UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"SISTEMA DE VISIÓN ARTIFICIAL CON APRENDIZAJE PROFUNDO EMBEBIDO EN PLATAFORMA ACELERADA PARA APLICACIONES INTELIGENTES"

Trabajo de titulación previo a la obtención del título en Ingeniero en Mecatrónica

Línea de investigación: Producción Industrial y tecnología sostenible

AUTOR:

José Julián Granja Trávez

DIRECTOR:

Dr. Carlos Xavier Rosero Chandi

Ibarra – Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	Granja Trávez José Julián

DATOS DE LA OBRA	
TÍTULO:	Sistema de Visión Artificial con Aprendizaje Profundo Embebido en Plataforma Acelerada para Aplicaciones Inteligentes
AUTOR (ES):	Granja Trávez José Julián
FECHA: DD/MM/AAAA	09/11/2026
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Mecatrónica
DIRECTOR /ASESOR:	PhD. Carlos Xavier Rosero Chandi MSc. Luz María Tobar Subía Contento

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 9 días del mes de noviembre de 2026

EL AUTOR:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
JOSE JULIAN GRANJA TRAVEZ

Nombre: José Julián Granja Trávez

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICAS
CARRERA DE INGENIERÍA EN MECATRÓNICA

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 09 de noviembre del 2026

PhD. Carlos Xavier Rosero Chandi

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del Trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

(f)……

PhD. Carlos Xavier Rosero Chandi
C.C.: 1002515821

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICAS
CARRERA DE INGENIERÍA EN MECATRÓNICA

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificado del trabajo de Integración Curricular "Sistema de Visión Artificial con Aprendizaje Profundo Embebido en Plataforma Acelerada para Aplicaciones Inteligentes" elaborado por José Julián Granja Trávez, previo a la obtención del título de INGENIERO EN MECATRÓNICA, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

(f)……

PhD. Carlos Xavier Rosero Chandi
C.C.: 1002515821

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
LUZ MARIA TOBAR SUBIA CONTENTO

(f)……

MSc. Luz María Tobar Subía Contento
C.I:1002444204

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS iii

ÍNDICE DE FIGURAS vi

ÍNDICE DE TABLAS viii

ÍNDICE DE ECUACIONES ix

AGRADECIMIENTO x

DEDICATORIA xi

RESUMEN xii

ABSTRACT xiii

CAPITULO I: INTRODUCCIÓN 1

1.1 Planteamiento del problema 1

1.2 Objetivos 2

1.2.1 General 2

1.2.2 Específicos 2

1.3 Alcance y delimitación 2

1.4 Justificación 3

CAPÍTULO II: MARCO TEÓRICO 5

2.1 Antecedentes 5

2.2 Bases Teóricas 7
2.2.1 Visión Artificial 7
2.2.2 Aprendizaje profundo embebido 10
2.2.3 Arquitectura ROS 2 12
2.2.4 Hardware empleado 13
2.2.5 Drivers y librerías 16
2.2.6 Modelos preentrenados y personalización de redes de detección 16
2.2.7 Controlador para la cámara Intel RealSense D415 17
2.2.8 Entrenamiento de modelos de IA para evitar colisiones 18
2.2.9 Gestión de decisiones en tiempo real y comunicación entre nodos 19
CAPÍTULO III: MARCO METODOLÓGICO 21
3.1 Enfoque y modelo de investigación 21
3.2 Diseño de la Investigación 22
3.2.1. Fase 1: Análisis del sistema de visión artificial y plataformas embebidas 22
3.2.2. Fase 2: Diseño del algoritmo de detección de objetos 23
3.2.3. Fase 3: Implementación y validación en tiempo real 23
CAPÍTULO IV: RESULTADOS 25
4.1 Características generales del sistema implementado 25
4.2 Construcción del conjunto de datos 26
4.2.1 Recopilación y evolución del conjunto 27

