puntos bidimensional o tridimensional que representa el entorno con gran fidelidad.
-

En muchos casos, un mismo pulso puede producir múltiples retornos, lo que permite obtener información más detallada en entornos con vegetación, estructuras complejas o elementos parcialmente transparentes. Mecalux (2024).

4.11.1 Unitree LiDAR 4D L1

En el presente proyecto se emplea el modelo Unitree 4D LiDAR L1, un sensor compacto y ligero diseñado para entornos robóticos y de navegación autónoma. Este dispositivo ofrece un barrido omnidireccional de gran amplitud que permite capturar el entorno sin zonas ciegas, junto con una elevada densidad de muestreo que garantiza un modelado preciso tanto de elementos cercanos como lejanos. Su alcance permite trabajar eficazmente en espacios amplios, mientras que su capacidad para detectar objetos a distancias muy reducidas lo hace igualmente adecuado en maniobras de precisión. Además, incorpora una IMU de tres ejes, lo que facilita su integración con soluciones SLAM y mejora la precisión en tareas de localización y mapeo, incluso en entornos dinámicos. Estas características lo convierten en una herramienta idónea para el escaneo continuo del entorno, la detección fiable de obstáculos y la generación de mapas en tiempo real. Lidar (2025).



Figura 4.9: Unitree LiDAR 4D L1
Obtenida de Lidar (2025)

4.12 RViz

RViz es una herramienta de visualización 3D diseñada para el ecosistema ROS. Su función principal es proporcionar una representación gráfica en tiempo real de los datos sensoriales

y del estado del robot, facilitando la comprensión del entorno y el comportamiento del sistema durante la operación. RViz permite visualizar mapas, nubes de puntos, trayectorias, posiciones y orientaciones, así como objetos detectados, rutas planificadas y otros elementos relevantes para la navegación y percepción robótica. Documentation (2018).

Esta herramienta es altamente configurable y extensible, funcionando mediante la suscripción a tópicos de ROS que transmiten información sobre sensores, estados y resultados computacionales. Los usuarios pueden añadir múltiples tipos de visualizaciones que incluyen, entre otros, mapas 2D y 3D, modelos de robot, marcadores, nubes de puntos, cámaras y textos descriptivos. Esta flexibilidad permite adaptar RViz a distintas necesidades y tipos de robots, consolidándose como una herramienta fundamental para el desarrollo, depuración y pruebas en robótica.

En este proyecto, RViz se ha empleado para visualizar el estado general del robot articulado Unitree Go2, incluyendo su posición estimada y orientación actual en el espacio. Además, se ha utilizado para representar el mapa generado por el sistema SLAM, la nube de puntos con los objetos detectados en el entorno, y las imágenes capturadas por la cámara y el LiDAR del robot. Estas visualizaciones permiten supervisar de forma eficiente el entorno y el desempeño del robot durante la ejecución de las tareas.

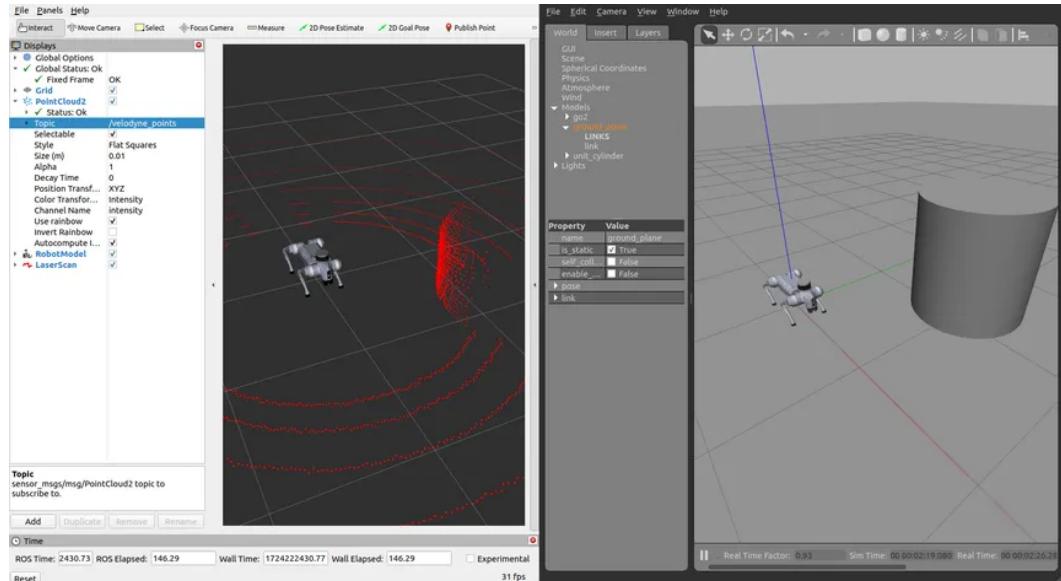


Figura 4.10: Visualización RViz

4.13 Cámara integrada

La cámara frontal que viene integrada en el robot articulado Unitree Go2 ofrece una transmisión de imagen con una resolución de 1280×720 píxeles y un campo de visión (FOV) de

120°. Está equipada con una lente gran angular que permite capturar una amplia parte del entorno, facilitando así la percepción visual en situaciones complejas.

Desde la perspectiva de integración, la cámara puede ser publicada en el ecosistema ROS2 a través de tópicos de imagen, lo que la hace útil en procesos de percepción y visualización. Su compatibilidad con bibliotecas como OpenCV o NumPy simplifica el procesamiento de datos visuales en aplicaciones que requieren análisis de escenas en tiempo real.

Una característica importante es cómo complementa a otros sensores, como el LiDAR, ya que proporciona información adicional en entornos complicados y mejora la representación del entorno en condiciones específicas. Además, la cámara se integra de manera nativa con herramientas de visualización como RViz, lo que facilita la supervisión gráfica de los datos recogidos durante la operación del robot.

En este proyecto, la cámara integrada del Go2 se ha utilizado como fuente de datos visuales dentro del flujo de percepción sensorial del sistema, contribuyendo a la representación y análisis del entorno junto con el resto de los sensores.

4.14 Repositorio go2_ros2_sdk

El *go2_ros2_sdk* es un repositorio de código abierto desarrollado por Abizov (2025), que proporciona un Software Development Kit (SDK) no oficial para integrar el robot articulado Unitree Go2 en el ecosistema ROS2.

Este repositorio incluye nodos y utilidades que permiten la comunicación entre el robot y ROS2, ofreciendo soporte para la transmisión de datos de sensores como la cámara integrada, el LiDAR y la IMU. Asimismo, incorpora herramientas para la visualización y navegación autónoma, como slam_toolbox, Nav2 y foxglove_bridge, facilitando de esta forma la representación del entorno y el control de la movilidad del robot.

En este proyecto, el repositorio *go2_ros2_sdk* ha sido empleado como base para la integración del robot con ROS2, habilitando la comunicación entre sus sensores y actuadores, y permitiendo la ejecución de tareas de percepción, mapeo y navegación. Gracias a esta herramienta, se logró reducir el tiempo de desarrollo, ya que proporciona configuraciones y paquetes previamente implementados que pueden adaptarse a las necesidades específicas del sistema.

4.15 Tkinter

Tkinter es la biblioteca estándar de Python para el desarrollo de cualquier Interfaz Gráfica de Usuario (GUI). Forma parte de la instalación por defecto de Python, lo que permite su uso sin necesidad de instalar dependencias adicionales. Está basada en el kit de herramientas Tk, un conjunto de herramientas multiplataforma que ofrece componentes visuales como botones, etiquetas, menús, campos de texto, paneles o cuadros de diálogo. Estos elementos pueden organizarse y configurarse de forma sencilla, lo que hace que Tkinter sea una opción

muy utilizada para prototipado y desarrollo rápido de aplicaciones de escritorio.

Una de las principales ventajas de Tkinter es que abstrae la complejidad de la programación gráfica de bajo nivel, permitiendo al desarrollador centrarse en la lógica de la aplicación. Además, su compatibilidad con Windows, macOS y Linux garantiza que el mismo código pueda ejecutarse sin cambios en distintos entornos. Además, está diseñado y preparado también para lenguajes dinámicos con un alto nivel, como pueden ser Tcl, Ruby o Perl, entre otros. Tkinter (2025).

En este proyecto, se ha empleado Tkinter para facilitar la interacción entre el operador y el sistema de control del robot articulado Unitree Go2. La interfaz implementada con Tkinter presenta una serie de botones que permiten al usuario controlar las acciones que el robot llevará a cabo, desde el movimiento del propio robot hasta el guardado de mapas, la detección de objetos y su localización en el mapa, hasta las acciones de seguimiento de los objetos detectados. Además, Tkinter permite abrir cuadros de diálogo en los que el usuario puede introducir un prompt textual. Este prompt se utiliza como entrada para cualquiera de las dos redes neuronales integradas, YOLOv8 y Grounding DINO. De este modo, el usuario puede definir de forma dinámica el objeto o concepto que desea buscar en el entorno, y el sistema adapta en tiempo real sus procesos de detección y localización en función de esa instrucción.

4.16 Numpy

NumPy es una biblioteca fundamental de Python para la computación científica, que proporciona soporte para arrays multidimensionales de alto rendimiento y herramientas para realizar operaciones matemáticas avanzadas sobre estos arrays. Su estructura principal, el objeto ndarray, permite almacenar y manipular grandes volúmenes de datos numéricos de manera eficiente, lo que la convierte en un componente esencial en proyectos de robótica y visión por computadora. Oliphant (2025).

En este proyecto, NumPy se ha utilizado para procesar y gestionar los datos sensoriales adquiridos por el robot articulado Unitree Go2. Gracias a su capacidad para manejar arrays multidimensionales, se ha facilitado la integración y manipulación de datos provenientes de diversas fuentes, como la cámara o el LiDAR. Además, NumPy ha permitido realizar operaciones matemáticas y estadísticas sobre estos datos, como transformaciones geométricas, esenciales para la interpretación y comprensión del entorno del robot.

La eficiencia y versatilidad de NumPy han sido clave para el procesamiento en tiempo real de los datos sensoriales, permitiendo una rápida adaptación y respuesta del sistema ante cambios en el entorno. Su integración con otras bibliotecas como OpenCV y Matplotlib ha facilitado la visualización y análisis de los datos, contribuyendo al desarrollo y evaluación del sistema de navegación y percepción del robot.

4.17 TF2

TF2 es un paquete del ecosistema de ROS diseñado para gestionar y consultar transformaciones entre distintos sistemas de referencia en robótica. Su objetivo principal es permitir que un robot pueda conocer en todo momento la posición y orientación relativa de sus diferentes componentes, así como la de los objetos y elementos del entorno que se han mapeado o detectado. Esto se logra mediante la publicación continua de transformaciones o frames en una estructura jerárquica llamada árbol de transformaciones, donde cada nodo representa un sistema de coordenadas y cada enlace define una transformación rígida en el espacio 3D.

A diferencia de su versión anterior, TF, la versión TF2 mejora la eficiencia y la consistencia de las transformaciones al separar la funcionalidad de almacenamiento y consulta de datos de las interfaces de comunicación. Esta arquitectura optimizada reduce la latencia y evita errores debidos a datos obsoletos, garantizando así que las transformaciones se calculen con la información más actualizada. Garage (2025).

En este proyecto, TF2 se ha empleado para mantener actualizado el árbol de transformaciones del robot articulado Unitree Go2. Esto incluye la relación entre la base del robot, sus sensores como el LiDAR o la cámara y las entidades detectadas en el entorno. Gracias a TF2, el sistema de SLAM puede ubicar correctamente los datos sensoriales en un marco de referencia común, lo que permite, por ejemplo, proyectar la nube de puntos del LiDAR sobre el mapa global o situar en él las detecciones realizadas por las redes neuronales YOLOv8 y Grounding DINO. Esta capacidad es fundamental para que la localización, el mapeado y la detección de objetos trabajen de forma coherente y coordinada.

5 Desarrollo

En esta sección, vamos a explorar los procesos prácticos que hemos desarrollado en nuestro proyecto para implementar en el robot Unitree Go2. Estos procesos incluyen tareas de navegación, percepción, mapeo y seguimiento. En la Figura 5.1, se puede ver un esquema visual que resume las diferentes etapas que hemos llevado a cabo para alcanzar los objetivos que mencionamos anteriormente. A continuación, se proporcionará un breve adelanto de los temas que vamos a tratar en detalle en esta sección.

Primero, hablaremos sobre la configuración inicial del robot, que nos permite establecer la conexión a través de WebRTC. Luego, nos enfocaremos en la navegación autónoma, utilizando los datos que nos proporciona el LiDAR. Más adelante, explicaremos el procedimiento de mapeo con la herramienta SLAM Toolbox y, una vez que hayamos completado este paso, detallaremos cómo se lleva a cabo la detección de objetos utilizando redes neuronales profundas como YOLOv8 y Grounding Dino. Por último, mostraremos cómo hemos implementado el seguimiento de objetos, apoyándonos nuevamente en las arquitecturas de redes neuronales que mencionamos antes.

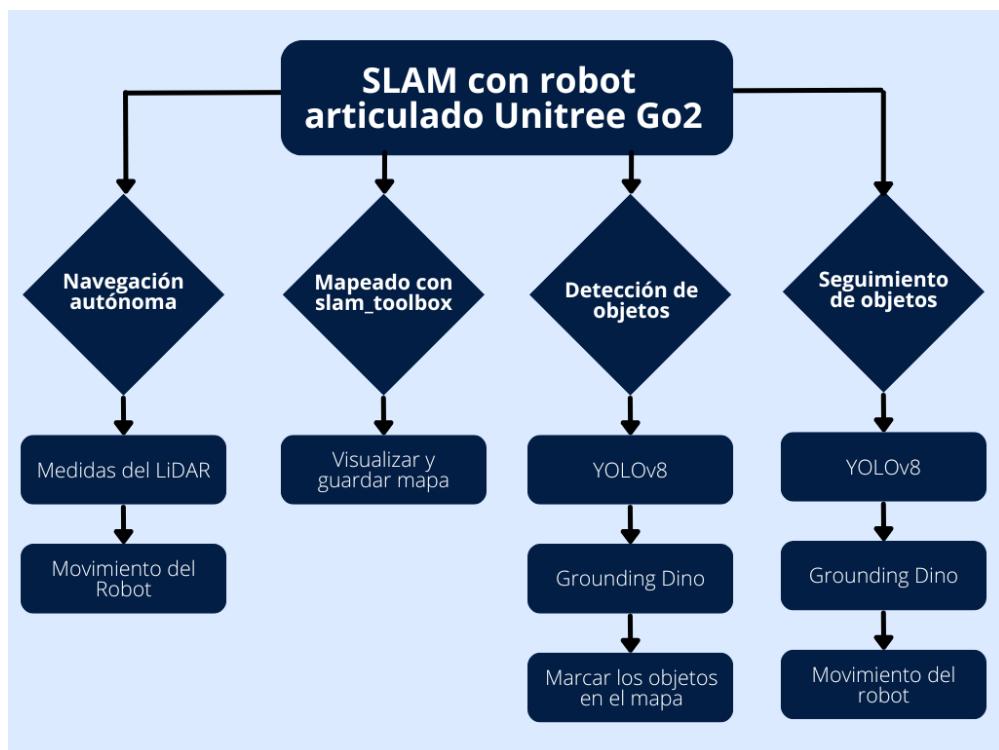


Figura 5.1: Esquema visual del proyecto

5.1 Conexión con el robot

El proceso de configuración del robot comienza con su encendido. Para ello, es necesario mantener presionado el botón de arranque durante unos segundos. El arranque se considera exitoso cuando los indicadores de la batería se iluminan en color verde, el LiDAR inicia su rotación y el robot se incorpora, adoptando la posición erguida. A continuación, se procede a configurar la red Wi-Fi del robot con el fin de establecer la conexión.