4.2.2 Etiquetado y ejemplos representativos 28
4.2.3 Catálogo final y exlusión de furniture 30
4.3 Entrenamiento del modelo personalizado 32
4.3.1 Configuración y ejecución mediante CPU 32
4.3.2 Métricas del modelo final 34
4.3.3 Inferencia y límites de interpretación 36
4.4 Migración de archivos y modelos a la Jetson Orin Nano 37
4.4.1 Organización del workspace y modelos 37
4.4.2 Diferencias entre desarrollo e inferencia embebida 38
4.4.3 Herramientas de software empleadas 39
4.5 Integración de la cámara Intel RealSense D415 en ROS 2 40
4.5.1 Adquisición RGB y profundidad 40
4.5.2 Consulta de profundidad y asociación con cajas 41
4.6 Fusión de modelos y lógica de navegación 42
4.6.1 Complementariedad entre modelo base y personalizado 42
4.6.2 Definición de zonas y prioridad de cercanía 43
4.6.3 Estados y umbrales 43
4.7 Ejecución del sistema en la Jetson Orin Nano 44
4.7.1 Ejecución y observación del nodo de visión 44
4.7.2 Integración funcional con puente_arduino 45

4.8 Revisión del ecosistema previo del robot 48
4.9 Estado final, pruebas y análisis de resultados 49
4.9.1 Organización y alcance de la evidencia experimental 50
4.9.2 Distancia real frente a profundidad RealSense 50
4.9.3 Error de distancia y comportamiento de las mediciones 53
4.9.4 Pruebas por zonas y respuesta física 56
4.9.5 Transiciones entre estados 57
4.9.6 Limitación física y operativa entre 300 y 400 mm. 60
4.9.7 Frecuencias de publicación y capacidad de operación 61
4.9.8 Memoria, GPU, temperaturas y potencia 62
4.9.9 Evaluación de errores 65
CAPITULO V: CONCLUSIONES Y RECOMENDACIONES 68
5.1 Conclusiones 68
5.2 Recomendaciones 69
5.3 Trabajo a futuro 69
Referencias 70
ANEXOS 74

Índice de figuras

Fig. 2.1 Diagrama de flujo de visión artificial 8

Fig. 2.2 Comparativa entre una imagen RGB y mapa de profundidad..... 9

Fig. 2.3 Gráfico de nodos y tópicos en ROS 2. 13

Fig. 2.4 Cámara Intel RealSense D415..... 14

Fig. 2.5 Jetson Orin Nano Developer Kit 8Gb 15

Fig. 4.1 Arquitectura funcional del sistema de visión y su aplicación asistida al robot. 26

Fig. 4.2 Carga inicial de imágenes en Roboflow durante la construcción del dataset..... 27

Fig. 4.3 Intefaz de anotación de Roboflow y catálogo histórico de clases. 28

Fig. 4.4 Anotación de door y wall en una escena de puerta y pared..... 29

Fig. 4.5 Anotación de ventanas y muros con diferentes tamaños aparentes..... 30

Fig. 4.6 Registro de las cuatro clases antes de la exclusión de furniture..... 31

Fig. 4.7 Registro del entrenamiento en las épocas 4 y 5..... 33

Fig 4.8 Registro del entrenamiento en las épocas 14 y 15..... 33

Fig. 4.9 Registro de cierre del entrenamiento y métricas. 35

Fig. 4.10 Detecciones del modelo base en una escena interior..... 37

Fig. 4.11 Detección general con puntos de centro y distancias en milímetros. 40

Fig. 4.12 Flujo de detección, profundidad y decisión del sistema integrado..... 42

Fig. 4.13 Registro de distancias, zonas y estados en la Jetson 45

Fig. 4.14 Ejecución conjunta de cámara, visión y puente en la validación final. 46

Fig. 4.15 Visualización de la ejecución *puente_arduino* 47

Fig. 4.16 Visualización de la ejecución lector_camara_fusion..... 48

Fig. 4.17 Toma de datos realizada dentro del entorno de trabajo. 50

Fig. 4.18 Distancia real y valores de profundidad registrados..... 52

Fig. 4.19 Error absoluto de cada valor de distancia anotado. 55

Fig. 4.20 Desviaciones de las transiciones de aproximación..... 60

Fig. 4.21 Intervalos registrados de frecuencia RGB, profundidad y /estado_vision 62