Este ajuste se realiza desde la aplicación móvil oficial de Unitree, seleccionando el modo Access Point (AP), en el cual el propio robot genera una red inalámbrica a la que el ordenador puede conectarse directamente. Este método representa la forma más sencilla de emplear WebRTC sin necesidad de infraestructura adicional. Además, la conexión mediante WebRTC permite aprovechar funcionalidades como la transmisión de vídeo, acceso a datos del LiDAR o envío de comandos básicos mediante tópicos de ROS. Por último, para poder usar el robot deberemos cambiar el modo del robot al modo general desde dentro de la aplicación.

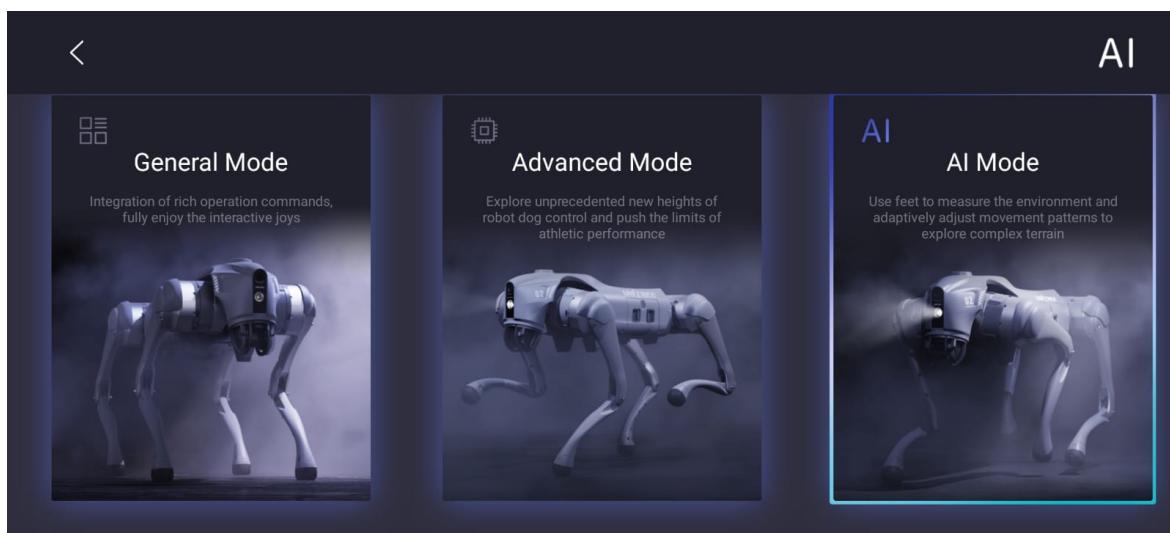


Figura 5.2: Selección de modo en la aplicación Unitree

Una vez configurada la red, el siguiente paso consiste en clonar el repositorio *go2_ros2_sdk*, siguiendo las indicaciones descritas en Abizov (2025). Posteriormente, se deben definir las variables de entorno que especifican tanto la dirección IP del robot como el tipo de conexión a utilizar. Para ello, se añaden al final del archivo *.bashrc* las líneas correspondientes, en las que, en nuestro caso, se establece la dirección IP *192.168.12.1* y el modo de conexión *webrtc*, tal como se indica en la documentación del repositorio.

Con esta configuración completada, solo resta compilar el paquete y ejecutar el archivo *robot.launch.py*. Si el procedimiento se ha realizado correctamente, se iniciará RViz mostrando la visualización representada en la Figura 5.3.

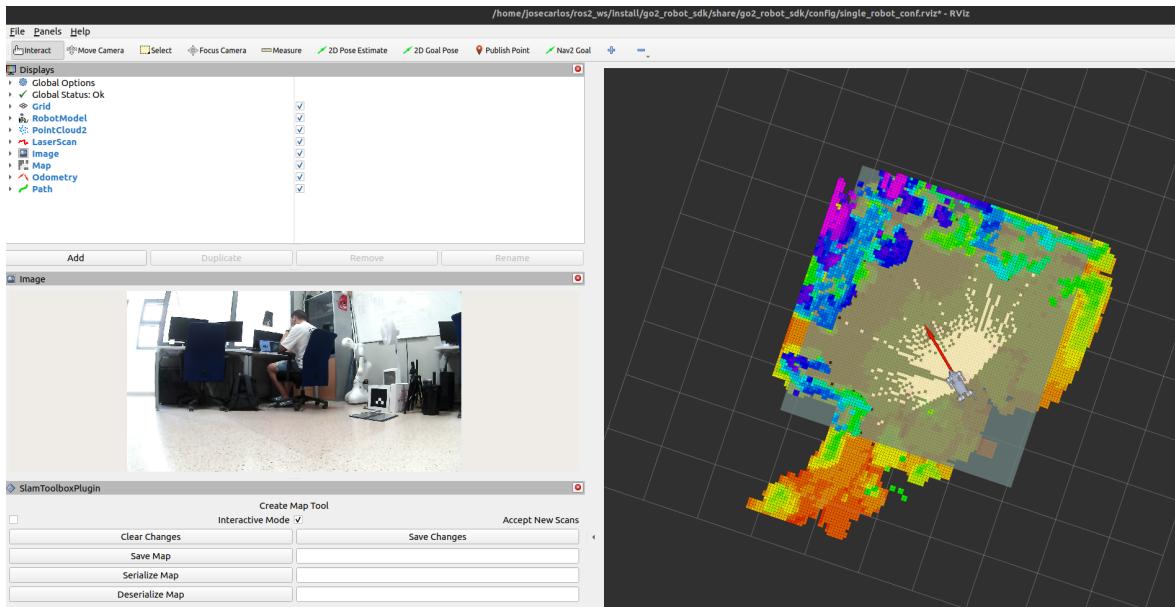


Figura 5.3: Visualización launch en RViz

5.2 Navegación autónoma

Volviendo al esquema mencionado al inicio de esta sección y mostrado en la Figura 5.1, el primer bloque que encontramos corresponde a la navegación autónoma. Este bloque se apoya en el sensor LiDAR que incorpora el robot, a través del cual se leen continuamente valores de distancia publicados en el tópico `/scan`. A partir de dicha información, el sistema genera órdenes de movimiento que se envían como mensajes de velocidad en el tópico `/cmd_vel`.

El funcionamiento comienza con la creación de un nodo de ROS2, encargado de gestionar la comunicación con el resto del sistema. Este nodo recibe de manera periódica las lecturas del sensor láser y, en paralelo, mantiene abierto un canal de salida que permite transmitir los comandos de velocidad al robot. Con el fin de asegurar un comportamiento en tiempo real, se establece un temporizador que activa la lógica de decisión cada décima de segundo, de modo que el robot actualiza su movimiento diez veces por segundo.

Al inicio, mientras no se reciben datos válidos del LiDAR, el robot permanece detenido, de forma que solo comienza a desplazarse una vez que la información sensorial está disponible. Cuando llega una nueva lectura, la lista de distancias se transforma en un formato numérico adecuado y se limpia de valores extraños. Si aparecen infinitos o valores no numéricos, se sustituyen por una distancia máxima de 20 metros, y además, todas las lecturas se limitan a un intervalo entre 0 y 20 metros para evitar que un error puntual pueda condicionar la decisión del sistema.

A partir de esos datos se calcula el ángulo de cada haz, lo que permite dividir el campo de visión en tres sectores: uno frontal, estrecho, centrado en el eje del robot, y dos laterales, izquierdo y derecho, que completan el resto del barrido. En la implementación se ha fijado

un sector frontal de aproximadamente 30° , mientras que cada lateral ocupa unos 165° . De este modo, todas las lecturas comprendidas entre -15° y $+15^\circ$ se consideran parte de la zona frontal, aquellas situadas entre -180° y -15° pertenecen al lado izquierdo, y las que van de $+15^\circ$ a $+180^\circ$ corresponden al lado derecho. La Figura 5.4 ilustra gráficamente esta división, quedando la sección 1 para el lado izquierdo, el sector 2 para la zona frontal y, por último, la sección 3 corresponde al lado derecho.

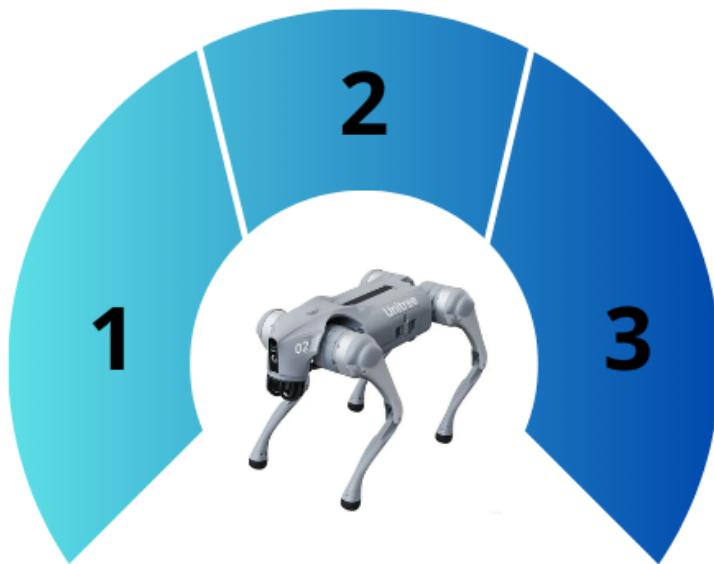


Figura 5.4: Rangos LiDAR

Para cada sector no se toma un único valor mínimo ni una media aritmética, ya que ambos pueden resultar poco fiables: el mínimo es demasiado sensible a lecturas aisladas y la media puede verse distorsionada por valores extremos. En su lugar se calcula el percentil 20, que equivale a escoger un valor de entre los más bajos, pero evitando quedarse siempre con la distancia mínima. De esta forma se obtiene una medida prudente y estable, que asume la presencia de obstáculos cercanos sin reaccionar de manera exagerada a un único dato erróneo.

Con estas distancias sectoriales actualizadas, el nodo conserva internamente el valor de cada lado y procede a la toma de decisiones. La primera condición que se comprueba es si el robot se encuentra en modo de marcha atrás. En ese caso, continúa retrocediendo a una velocidad fija durante un tiempo máximo de unos dos segundos. Una vez transcurrido ese intervalo, se detiene y selecciona al azar una dirección de giro, con el objetivo de reorientarse y salir de un posible callejón sin salida.

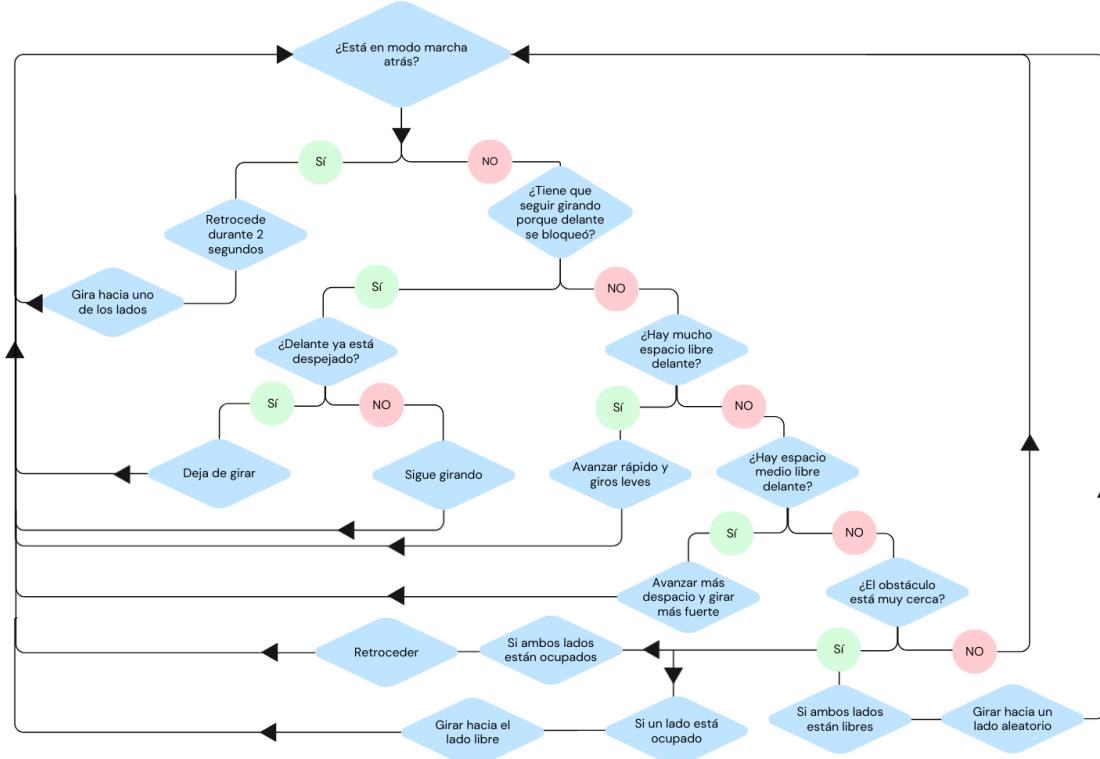


Figura 5.5: Diagrama de flujo navegación autónoma

Si no está retrocediendo, puede encontrarse en un giro forzado, es decir, en una situación en la que el frente está bloqueado, pero uno de los laterales ofrece espacio libre. En ese estado, el robot mantiene la rotación en un único sentido hasta que la zona frontal se despeja un poco, momento en el cual recupera el modo de avance normal.

Cuando el sector frontal indica que existe suficiente espacio libre, más de metro y medio, el robot avanza a su velocidad de crucero. Mientras lo hace, sigue controlando los sectores laterales para corregir pequeñas desviaciones: si detecta que se aproxima demasiado a un obstáculo por la izquierda, corrige ligeramente hacia la derecha, y si ocurre lo contrario, corrige hacia la izquierda. En ausencia de paredes cercanas, mantiene una ligera deriva, lo que también contribuye a corregir una pequeña desviación que tiene el robot al intentar avanzar en línea recta.

Si el obstáculo se encuentra a una distancia intermedia, no crítica pero tampoco amplia, el robot reduce su velocidad y aplica giros más pronunciados hacia el lado que se encuentra más despejado. Este comportamiento le permite ganar margen de seguridad y reaccionar antes de que la situación se vuelva peligrosa.

Cuando finalmente un obstáculo se aproxima demasiado, el avance se detiene por completo. En ese momento se decide la maniobra a seguir: si tanto la izquierda como la derecha se encuentran bloqueadas, se activa el modo marcha atrás para poder liberarse. Si solo uno

de los lados está ocupado, se selecciona el giro hacia el lado contrario y se mantiene hasta despejar el frente, y si ambos lados están despejados, se elige de forma aleatoria una dirección de giro.

La decisión tomada en cada ciclo se traduce en una orden de movimiento que se envía al robot publicando un mensaje de velocidad en el tópico `/cmd_vel`. Este proceso se repite constantemente, de forma que el robot se adapta de manera continua a lo que percibe en su entorno. Finalmente, cuando el programa se detiene, se publica una orden con velocidad nula para garantizar que el robot quede completamente parado.

5.3 Mapeado

Tras la revisión teórica de las técnicas SLAM y de las funcionalidades del paquete SLAM Toolbox en la sección 4, en este apartado se describe su aplicación práctica en el robot Unitree Go2.

De los métodos posibles para realizar el mapeado, el sistema se configuró en modo síncrono, de forma que el mapa se generaba en tiempo real mientras el robot recorría el entorno. Para ello, el nodo de `slam_toolbox` recibía como entrada las lecturas del sensor LiDAR publicadas en el tópico `/scan`, así como la odometría interna del robot. A partir de esa información, el algoritmo estima la trayectoria seguida mientras actualiza de manera incremental un mapa de ocupación 2D.

Durante la exploración, la visualización en tiempo real del mapa se realizó mediante RViz, lo que permitió comprobar de forma inmediata la coherencia del trazado y la detección de paredes y obstáculos. En este proceso, el robot ejecutaba simultáneamente la lógica de navegación autónoma descrita en el apartado anterior, de modo que avanzaba de manera reactiva mientras generaba el mapa del entorno. El operador podía observar en RViz cómo el mapa se iba completando hasta decidir finalizar la exploración. Para detener la construcción del mapa es necesario parar previamente la ejecución de la navegación autónoma y, a continuación, proceder al guardado del resultado.

El mapa generado se almacena en los formatos estándar de ROS2: un archivo `.pgm` con la representación gráfica y un archivo `.yaml` con los metadatos como origen, resolución y escala. Este guardado puede realizarse mediante el comando en terminal:

```
ros2 run nav2_map_server map_saver_cli
```

O bien utilizando el menú de guardado en RViz, donde indicando el nombre deseado y pulsando el botón Save Map se guardará el mapa, o bien utilizando el botón añadido en la interfaz desarrollada, que automatiza este proceso para mayor comodidad.

Estos mapas pueden reutilizarse posteriormente, lo que evita reconstruir el entorno en cada sesión y permite que el robot se posicione directamente en un entorno previamente explorado. Esta funcionalidad resulta especialmente útil para la integración con el sistema de navegación autónoma, ya que combina la evitación reactiva de obstáculos con la referencia global proporcionada por el mapa.

La Figura 5.6 muestra un ejemplo del resultado obtenido en un entorno de pruebas, donde se aprecian claramente las áreas ocupadas por paredes y mobiliario, las regiones libres por las que el robot puede desplazarse y las zonas desconocidas no exploradas.

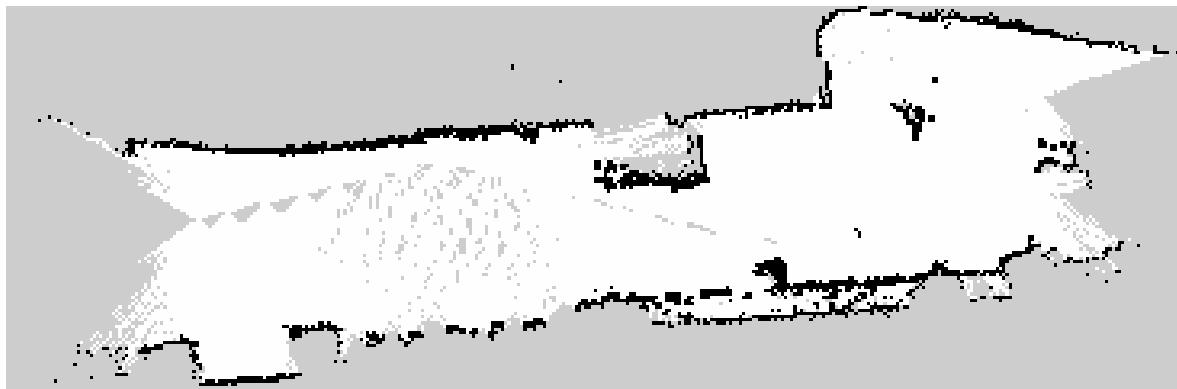


Figura 5.6: Mapa creado con `slam_toolbox`

5.4 Detección de objetos

Durante esta sección pasaremos al tercer bloque que se encuentra en el esquema visual de la Figura 5.1. En este punto debemos mencionar que se hará uso del sensor LiDAR que posee el robot a través de los valores leídos en el tópico `/scan`. También se utilizará el mapa que va creando el robot y de la cámara que posee el robot a través de los valores leídos en los tópicos `/map` y `/camera/image_raw`, respectivamente.

En términos generales, durante este bloque se busca aplicar dos redes neuronales profundas, como son YOLOv8 y Grounding Dino, a la imagen captada por la cámara integrada del robot. Durante esta sección se mostrará cómo se ha conseguido cumplir el objetivo de detectar los objetos deseados para posteriormente marcarlos en el mapa que crea el robot, almacenando también la posición de los objetos encontrados.

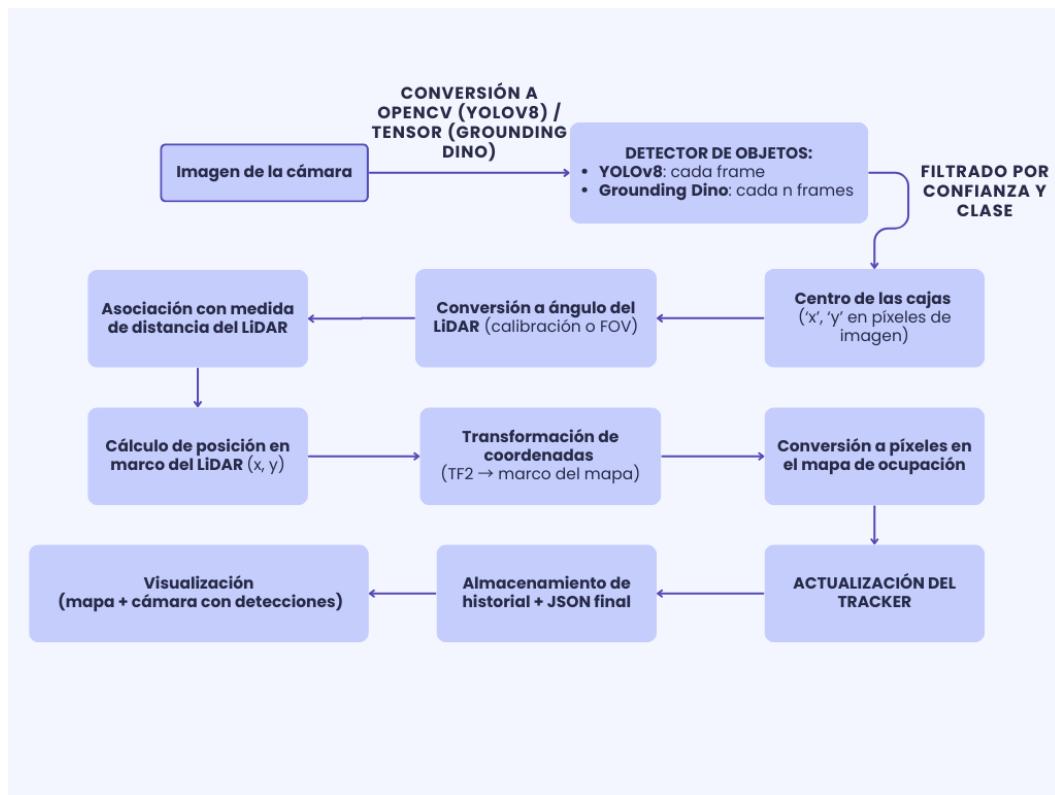


Figura 5.7: Pipeline códigos detección de objetos

5.4.1 YOLOv8

El proceso se inicia con la creación de un nodo que establece el puente para convertir imágenes, reserva los valores necesarios para el mapa, inicializa un mecanismo de seguimiento de objetos, carga el modelo YOLOv8 y configura los marcos de referencia a utilizar, que en este caso son la cámara, el LiDAR y el mapa. Además, se generan variables de sesión destinadas a almacenar los resultados obtenidos.

Antes de comenzar a procesar datos, es necesario sincronizar las lecturas de la cámara con las del escáner láser mediante un sincronizador temporal. De este modo se garantiza que la imagen y el barrido del láser corresponden al mismo instante. Tras este paso, se abre una ventana sencilla, implementada con Tkinter, en la que el usuario debe indicar la clase que desea detectar. El sistema permite escribir una clase concreta o bien seleccionar la opción "All", lo que hace que se procesen todas las detecciones encontradas.

Una vez indicada la clase, el nodo espera a recibir la calibración de la cámara, a partir de la cual se calcula el ángulo de visión utilizando la matriz intrínseca. Este cálculo es esencial para relacionar los píxeles de la imagen con direcciones en el espacio real. Posteriormente se recibe el mapa que va construyendo el robot, transformándolo en una imagen para poder mostrarlo en tiempo real.

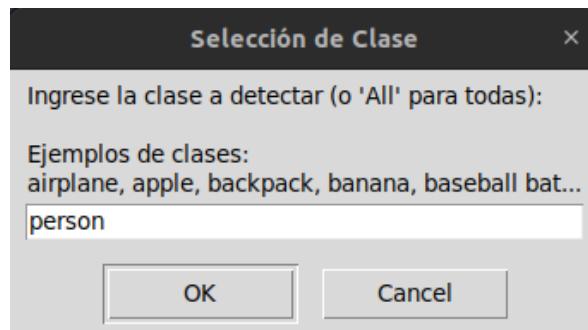


Figura 5.8: Introducción prompt para YOLOv8

Mientras no exista una clase seleccionada o el mapa no esté disponible, el nodo permanece inactivo y no procesa mensajes. Una vez todo está preparado, la imagen recibida se convierte a formato OpenCV y, junto con los límites angulares del láser, obtenidos de la información de su tópico, se pasa al detector YOLOv8 para obtener las predicciones. En paralelo, se crea una copia del mapa en la que se irán representando los objetos detectados.

Las detecciones devueltas por YOLO se recorren una a una. Para cada una de ellas se dibuja un recuadro con la etiqueta correspondiente en la imagen de la cámara, descartando aquellas cuyo nivel de confianza es inferior a un umbral definido, lo que evita falsas detecciones. Si el usuario ha seleccionado una clase concreta, solo se tendrán en cuenta las coincidencias con esa clase; en caso contrario, se procesarán todas las cajas detectadas.

Para cada detección válida se toma el centro de la caja, que se asume como aproximación al centro del objeto, y se convierte en una dirección en la imagen y en un ángulo equivalente dentro del rango del LiDAR. Este ángulo puede calcularse de dos formas: si se dispone de calibración, se utiliza la matriz intrínseca de la cámara; si no, se recurre al campo de visión aproximado. Con el ángulo calculado se localiza el índice correspondiente en el vector de rangos del láser.

En lugar de tomar directamente la medida en ese índice, se aplica un filtro de robustez. Lo que se hace es comprobar si la lectura en ese punto es válida; si no lo es, se examina una pequeña ventana de valores vecinos y se escoge el más cercano que resulte correcto. De este modo se evita depender de una sola lectura que podría estar vacía o corrupta. Si no se encuentra ninguna medida válida, la detección se descarta.

Con un ángulo y una distancia válidos, se calcula la posición del objeto en el marco del LiDAR mediante las fórmulas:

$$x = r \cdot \cos(\theta) \quad (5.1)$$

$$y = r \cdot \sin(\theta) \quad (5.2)$$

donde r es la distancia medida y θ el ángulo calculado.

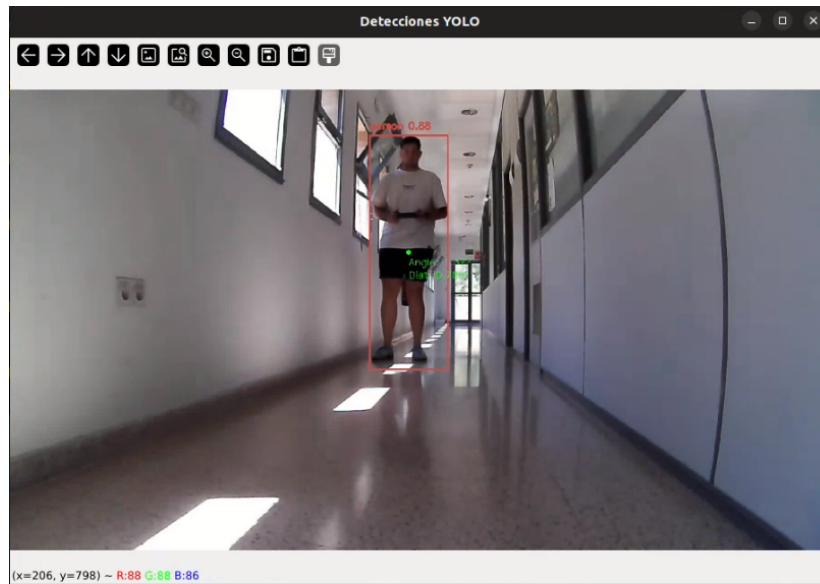


Figura 5.9: Detección con YOLOv8

Esa posición todavía está en el sistema de referencia del LiDAR, por lo que a continuación se utiliza la transformación de coordenadas, TF2, para pasar del marco del sensor al marco global del mapa. Una vez en coordenadas del mapa, se convierten a píxeles teniendo en cuenta la resolución y el origen del mismo. De esta forma la detección queda ubicada sobre el mapa y puede representarse visualmente.

El siguiente paso consiste en actualizar el *tracker* de objetos. Para ello se comprueba la cercanía de cada detección nueva respecto a las ya existentes: si un objeto aparece dentro de un umbral de distancia de otro previamente registrado y tiene la misma etiqueta, se considera que es el mismo objeto y se actualiza suavizando la posición. Si no coincide con ninguno, se crea un objeto nuevo. Además, los objetos que llevan un tiempo sin detectarse se marcan como inactivos, lo que se refleja en el mapa con un cambio de color.

Finalmente, el nodo dibuja la posición del robot en el mapa, también usando TF2, y muestra las ventanas con las imágenes del mapa y de la cámara en las que se incluyen las detecciones. Al concluir la sesión se guardan los datos relevantes, como las estadísticas de la sesión, es decir, número total de objetos detectados, cuántos de estos objetos están activos y sus respectivas posiciones en las que se han detectado y la duración total de la sesión en segundos, todo esto en un archivo JSON. También se guardará el mapa del entorno en formato *.pgm* y *.yaml* usando la herramienta *map_saver_cli* de ROS2 mencionada justo en el apartado anterior. Por último, también se guardará una imagen del mapa final en formato PNG que incluirá los objetos detectados dibujados junto a la posición final del robot.

5.4.2 Grounding DINO

En el caso de Grounding DINO, el nodo se inicia de manera muy similar al de YOLOv8: establece el puente para convertir imágenes, reserva memoria para el mapa, inicializa el mecanismo de seguimiento de objetos, carga el modelo Grounding Dino y configura los marcos de referencia a utilizar, que en este caso son la cámara, el LiDAR y el mapa. Asimismo, define la visualización mediante Matplotlib con dos paneles, uno para el mapa y otro para la cámara.

Al inicio, se abre una ventana con Tkinter para solicitar al usuario la clase a detectar, que puede introducirse en texto libre. Si no se introduce nada, se utiliza por defecto la clase *person*. También se inicializan variables de sesión para almacenar la información obtenida durante la ejecución.

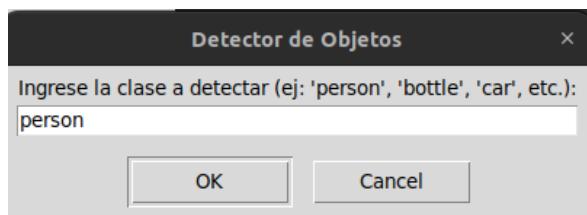


Figura 5.10: Introducción prompt para Grounding DINO

Al igual que en YOLOv8, el nodo se suscribe a los tópicos de calibración de la cámara, al mapa de ocupación y sincroniza las lecturas de la cámara y el escáner láser. Cuando se recibe la calibración de la cámara, se calcula el campo de visión a partir de la matriz intrínseca, y al recibir el mapa, este se convierte a imagen y se marca como disponible para su visualización y posterior procesado.

Una diferencia clave con respecto a YOLOv8 es que Grounding DINO es una red neuronal mucho más pesada, lo que dificulta su ejecución en tiempo real en todos los fotogramas. Para solventar esto, la detección se lanza de forma periódica, ya sea cada cierto número de imágenes procesadas o bien si ha transcurrido un intervalo de tiempo mínimo desde la última ejecución. Cuando el nodo decide realizar una detección, la imagen de entrada se reduce a 320x240 píxeles, se convierte a formato RGB y se transforma en un tensor para que el modelo pueda procesarla de forma más rápida. A continuación, se lanza la predicción, indicando el texto de la clase objetivo y aplicando los umbrales de confianza configurados.

El modelo devuelve como salida un conjunto de cajas normalizadas con respecto a la imagen reducida. El nodo es el encargado de convertir esas cajas de vuelta a coordenadas de píxeles sobre el fotograma original. El proceso sigue tres pasos: primero se calculan las coordenadas de la caja en la imagen reducida, después se reescalan al tamaño real del frame de la cámara y, finalmente, se filtran las detecciones con una confianza inferior al umbral definido, descartando falsas detecciones. Para cada detección válida se toma el centro de la caja como aproximación al centro del objeto.

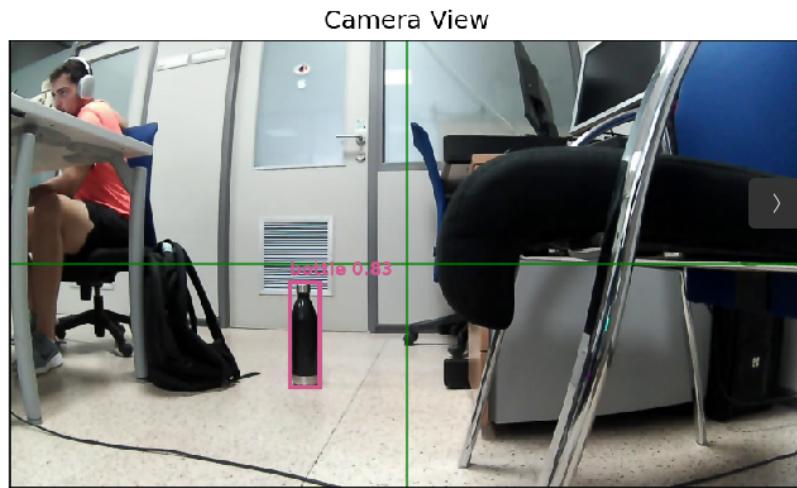


Figura 5.11: Detección con Grounding DINO

A partir de ese punto central, el sistema calcula el ángulo en el LiDAR utilizando los parámetros intrínsecos de la cámara. Se comprueba que dicho ángulo se encuentra dentro del rango de medidas del escáner láser y se calcula el índice correspondiente en el vector de rangos. Para garantizar la robustez, no se toma directamente la lectura en ese índice, sino que se busca en una pequeña ventana de valores vecinos y se selecciona la medida más cercana que resulte válida. Si no se encuentra ninguna medida confiable, la detección se descarta.

Con un ángulo y una distancia válidos se obtiene la posición del objeto en el marco del LiDAR. Posteriormente, mediante TF2, se transforma dicha posición al marco global del mapa, lo que permite calcular sus coordenadas en píxeles dentro del mapa de ocupación. En este momento, la detección se almacena tanto en el historial como en el *tracker*. Para cada detección se registra un conjunto de datos que incluye: la etiqueta en texto, el valor de confianza, la caja delimitadora en coordenadas de píxeles del fotograma original, la posición del objeto en el mundo, las coordenadas en el mapa, la distancia medida y el ángulo correspondiente. Esta información constituye la base tanto para el análisis posterior como para la representación gráfica.

El *tracker* de objetos se actualiza utilizando la misma lógica que en el caso de YOLOv8, compara las nuevas detecciones con los objetos ya registrados, emparejándolos en función de su proximidad y de la etiqueta detectada. Cuando se considera que corresponde al mismo objeto ya que se ha encontrado en una posición muy cercana a un objeto ya detectado, se actualiza suavizando la posición y se incrementa el número de detecciones acumuladas; en caso contrario, se crea un nuevo registro. Los objetos que llevan un tiempo prolongado sin ser detectados pasan a estar inactivos, lo cual se refleja en la visualización del mapa.

En cuanto a la visualización, Grounding DINO utiliza Matplotlib en lugar de OpenCV. Esto permite representar de manera simultánea la imagen de la cámara y el mapa en una única ventana con dos paneles. En el panel de la cámara se muestran las cajas detectadas,

las etiquetas, la confianza y una cruz en el centro de la imagen. En el panel del mapa se representa la posición del robot y, mediante TF2, las detecciones proyectadas en coordenadas de mapa y la etiqueta con el identificador asignado por el *tracker*. Para mantener la fluidez, el sistema no redibuja todos los elementos en cada fotograma, sino que aplica una política de refresco periódico y limpieza de la ventana antes de volver a dibujar.

Al cerrar la sesión, se guardan los resultados de manera estructurada. En primer lugar, se almacena un archivo JSON con toda la información del historial de detecciones y los objetos finales registrados, incluyendo estadísticas de cuántos objetos fueron detectados, cuántos permanecen activos, cuántos totales se han visto y las últimas coordenadas en las que ha sido encontrado cada objeto. Además, se guarda un PNG con el mapa final en el que aparecen representadas todas las detecciones encontradas. Finalmente, se ejecuta la herramienta *map_saver_cli* de ROS2 para almacenar el mapa generado en formato *.pgm* junto a su archivo *.yaml*, de manera análoga al procedimiento seguido con YOLOv8.

5.5 Seguimiento de Objetos

Durante esta sección se abordará el cuarto bloque reflejado en el esquema de la Figura 5.1, dedicado al seguimiento de objetos. En este punto, además de emplear los sensores del robot como el LiDAR, a través del tópico */scan* y la cámara integrada con el tópico */camera/image_raw*, se hará uso de los resultados de la detección para que el robot no solo identifique los objetos presentes en su entorno, sino que también sea capaz de mantener la atención sobre ellos a lo largo del tiempo y desplazarse de forma reactiva en función de su posición, para desplazarse se enviarán órdenes de movimiento al tópico */cmd_vel*.

En términos generales, el objetivo de este bloque es implementar un sistema de seguimiento visual que combine visión artificial y control de movimiento, permitiendo al robot avanzar hacia un objeto específico, detenerse cuando está demasiado cerca, buscarlo en caso de pérdida y, si no consigue localizarlo, pasar a un modo de navegación autónoma en el que se priorice la exploración segura del entorno. Para ello, se ha hecho como en el apartado anterior, utilizando las dos redes neuronales profundas mencionadas, YOLOv8, que aporta rapidez y estabilidad en la detección, y Grounding DINO, que ofrece mayor flexibilidad al aceptar clases definidas en lenguaje natural.

5.5.1 YOLOv8

El código comienza creando un nodo de ROS2 denominado *object_tracker*, encargado de inicializar el puente que convierte imágenes de ROS a formato OpenCV. Además, se definen estructuras internas como variables de estado, listas de detecciones y el mecanismo de seguimiento de objetos. Finalmente, en esta fase se carga el modelo YOLOv8 que se empleará para la detección.

Tras la carga del modelo se procede a la selección del objeto a seguir. Para ello se abre una ventana mediante Tkinter en la que el usuario introduce la clase del objeto que desea rastrear. Esa clase se guarda internamente y servirá como filtro para las detecciones que genere

YOLOv8.

El nodo se suscribe a los tópicos mencionados anteriormente, tanto a la cámara como al LiDAR, y comienza a recibir datos. Cada nueva imagen se convierte a formato OpenCV y se procesa con YOLOv8. De entre los resultados se conservan únicamente las cajas correspondientes a la clase elegida, descartando aquellas con baja confianza o que no coincidan con la categoría seleccionada.

Para cada caja válida se calcula el centro y se asigna un identificador mediante el *tracker*. Si un objeto coincide en posición y clase con uno ya visto en fotogramas anteriores, su posición se actualiza suavizándola, manteniendo un historial de objetos activos e inactivos.

De entre todos los candidatos, el robot seguirá al objeto considerado más estable, es decir, aquel que acumula más detecciones consecutivas con un nivel de confianza suficiente. Si no se encuentra ningún objeto que cumpla estos criterios, el robot no entra en modo de seguimiento, pasando directamente al estado de búsqueda. Es necesario mencionar que, mientras el sistema no reciba imágenes de la cámara del robot, se mantendrá quieto, esperando a que se inicie la cámara para empezar con el seguimiento.

En cuanto al comportamiento del robot, se distinguen tres modos principales:

- **Tracking:** Se calcula el error horizontal entre el centro de la imagen de la cámara (x_c) y el centro del objeto detectado (x_o) mediante la fórmula:

$$e_x = x_o - x_c \quad (5.3)$$

Si $e_x < 0$, el objeto está a la izquierda y el robot gira en esa dirección, mientras que si $e_x > 0$, el objeto está a la derecha y el giro es hacia ese lado. Cuando $|e_x|$ se encuentra dentro de un margen predefinido, el objeto se considera centrado y el robot avanza hacia él. Durante este avance se comprueba constantemente la distancia frontal mínima medida por el LiDAR. Si esa distancia es menor que un umbral de seguridad, establecido como 1.5 m, se interpreta que el objeto está demasiado cerca y el robot se detiene para evitar una colisión.

- **Búsqueda:** Si el objeto se pierde de vista, el robot entra en un estado de búsqueda girando sobre sí mismo en la dirección donde se vio por última vez. Este estado se mantiene durante 20 segundos.
- **Navegación autónoma:** Si tras esos 20 segundos el objeto no se ha recuperado, el sistema activa el modo de navegación autónoma explicado previamente, en el que se prioriza la exploración segura del entorno.

Finalmente, el nodo muestra en una ventana de OpenCV la imagen de la cámara con las cajas de detección en color gris y el objeto que se está siguiendo resaltado junto a su identificación. También se dibuja una cruz en el centro de la imagen y dos líneas verticales que delimitan el margen de centrado. En una esquina de la ventana se indica el estado actual del robot, especificando si se encuentra en modo tracking, búsqueda o navegación.

5.5.2 Grounding DINO

En este caso el código arranca creando un nodo de ROS2 específico para el seguimiento con Grounding Dino. Al igual que en el caso anterior, se inicializan las estructuras internas necesarias para guardar detecciones, el *tracker* y las variables de estado. Sin embargo, a diferencia de YOLOv8, aquí la carga del modelo se realiza sobre la red Grounding DINO, lo que permite realizar detecciones a partir de descripciones en lenguaje natural en lugar de limitarse a una lista cerrada de clases.

Antes de comenzar con el procesamiento es necesario que el usuario introduzca el texto con la descripción del objeto que desea rastrear. Esta cadena se almacena y se utilizará como *prompt* en el modelo. El sistema permanece inactivo hasta que recibe tanto esa descripción como la primera imagen válida de la cámara, hasta ese momento el robot no emite ningún comando de movimiento.

El nodo se suscribe a los tópicos de la cámara y del LiDAR, además de preparar la publicación de comandos en el tópico */cmd_vel*. Cuando se recibe una imagen, esta se convierte a un formato de tensores compatibles con PyTorch utilizando las transformaciones que exige el modelo Grounding DINO. A diferencia de YOLOv8, aquí no se utiliza OpenCV para la detección, aunque sí puede emplearse para algunas conversiones básicas. El modelo procesa la imagen junto con el texto introducido por el usuario y devuelve un conjunto de cajas delimitadoras con su puntuación de confianza. Posteriormente, dichas cajas se reescalán a las dimensiones originales de la imagen para mantener la correspondencia espacial.

Al igual que con YOLOv8, para cada caja se toma el centro como referencia de posición y se asocia a un identificador único a través del *tracker*. Si una nueva detección está próxima a otra anterior con la misma etiqueta, se considera el mismo objeto y su posición se actualiza suavizándola. En caso contrario, se crea una nueva entrada. Todo esto se realiza mientras se mantiene un historial que diferencia objetos activos e inactivos.

El robot selecciona como objetivo de seguimiento el objeto más estable, es decir, aquel que acumula más detecciones consecutivas con suficiente confianza. Si no se encuentra ningún objeto válido, no se entra en modo de seguimiento y se pasará directamente al estado de búsqueda.

Los modos de comportamiento son exactamente los mismos a los descritos en el caso de YOLOv8:

- **Tracking:** Se calcula el error horizontal entre el centro de la imagen y el centro del objeto detectado mediante la expresión mostrada en la subsección anterior. Y el resultado se tratará de igual forma, si el error es negativo el robot gira hacia la izquierda, y si es positivo hacia la derecha. Al igual que con YOLOv8, cuando el error es menor que un margen de tolerancia se considera que el objeto está centrado, en cuyo caso el robot avanza. Durante este avance se comprueba el valor mínimo devuelto por el LiDAR en el sector frontal. Si la distancia medida cae por debajo de un umbral de seguridad, se interpreta que el objeto está demasiado cerca y el robot se detiene de inmediato.

- **Búsqueda:** Si el objeto se pierde de vista, el robot gira sobre sí mismo en la dirección en la que fue visto por última vez, manteniendo este estado durante 20 segundos.
- **Navegación autónoma:** Si pasado este tiempo el objeto no se ha recuperado, se activa el modo de navegación autónoma explicado previamente.

Por último, para la visualización se emplea Matplotlib en lugar de OpenCV. En la ventana mostrada aparecen las cajas de detección y las etiquetas correspondientes, resaltando en color el objeto seguido, además de incluir una indicación textual del estado actual del robot: tracking, búsqueda o navegación.

5.6 Interfaz

El último objetivo planteado en el proyecto ha sido el desarrollo de una interfaz gráfica que centralizase el control del robot, facilitando al operador la gestión de todos los procesos sin necesidad de utilizar múltiples terminales. Para ello se diseñó una aplicación en Tkinter, que ha ido evolucionando desde versiones preliminares con funcionalidades muy básicas hasta llegar a la versión final. Esta versión incorpora un diseño más visual y estructurado, lo que mejora tanto la usabilidad como la rapidez de interacción. Además, cada proceso se ejecuta en un hilo independiente, permitiendo lanzar y detener módulos de forma aislada, con indicadores de estado en tiempo real.

La interfaz cuenta con un amplio conjunto de botones que permiten ejecutar y detener procesos según las necesidades del usuario. Estos controles están distribuidos en diferentes bloques. En primer lugar, los controles básicos permiten:

- ”Lanzar RVIZ”: abre la herramienta de visualización RVIZ en ROS2.
- ”Mover Robot”: activa la navegación autónoma explicada anteriormente.
- ”Parar”: uno de ellos detiene la ejecución de RVIZ mientras que el otro interrumpe la navegación autónoma.
- ”Guardar Mapa PGM”: guarda en memoria el mapa generado en formato *.pgm* y *.yaml*.

Dentro de la sección de detección de objetos se encuentran los botones dedicados a lanzar la detección de objetos con cada una de las redes neuronales integradas:

- ”YOLOv8”: inicia el proceso de detección de objetos con este modelo.
- ”Grounding DINO”: lanza la detección basada en lenguaje natural con el modelo Grounding DINO.
- ”Parar y Guardar Mapa”: detiene la detección y guarda un mapa con los resultados obtenidos de cada red neuronal.

En el apartado de seguimiento de objetos se incluyen controles equivalentes para ambos modelos:

- "YOLOv8": inicia el seguimiento de objetos con YOLOv8.
- "Grounding DINO": ejecuta el seguimiento con Grounding DINO.
- "Parar": detiene el seguimiento de cada una de las redes neuronales.



Figura 5.12: Controles básicos interfaz

Para facilitar la supervisión, se incorpora un panel de estado de procesos donde se muestra visualmente si cada módulo está detenido, en ejecución o ha finalizado con éxito. En este panel aparecen los indicadores correspondientes a:

- "RVIZ": indica si la visualización está activa.
- "Mover Robot": refleja el estado de la navegación autónoma.
- "Guardar Mapa": muestra si se ha realizado correctamente el guardado del mapa.
- "Detección - YOLOv8": informa sobre la ejecución del detector YOLOv8.
- "Detección - Grounding DINO": muestra si el detector con Grounding DINO está en marcha.
- "Seguimiento - YOLOv8": refleja el estado del seguimiento con YOLOv8.
- "Seguimiento - Grounding DINO": indica si el seguimiento con Grounding DINO se encuentra activo.

La interfaz también incluye un espacio de visualización de mapas, dividido en dos apartados:

- "Cargar Mapa PNG": permite visualizar un mapa en formato PNG con las detecciones realizadas.
- "Cargar Mapa PGM": carga y muestra un mapa en formato PGM dentro de la aplicación ajustado al área de visualización.



Figura 5.13: Estado de procesos en la interfaz

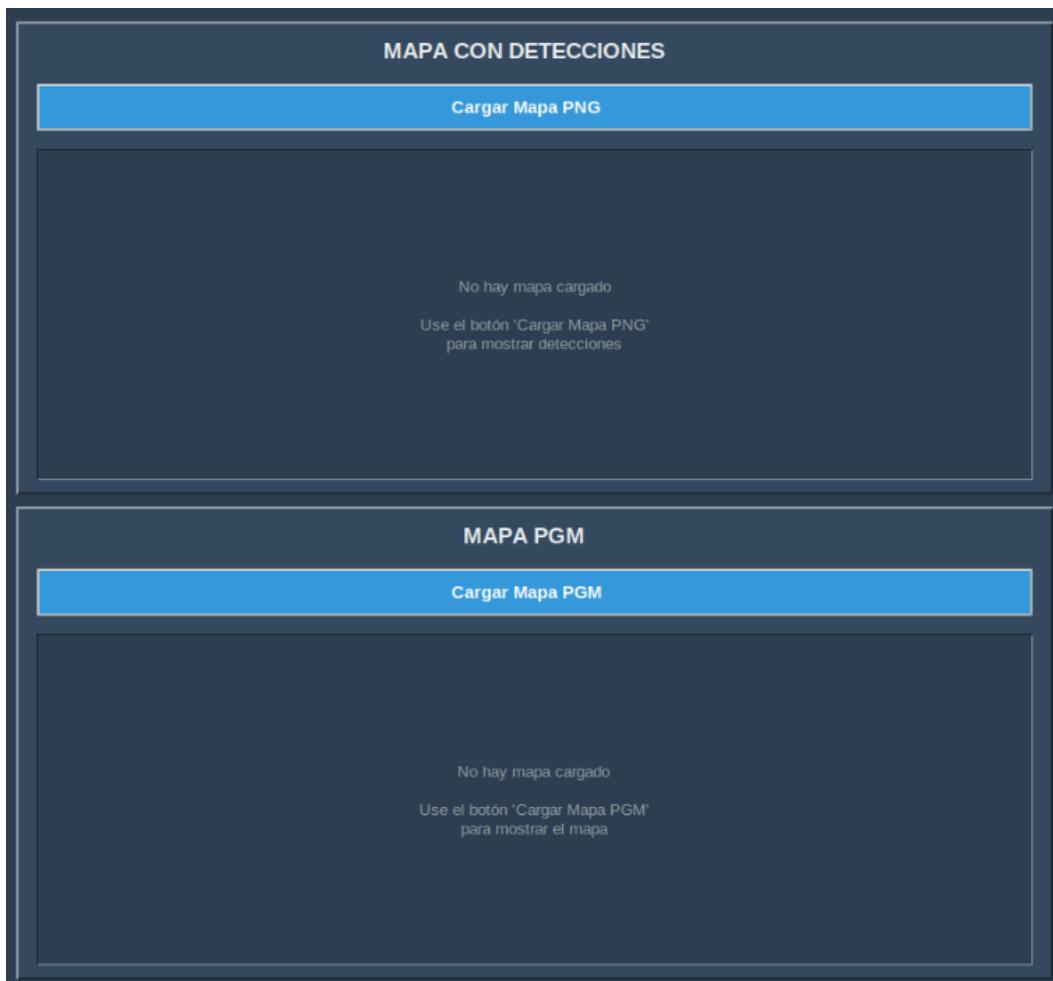


Figura 5.14: Carga de mapas en la interfaz

Como medida de seguridad se incorpora un botón de emergencia:

- ”PARAR TODOS LOS PROCESOS”: detiene inmediatamente cualquier ejecución en curso, garantizando un apagado rápido del sistema.

Finalmente, un botón de cierre general permite finalizar la aplicación:

- "Cerrar Interfaz": cierra la ventana, detiene los procesos activos y libera los recursos empleados.



Figura 5.15: Botones de cierre en la interfaz

De esta manera, la interfaz no solo centraliza en un único entorno todas las funciones de navegación, detección, seguimiento y visualización, sino que también aporta un mayor grado de usabilidad y seguridad para el operador.

6 Resultados

En este capítulo se presentan los resultados experimentales obtenidos durante el desarrollo del proyecto, cuyo objetivo principal ha sido la integración de un sistema de SLAM con capacidades de detección y seguimiento de objetos en el robot articulado Unitree Go2. La finalidad de esta sección es evaluar el rendimiento y la fiabilidad de cada una de las funcionalidades implementadas, analizando su comportamiento en condiciones de operación real. Para ello, los resultados tratarán de evaluar la navegación autónoma basada en LiDAR, la eficacia del sistema de evitación de obstáculos, la capacidad de detección visual mediante modelos de redes neuronales profundas y su comparación práctica, hasta la validación del módulo de seguimiento de objetos. Además estos resultados permiten establecer posibles mejoras para trabajos futuros.

6.1 Entorno de pruebas

Las pruebas experimentales se realizaron en la primera planta de la Escuela Politécnica Superior III (EPS III) de la Universidad de Alicante. En este espacio se dispusieron diversos obstáculos a lo largo del recorrido con el objetivo de comprobar la capacidad del robot para desenvolverse en un entorno desconocido, adaptándose a trayectorias variables y evitando colisiones mientras avanzaba de manera autónoma. Para ello se utilizó el escenario representado en la Figura 6.1.



Figura 6.1: Entorno de pruebas con obstáculos

6.2 Experimentación

Para evaluar el rendimiento del sistema desarrollado se realizaron pruebas experimentales centradas en las funcionalidades principales del robot Unitree Go2, siguiendo los objetivos planteados en el Capítulo 3. Estas pruebas se enfocaron en la navegación autónoma, la percepción visual y la interacción mediante la interfaz gráfica.

Se plantearon los siguientes objetivos de experimentación:

- Evaluar la navegación autónoma y el sistema SLAM:
 - ¿El sistema SLAM genera mapas precisos y consistentes del entorno?
 - ¿El robot evita colisiones de manera efectiva durante la navegación autónoma?
 - ¿Se adapta correctamente a trayectorias variables y obstáculos imprevistos?
- Comprobar la detección, clasificación y seguimiento de objetos:
 - ¿El sistema detecta correctamente los objetos presentes en el entorno?
 - ¿Se clasifican adecuadamente los objetos según los modelos entrenados?
 - ¿El seguimiento de objetos es preciso y consistente durante el movimiento del robot?
 - ¿El procesamiento de imágenes se realiza en tiempo real sin afectar la navegación?
- Verificar la interfaz gráfica y la interacción con el sistema:
 - ¿La interfaz permite gestionar de manera intuitiva todos los procesos?
 - ¿Se visualizan correctamente los objetos detectados y su seguimiento en tiempo real?

6.3 Resultados obtenidos

6.3.1 Navegación autónoma

Se realizó una evaluación específica del sistema de evitación de obstáculos para comprobar que, durante la navegación autónoma, el robot Unitree Go2 pudiera detectar obstáculos en su entorno y esquivarlos adecuadamente, garantizando un desplazamiento seguro.

Durante las primeras pruebas en el laboratorio, se observaron algunas colisiones debido a la falta de experiencia en el procesamiento de los datos obtenidos por el LiDAR y a la necesidad de ajustar las velocidades máximas del robot. Estas dificultades iniciales permitieron identificar la manera correcta de tratar las mediciones del láser y aplicar las medidas de seguridad necesarias para la navegación autónoma.

Una vez implementadas las correcciones, se realizaron pruebas sistemáticas en el entorno de la EPS III, donde se dispusieron obstáculos de diferentes tamaños y materiales, incluyendo sillas, cajas, etc. Durante estas pruebas se evaluó la capacidad del robot para:

- Detectar obstáculos estáticos y dinámicos a distintas distancias.
- Replanificar su trayectoria de forma eficiente ante cambios inesperados en el entorno.
- Mantener una velocidad de desplazamiento segura sin comprometer la estabilidad.

Los resultados mostraron que el robot pudo evitar todos los obstáculos, adaptando su trayectoria con precisión incluso en corredores estrechos o ante elementos imprevistos. Además, se observó que la combinación del sistema de navegación basado en LiDAR con la lógica de evasión implementada permitió al robot mantener un comportamiento estable, minimizando oscilaciones y movimientos bruscos al esquivar objetos. Entre las limitaciones detectadas se encuentra la dificultad para detectar superficies reflectantes o poco reactivas al láser; por ejemplo, las cristalerías presentes en la EPS III no eran detectadas inicialmente, provocando algunos choques al inicio de las pruebas.

En conclusión, tras realizar los ajustes necesarios en el procesamiento de los datos del LiDAR, la calibración de velocidades y la implementación de las medidas de seguridad, el robot demostró un desempeño completamente fiable y seguro. El sistema de navegación y evitación de obstáculos funciona correctamente en todas las condiciones probadas, adaptando su trayectoria con precisión ante obstáculos conocidos y desconocidos, y evidenciando la eficacia y estabilidad del enfoque implementado.

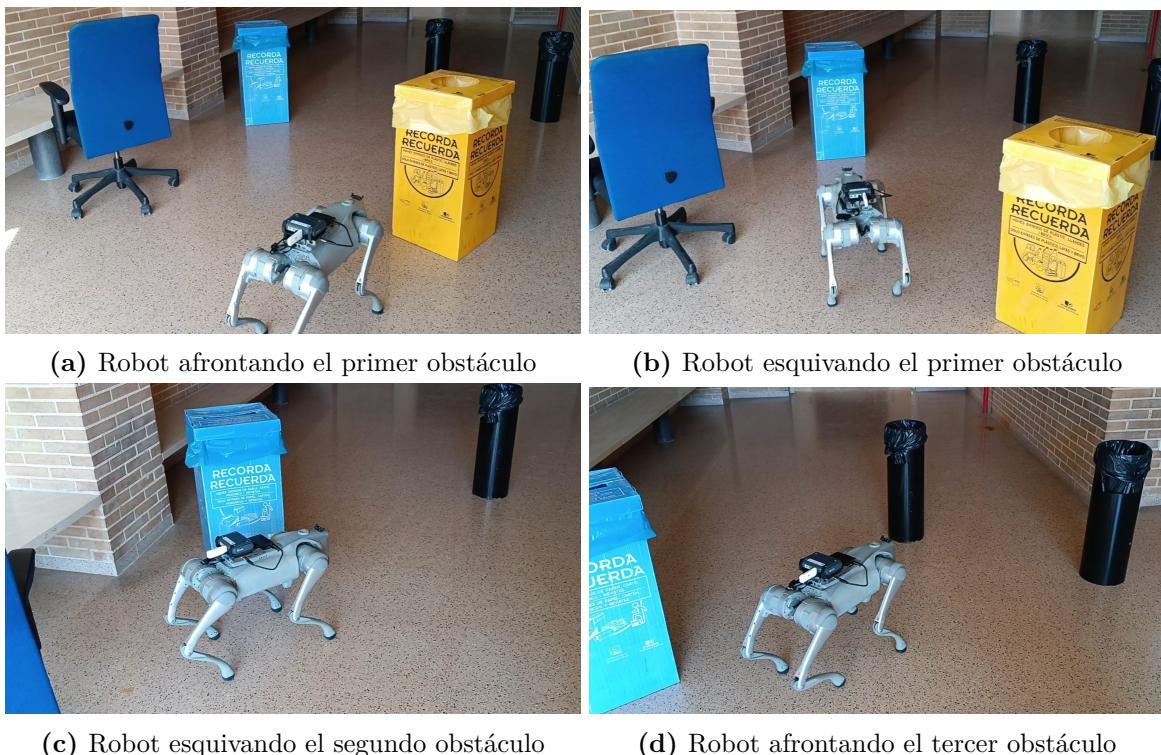


Figura 6.2: Evitación de obstáculos

6.3.2 Detección de objetos

En esta sección se presentan los resultados obtenidos durante la fase de detección de objetos, en la cual se evaluó el rendimiento de los dos modelos de redes neuronales profundas integrados en el sistema: YOLOv8 y Grounding DINO. El propósito principal de esta evaluación fue comprobar la capacidad del robot para identificar, localizar y proyectar sobre el mapa los objetos presentes en su entorno utilizando de manera conjunta la cámara integrada y el sensor LiDAR.

En términos generales, ambos modelos fueron capaces de detectar correctamente los objetos seleccionados, aunque se observaron diferencias notables en velocidad de ejecución, precisión de las cajas, flexibilidad semántica y requisitos computacionales. Cabe destacar que Grounding DINO no resulta aconsejable para tareas en tiempo real, ya que depende de manera obligatoria de GPU y aun así presenta un mayor tiempo de inferencia. En contraste, YOLOv8 ofrece detecciones rápidas y precisas incluso en CPU, lo que lo hace más adecuado para aplicaciones que requieren rapidez, como la navegación autónoma.

La Tabla 6.1 resume las principales características y diferencias observadas entre ambos modelos.

Aspecto	YOLOv8	Grounding DINO
Velocidad	Muy rápido, ejecutable en CPU	Lento, requiere GPU para detecciones en tiempo real
Precisión de las cajas	Ajustadas y bien definidas	Correctas pero depende del reescalado de la imagen
Flexibilidad semántica	Limitado a las clases vistas durante el entrenamiento	Capaz de detectar objetos no vistos previamente gracias a su entendimiento semántico
Problemas encontrados	Procesar todas las clases simultáneamente ralentiza la detección, solucionado con entrada Tkinter para filtrar objetos	Detecciones iniciales desescaladas (solucionado con ajuste de tensor) y muy dependiente de GPU
Uso recomendado	Entornos donde se priorice la rapidez y el bajo coste computacional	Entornos donde se requiera mayor generalización y flexibilidad en las detecciones

Cuadro 6.1: Comparación entre YOLOv8 y Grounding DINO en la tarea de detección de objetos.

Además de esta comparativa cualitativa, se midieron métricas cuantitativas que permiten evaluar de forma más objetiva el rendimiento de ambos modelos. En particular, se analizaron el tiempo medio de detección por imagen, los fotogramas procesados por segundo, la precisión de las detecciones y el consumo de memoria, entre otros. Los resultados obtenidos se

muestran en la Tabla 6.2.

Métrica	YOLOv8	Grounding DINO
Tiempo medio de detección	0.012 s	0.85 s
FPS alcanzados	82	1.18
Precisión (clase: "person")	0.91	0.87
Memoria RAM utilizada	1.2 GB	9.6 GB
Uso de GPU	2.5 GB	>8 GB
Tamaño del modelo	90 MB	1.2 GB
Dependencia de hardware	CPU y GPU (opcional)	Requiere GPU
Escalabilidad a dispositivos embebidos	Sí, es compatible con Jetson/Raspberry Pi	No viable en hardware limitado

Cuadro 6.2: Resultados cuantitativos en la comparación de YOLOv8 y Grounding DINO.

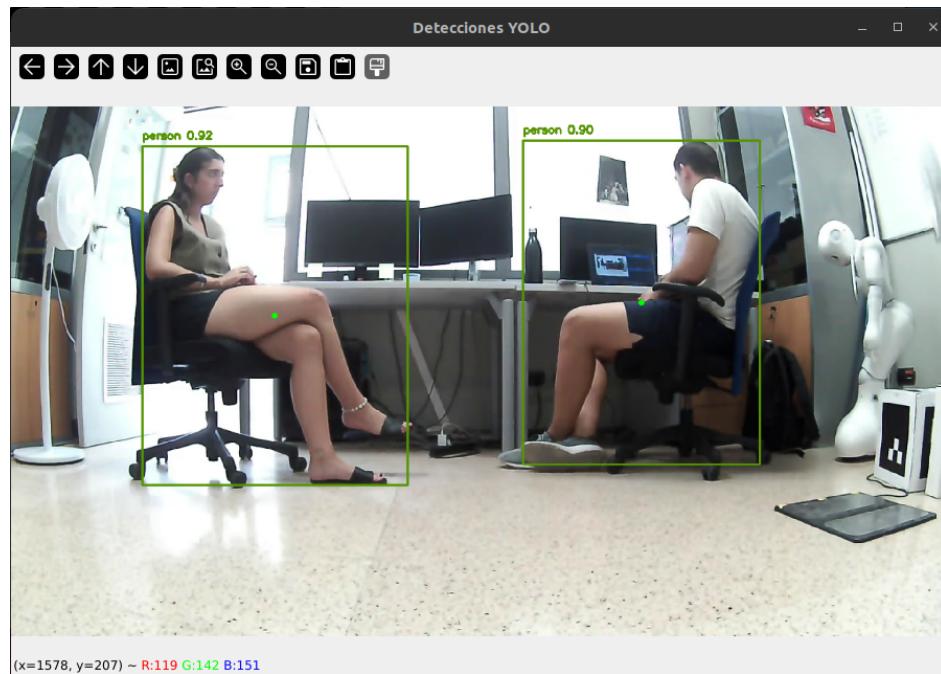
Los ejemplos visuales de la detección de personas y sillas con ambos modelos se presentan en las Figuras 6.3 y 6.4. En la Figura 6.3 se aprecia cómo YOLOv8 ofrece detecciones más precisas y ajustadas en un escenario de tiempo real, lo que confirma su idoneidad para aplicaciones en las que la rapidez es un factor determinante.

Por otro lado, en la Figura 6.4 se observa que Grounding DINO alcanza un mejor rendimiento cuando las imágenes se procesan fuera de un entorno en tiempo real, logrando una correcta localización de los objetos incluso sin haber sido entrenado explícitamente sobre ellos. Esta capacidad de generalización le otorga una ventaja en contextos donde la flexibilidad semántica es prioritaria, aunque a costa de un mayor tiempo de inferencia.

El análisis de la Tabla 6.2 permite profundizar en estas diferencias de manera cuantitativa. Los resultados muestran que YOLOv8 supera ampliamente a Grounding DINO en velocidad, alcanzando más de 80 Fotogramas por Segundo (FPS) frente a poco más de 1 FPS, lo que lo hace viable para aplicaciones en entornos dinámicos. Asimismo, en el caso de YOLOv8, el menor consumo de memoria RAM y GPU, junto con un tamaño de modelo significativamente inferior, facilitan su ejecución en dispositivos con recursos limitados, incluyendo plataformas embebidas como Jetson o Raspberry Pi. En contraste, Grounding DINO requiere una GPU dedicada con gran capacidad de memoria, lo que restringe su uso a estaciones de trabajo potentes y lo descarta para escenarios de robótica móvil en tiempo real.

En conclusión, tanto los resultados visuales como los cuantitativos refuerzan la misma

idea: YOLOv8 se posiciona como el modelo más eficiente y práctico en entornos dinámicos que requieren detecciones rápidas, mientras que Grounding DINO resulta más adecuado en situaciones donde el tiempo de procesamiento no es crítico, pero sí lo es la capacidad de detectar una amplia variedad de objetos.

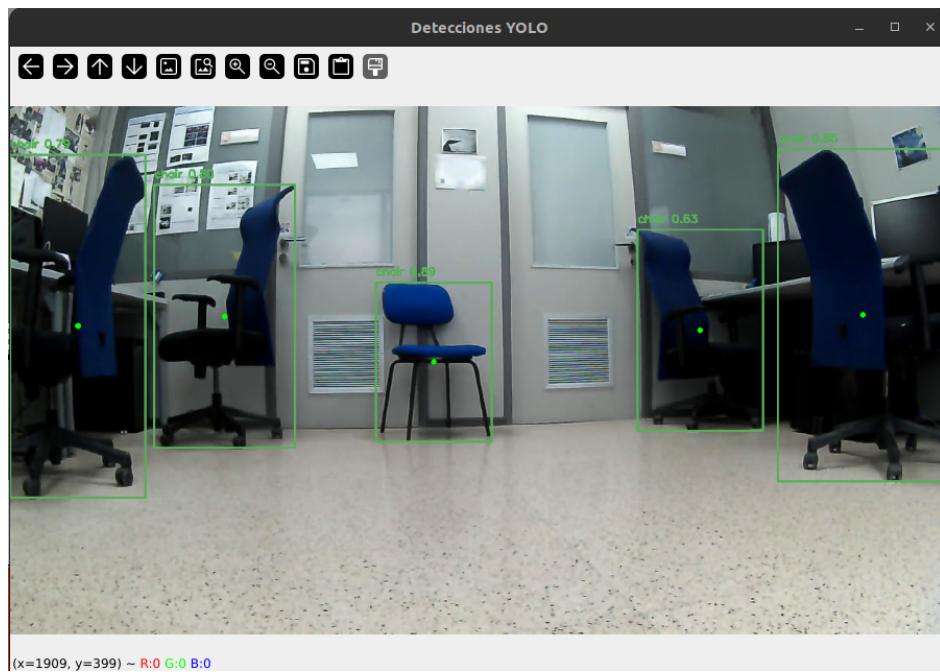


(a) Detección de personas con YOLOv8

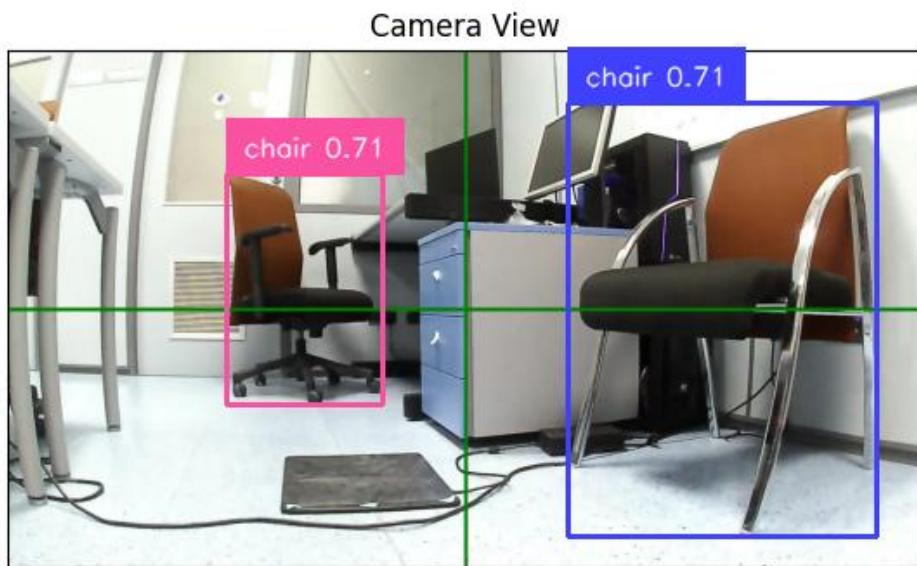


(b) Detección de personas con Grounding DINO

Figura 6.3: Comparativa de detección de personas



(a) Detección de sillas con YOLOv8



(b) Detección de sillas con Grounding DINO

Figura 6.4: Comparativa de detección de sillas

Respecto a los mapas generados durante la detección de objetos, se identificaron diversos problemas durante su implementación. El primero de ellos consistió en determinar cómo realizar anotaciones gráficas sobre una copia del mapa, de manera que fuese posible representar con puntos la posición de los objetos detectados.

El segundo problema surgió porque el sistema, en su diseño inicial, registraba cada detección en la imagen como un objeto nuevo, independientemente de si ya había sido detectado previamente. Esto provocaba que en el mapa apareciesen múltiples instancias de un mismo objeto. Para solventar esta limitación se incorporó un *tracker*, que permitió hacer un seguimiento continuo de cada detección y mantener su identidad en el tiempo.

Por último, se detectó que el robot no gestionaba adecuadamente la pérdida temporal de un objeto. En esos casos, se mantenía su última posición registrada, pero si un nuevo objeto aparecía cerca de esa ubicación, el sistema lo interpretaba como una detección diferente, reproduciendo así el error anterior. Este problema se resolvió introduciendo un mecanismo de memoria en el código, mediante el cual si un objeto aparecía en las proximidades de la última posición registrada de otro de la misma clase, se consideraba como la misma identidad y se actualizaba su posición en lugar de generarse una detección duplicada.

Una vez solucionados los problemas descritos previamente, se obtuvieron los mapas mostrados en las Figuras 6.5a y 6.5b. En ambas imágenes se representa el proceso de mapeado realizado durante las tareas de detección de objetos. En el caso de YOLOv8, durante la ejecución aparecieron tres personas, que quedaron registradas en el mapa mediante marcadores de color verde. En contraste, con Grounding DINO únicamente apreció una persona, también reflejada en el correspondiente mapa.

En la representación gráfica, los círculos rojos indican las detecciones proporcionadas por la red neuronal, el cuadrado rojo señala la posición inicial desde la que parte el robot, y los círculos azules corresponden a los obstáculos localizados a lo largo del proceso de mapeado.

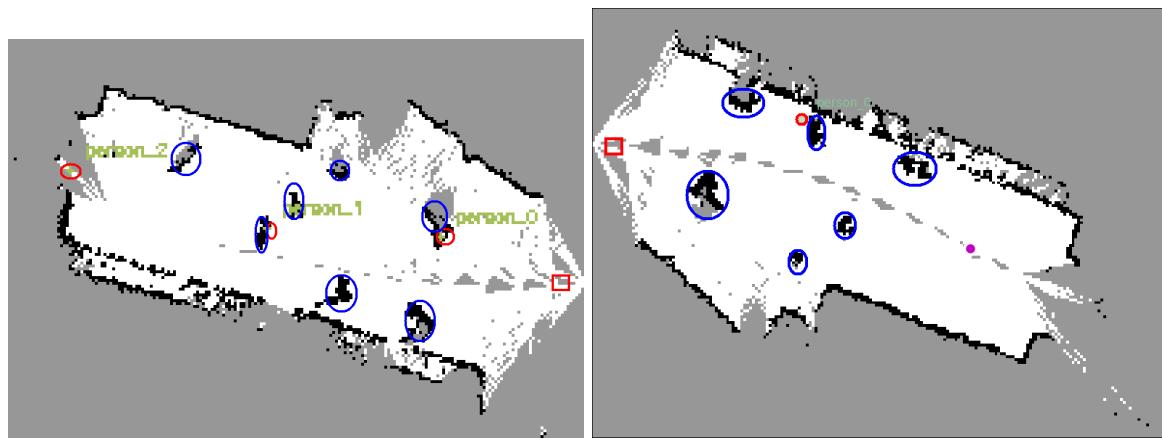


Figura 6.5: Resultados de mapeado con detección de objetos

Por último las ventanas que se abrirán cuando se ejecuten los procesos serán las mostradas en la Figura 6.6, donde para ambos procesos se abren el mapa indicando las detecciones y la imagen captada por la cámara con las detecciones.

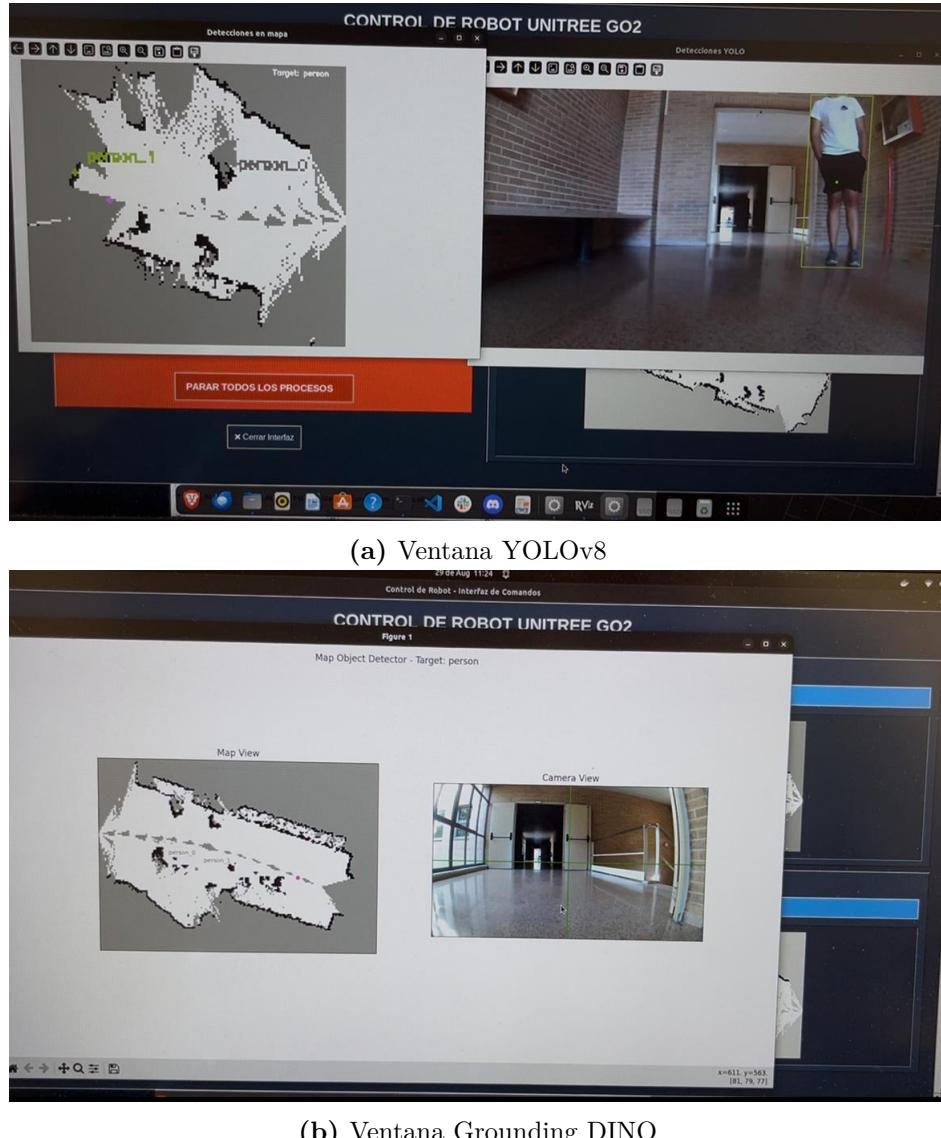


Figura 6.6: Ventanas mostradas

6.3.3 Seguimiento de objetos

Tras la fase de detección de objetos, se implementó un sistema de seguimiento que permite mantener la identidad de cada objeto a lo largo del tiempo y del desplazamiento del robot. Mientras que la detección proporciona información puntual en cada fotograma, el seguimiento asegura la coherencia temporal de las detecciones, evitando duplicados y permitiendo conocer la trayectoria seguida por cada objeto en el entorno.

Esta funcionalidad es especialmente relevante en escenarios dinámicos, donde los objetos pueden desaparecer momentáneamente del campo de visión o desplazarse dentro del entorno. De este modo, el robot no solo identifica la presencia de un objeto en un instante concreto,

sino que también puede relacionar detecciones sucesivas, actualizar su posición y conservar la información de aquellos objetos que hayan dejado de estar visibles temporalmente.

Como se mencionó en el apartado de desarrollo, en el caso de YOLOv8 se abre una ventana que permanecerá vacía hasta recibir la primera imagen de la cámara del robot. Esto permite al usuario asegurarse de que el robot no comenzará a moverse hasta que haya recibido imagen y se haya indicado la clase objetivo. Por su parte, Grounding DINO no abrirá ninguna ventana hasta que se reciba un prompt por parte del usuario o la primera imagen de la cámara.

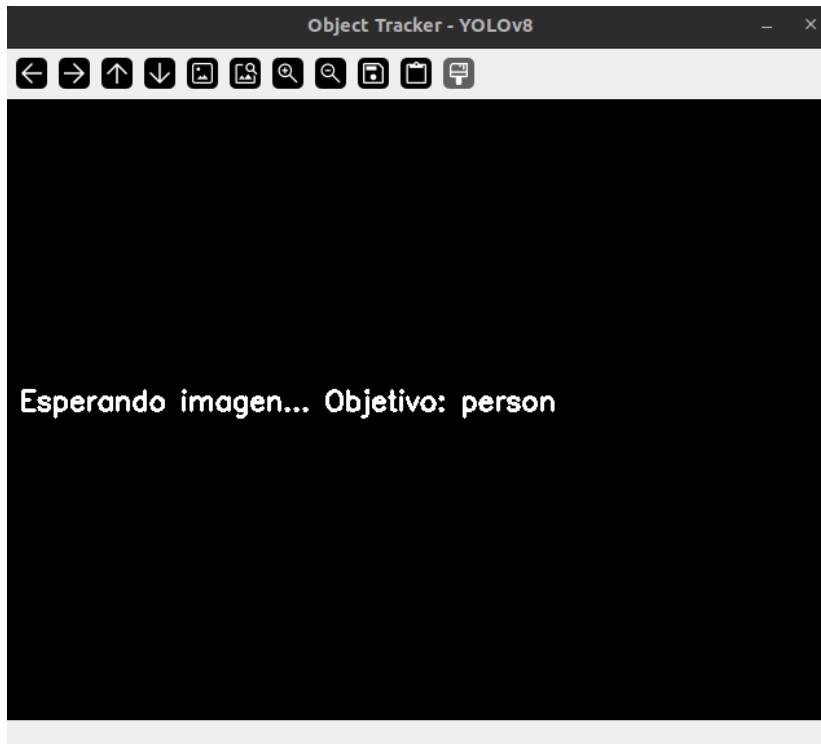
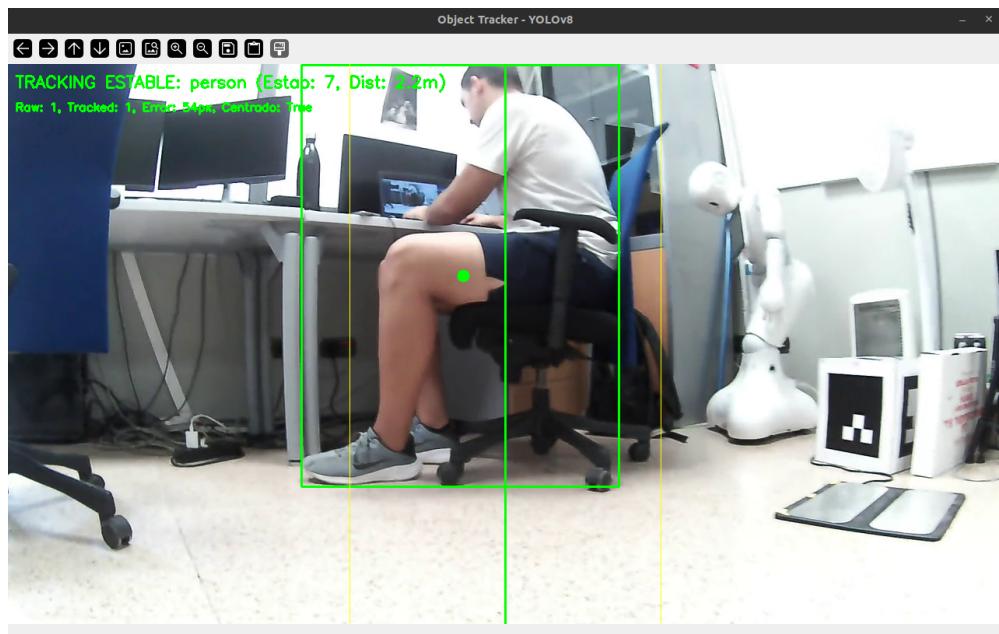


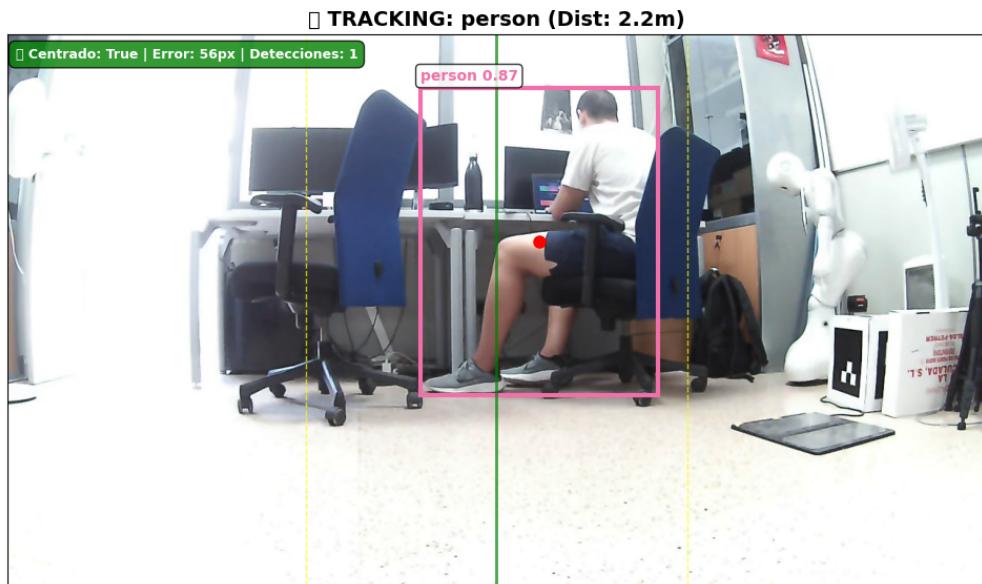
Figura 6.7: Pantalla de espera en YOLOv8

Una vez recibida la imagen, se abren las ventanas de seguimiento mostradas en las Figuras 6.8a y 6.8b. En estas ventanas se visualiza la imagen de la cámara con las detecciones superpuestas y líneas verticales que indican el centro de la imagen y los márgenes de seguridad definidos para considerar si el objeto está centrado.

En el caso de YOLOv8, la ventana también muestra el estado del robot (Tracking, Búsqueda o Navegación autónoma), si el objeto está centrado, la distancia al objeto y el número de frames consecutivos en los que se ha detectado para considerarlo estable. Para Grounding DINO, la misma información se presenta en una ventana de matplotlib en lugar de OpenCV, manteniendo la consistencia de los datos mostrados al usuario.



(a) Seguimiento de objetos con YOLOv8



(b) Seguimiento de objetos con Grounding DINO

Figura 6.8: Ventanas de seguimiento de objetos

Durante la implementación del seguimiento de objetos, se identificó un problema principal: la aparición de falsos positivos o de múltiples instancias de la misma clase, lo que provocaba que el robot no supiera qué objeto debía seguir en cada momento.

Para resolver esta situación, se implementó un criterio de selección basado en la fiabilidad de cada detección. Concretamente, el sistema prioriza el seguimiento del objeto que presente el mayor porcentaje de acierto o que haya sido detectado durante más frames consecutivos. De este modo, se considera que dicho objeto es el más confiable y se minimiza la probabilidad de que el robot siga un falso positivo.

6.3.4 Interfaz

Para evaluar los resultados obtenidos durante la experimentación con la interfaz, se analizaron dos aspectos principales: su correcto funcionamiento y su facilidad de uso. La interfaz desarrollada se muestra en la Figura 6.9.

En la imagen puede observarse el diseño final de la interfaz, que integra todos los botones descritos en el apartado anterior, así como un panel de control de estados que permite al usuario conocer en todo momento qué procesos se encuentran en ejecución. Asimismo, se añadieron dos menús para la carga de mapas en formatos *.png* y *.pgm*, con el fin de verificar si las detecciones se proyectan adecuadamente sobre el mapa. Finalmente, se incorporó un control de emergencia que posibilita cerrar todos los procesos de manera inmediata en caso de que surja algún problema.

Tras la evaluación, se comprobó que los objetivos planteados para la interfaz se cumplen satisfactoriamente. Por un lado, la simplicidad de su diseño la hace muy intuitiva, de modo que el usuario puede comprender su funcionamiento de forma inmediata. Por otro, las pruebas realizadas confirmaron que la interfaz responde correctamente: al presionar un botón, se ejecuta la función correspondiente, tal como se indica en su etiqueta.

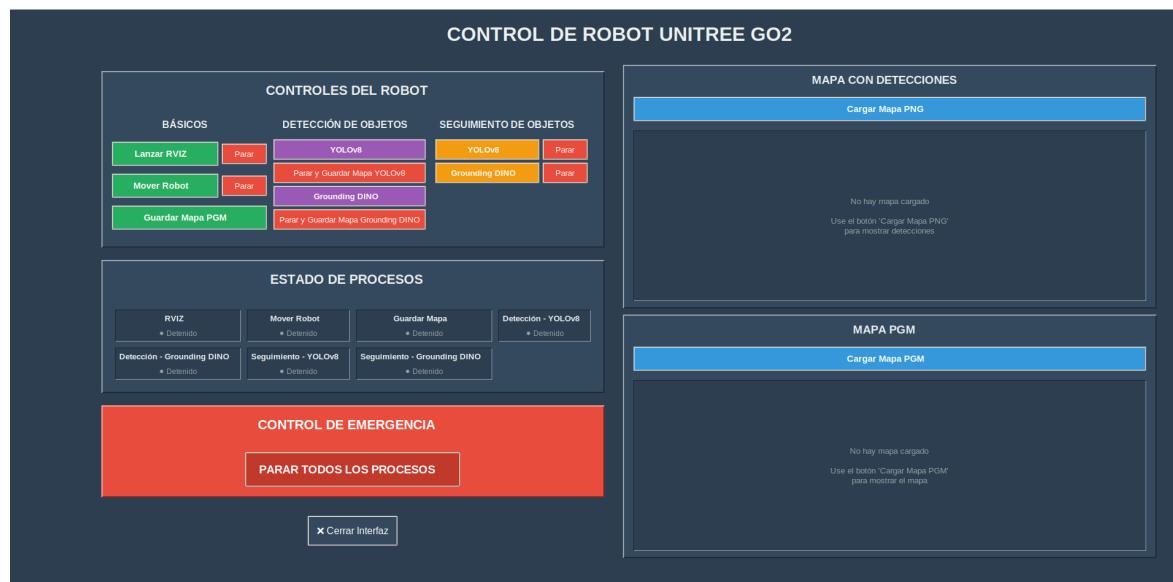


Figura 6.9: Interfaz gráfica final desarrollada para el control del robot

El resultado final del proyecto puede visualizarse en el vídeo disponible en el siguiente enlace, donde se muestra el funcionamiento general del sistema: <https://youtu.be/Ee76DDSeNbM>

Además, el código completo se encuentra en el siguiente repositorio de GitHub, que contiene todos los archivos necesarios para la ejecución del proyecto: <https://github.com/JoseCarlosPenMac/TFM---SLAM-con-el-robot-articulado-Unitreee-Go2.git>

7 Conclusiones

En el desarrollo de este proyecto se ha abordado la integración de un sistema SLAM en el robot articulado Unitree Go2, complementado con una navegación autónoma, detección y seguimiento de objetos y una interfaz gráfica que facilita la interacción con el sistema. A lo largo del trabajo, se han implementado, probado y ajustado distintas funcionalidades con el objetivo de garantizar un rendimiento fiable y una experiencia de uso intuitiva. En este apartado se presenta un resumen del proceso de desarrollo, una discusión de los resultados alcanzados y una propuesta de posibles líneas de mejora para trabajos futuros.

Tras revisar los objetivos planteados al inicio, puede afirmarse que todos ellos se han cumplido de manera satisfactoria. En lo que respecta a la navegación autónoma y al sistema SLAM, el robot ha demostrado ser capaz de generar mapas consistentes y desplazarse de forma segura en entornos desconocidos. Si bien en las primeras pruebas se produjeron colisiones, los ajustes realizados en el procesamiento del LiDAR y en los parámetros de velocidad permitieron lograr un sistema de evitación de obstáculos estable y preciso, capaz de adaptarse a trayectorias variables y resolver con eficacia situaciones imprevistas.

En relación con la detección y el seguimiento de objetos, la integración de modelos de visión por computador como YOLOv8 y Grounding DINO permitió comprobar la eficacia de ambos enfoques. YOLOv8 resultó más adecuado para aplicaciones en tiempo real debido a su rapidez y precisión, mientras que Grounding DINO destacó por su flexibilidad semántica, aunque con mayores requerimientos computacionales. El seguimiento de objetos, por su parte, resolvió la dificultad inicial de falsos positivos y de múltiples instancias de la misma clase mediante un criterio de selección basado en la fiabilidad de la detección, lo que aseguró trayectorias coherentes y una identificación más robusta.

La interfaz gráfica desarrollada se confirmó como una herramienta intuitiva y funcional, ya que permitió al usuario gestionar todos los procesos del sistema de manera sencilla, cargar mapas en diferentes formatos, visualizar en tiempo real las detecciones proyectadas sobre el entorno y detener la ejecución de manera segura mediante un control de emergencia. Su simplicidad de uso y correcto funcionamiento fueron corroborados durante las pruebas experimentales, lo que la convierte en un componente esencial para la interacción del operador con el sistema.

En conjunto, los resultados obtenidos validan la eficacia del sistema diseñado, logrando la integración de navegación, detección, seguimiento e interacción en un mismo entorno operativo. No obstante, el proyecto abre también la puerta a nuevas líneas de investigación y mejora, como la optimización de la detección de superficies reflectantes o transparentes mediante la fusión del LiDAR con técnicas de visión por computador, la implementación de un sistema de

navegación que, una vez generado el mapa y conociendo la posición de los objetos detectados almacenada en el archivo JSON, permita al robot desplazarse directamente hacia el objeto indicado por el usuario evitando los obstáculos conocidos o los nuevos obstáculos que puedan aparecer, y el uso de algoritmos de seguimiento más avanzados que incrementen la robustez en escenarios dinámicos o con múltiples objetivos en movimiento.

En definitiva, el proyecto ha alcanzado con éxito los objetivos planteados, aportando un sistema robusto, flexible y escalable que sienta las bases para futuras mejoras en el ámbito de la navegación autónoma y la percepción multimodal en robots móviles.

Bibliografía

- 3CX. (2025). *¿qué es WebRTC?* <https://www.3cx.es/voip-sip/que-es-webrtc/>.
- Abizov, N. (2025). *go2_ros2_sdk: Unofficial ros2 sdk support for unitree go2 air/pro/edu.* GitHub repository. Descargado de https://github.com/abizovnuralem/go2_ros2_sdk
- Alexey Dosovitskiy, A. K. D. W. X. Z. T. U. M. D. M. M. G. H. S. G. J. U., Lucas Beyer, y Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. En *International conference on learning representations (iclr)*. Descargado de <https://arxiv.org/abs/2010.11929>
- Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., ... Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6), 1309–1332. doi: 10.1109/TRO.2016.2624754
- Documentation, R. (2018). *Ros documentation: rviz.* Descargado de <https://wiki.ros.org/rviz>
- Durrant-Whyte, H., y Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, 13(2), 99-110. doi: 10.1109/MRA.2006.1638022
- Garage, W. (2025). *tf2: Transform library for ros.* Descargado de <https://wiki.ros.org/tf2>
- García Ulloa, I. A., y Nieves Guerrero, W. J. (2024). *Desarrollo de un robot autónomo con sistema de mapeo y detección de obstáculos mediante sensor lidar* (Tesis de Grado). Universidad Politécnica Salesiana. Descargado de <http://dspace.ups.edu.ec/handle/123456789/27937>
- Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 580–587). doi: 10.1109/CVPR.2014.81
- Google WebRTC Team. (2025). *Webrtc: Comunicación en tiempo real para la web.* <https://webrtc.org/?hl=es-419>. Google. Descargado de <https://webrtc.org/?hl=es-419>
- Gordon, C., Encalada, P., Lema, H., León, D., y Chicaiza, D. (2019). Objects detection techniques and deep neural networks for autonomous navigation of robot youbot kuka with signaling. *Iberian Journal of Information Systems and Technologies*, 305–316. Descargado de <https://www.proquest.com/openview/ca355d43bc1ac0d2721fc1078849f5f6/1?pq-origsite=gscholar&cbl=1006393> doi: 10.1007/978-3-030-29513-4_70

- IBM. (2025). *¿qué es pytorch?* Descargado de <https://www.ibm.com/es-es/topics/pytorch>
- Invelon Technologies SL. (2025). *Robots cuadrúpedos.* <https://robotics.invelon.com/robots-cuadrupedos>.
- Lidar, U. R. (2025). *Unitree 4d lidar l1.* Descargado de <https://www.unitree.com/LiDAR>
- Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., y cols. (2023). *Grounding dino: Marrying dino with grounded pre-training for open-set object detection.* <https://github.com/IDEA-Research/GroundingDINO>.
- Macenski, S., y Jambrecic, I. (2021). Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61), 2783. Descargado de <https://doi.org/10.21105/joss.02783> doi: 10.21105/joss.02783
- Matplotlib. (2025). *Introducción a matplotlib.* [https://www.geeksforgeeks.org/python/python-introduction-matplotlib/](https://www.geeksforgeeks.org/python-python-introduction-matplotlib/). Descargado de <https://www.geeksforgeeks.org/python/python-introduction-matplotlib/>
- Mecalux. (2024). *Sistema lidar: funcionamiento y aplicaciones en logística.* Descargado de <https://www.mecalux.es/blog/sistema-lidar>
- Oliphant, T. (2025). *Documentación numpy.* <https://numpy.org/>. Descargado de <https://numpy.org/>
- Open Robotics. (2022). *Ros 2 documentation: Humble hawksbill.* <https://docs.ros.org/en/humble/index.html>.
- Open Robotics. (2025). *rclpy (python client library for ros 2).* Descargado de <https://docs.ros.org/en/rolling/p/rclpy/>
- OpenCV. (2024). *Página oficial para descarga de opencv.* <https://opencv.org/>. Descargado de <https://opencv.org/>
- OpenWebinars. (2023). *Qué son los embeddings y para qué sirven.* Descargado de <https://openwebinars.net/blog/embeddings/>
- Python. (2023). *Python language reference, version 3.8.* <https://www.python.org>.
- Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016). You only look once: Unified, real-time object detection. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)* (pp. 779–788). doi: 10.1109/CVPR.2016.91
- Ricardo Urvina Córdova, E. A. T., y Álvaro Prado Romo. (2023). Localización simultánea y mapeo para control de un robot móvil autónomo usando escaneo de nube de puntos lidar y métodos de aprendizaje de máquina. *Ingeniare. Revista chilena de ingeniería*, 31(13). Descargado de https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-33052023001300155 doi: 10.4067/S0718-33052023001300155
- Robotics, U. (2025). *Robot unitree go2.* <https://www.unitree.com/go2>.

- Skalski, P. (2023). *Grounding dino: Sota zero-shot object detection.* <https://blog.roboflow.com/grounding-dino-zero-shot-object-detection/>.
- Tkinter. (2025). *Información general tkinter.* <https://keepcoding.io/blog/que-es-tkinter/>. Descargado de <https://keepcoding.io/blog/que-es-tkinter/>
- Ultralytics. (2025). *Yolov8 – the latest version of the you only look once object detector.* Descargado de <https://docs.ultralytics.com/es/models/yolov8/>
- VisionPlatform.ai. (2025). *Yolov8: Detección de objetos de última generación en reconocimiento de imágenes (computer vision).* Descargado de <https://visionplatform.ai/es/yolov8-deteccion-de-objetos-de-ultima-generacion-en-reconocimiento-de-imagenes-computer-vision/>

Lista de Acrónimos y Abreviaturas

2D	2 Dimensiones.
3D	3 Dimensiones.
AP	Access Point.
API	Application Programming Interface.
CNN	Redes Neuronales Convolucionales.
DDS	Data Distribution Service.
DTLS	Datagram Transport Layer Security.
EKF	Filtro de Kalman Extendido.
FPS	Fotogramas por Segundo.
GMM	Modelos de Mezcla Gaussianos.
GUI	Interfaz Gráfica de Usuario.
IA	Inteligencia Artificial.
IMU	Unidad de Medición Inercial.
LiDAR	Light Detection And Ranging.
LTS	Soporte a Largo Plazo.
MAP	Media de Precisión Promedio.
R-CNN	Redes Neuronales Convolucionales de Región.
RGB	Red, Green, Blue.
ROS	Robot Operating System.
ROS2	Robot Operating System 2.
RPN	Region Proposal Network.
RRT	Rapidly-exploring Random Tree.
SDK	Software Development Kit.
SLAM	Simultaneous Localization and Mapping.
S RTP	Secure Real-time Transport Protocol.
TFG	Trabajo Final de Grado.
TFM	Trabajo Final de Máster.
ViT	Vision Transformer.
WebRTC	Web Real-Time Communication.
YOLO	You Only Look Once.