Fig. 4.22 Uso de memoria RAM..... 64

Fig. 4.23 Rangos de temperatura y potencia en la ejecución..... 65

Fig. 4.24 Matriz de confusión del modelo personalizado..... 67

Índice de tablas

Tabla 4.1 Componentes y funciones de la aplicación final..... 26

Tabla 4.2 Componentes y funciones de la aplicación final..... 31

Tabla 4.3 Configuración del entrenamiento..... 32

Tabla 4.4 Métricas del modelo final. 34

Tabla 4.5 Organización de los archivos 38

Tabla 4.6 Herramientas de software utilizadas 39

Tabla 4.7 Reglas de profundidad y comandos. 43

Tabla 4.8 Estado y señal enviada hacia Arduino 46

Tabla 4.9 Ejemplos de visualización..... 47

Tabla 4.10 Pruebas de distancia con valores registrados. 51

Tabla 4.11 Pruebas con estados y aciertos. 51

Tabla 4.12 Errores obtenido a partir de las lecturas de profundidad. 53

Tabla 4.13 Pruebas a 600 mm con ubicación y estados. 56

Tabla 4.14 Comando esperado y respuesta física. 56

Tabla 4.15 Transiciones registradas en las pruebas y distancias 58

Tabla 4.16 Estados y resultado de la lógica 58

Tabla 4.17 Frecuencias registradas durante la ejecución del sistema 61

Tabla 4.18 Recursos y condiciones de operación registrados 63

Tabla 4.19 Evaluación de reconocimiento bajo diferentes condiciones de iluminación 66

Índice de ecuaciones

Ecuación 4.1 Coordenada horizontal del centro de la caja delimitadora 41

Ecuación 4.2 Coordenada vertical del centro de la caja delimitadora 41

Ecuación 4.3 Error 53

Ecuación 4.4 Error absoluto 53

Ecuación 4.5 Error porcentual 53

AGRADECIMIENTO

A mi Dios, por brindarme la fortaleza necesaria cuando más lo necesitaba en medio de una ciudad nueva para mí que se terminó convirtiendo en mi segundo hogar.

A la Universidad Técnica del Norte, noble institución que me acogió y ayudó a terminar de forjar en mí el carácter necesario para afrontar retos que pensaba eran insuperables. A través de su compromiso por la educación e investigación sembraron en mí la semilla de la búsqueda constante de la perfectibilidad personal y profesional.

A mi director, PhD. Xavier Rosero, quien más allá de su trabajo como docente destacado, también se encargó de mostrarme que en el mundo aún existen personas que les importa el bien colectivo, los principios y hacer bien las cosas sin importar contra quien debamos hacernos frente.

A mi asesora, Msc. Luz María Tobar, siempre presente con su consejo, guía y notable predisposición para la realización de este trabajo y también a lo largo de mi carrera universitaria.

A los docentes que conforman la carrera de Mecatrónica, quienes no solamente fueron excelentes profesionales sino también personas interesadas en el bienestar de sus estudiantes.

DEDICATORIA

El presente logro que se mantuvo esquivo para mí durante tanto tiempo se lo dedico de manera especial a las personas que más me sostuvieron en pie en su debido momento, soportando como los más robustos y fuertes pilares en mis momentos más oscuros; mis padres Mercedes y Fernando, sin su apoyo incondicional en todos los ámbitos posibles simplemente esto no se hubiera podido orquestar.

A mis hermanos, Rommel, Johanna y Fernando, que pese a estar en ciudades y países distintos, cada uno con sus respectivos problemas, nunca me dejaron sentirme solo, siempre supieron demostrarme de alguna u otra manera que su apoyo era incondicional, que su confianza en lo que podía lograr nunca vaciló en lo más mínimo y que el amor de hermanos no conoce tiempo ni fronteras, estas palabras también van hacia Christian, más que mi cuñado es un hermano más para mí.

A todos mis sobrinos, que no tienen idea de la importancia su papel en mi vida, su capacidad para hacerme sonreír en los momentos más adversos y darme ese impulso constante de ser una mejor persona me dieron y me siguen dando años de vida.

A mi prometida Victoria, que me acompañó casi toda la universidad, siempre entendiéndome y ayudándome a hacer las cosas de la vida mucho más fáciles pese a la distancia, por su enorme paciencia y confianza depositada en mí, te agradeceré por toda la vida futura esposa.

A los amigos que hice en la universidad, Andrew G. Pedro H. Edwin M. y Erika F. que llegaron como compañeros, pero el tiempo los convirtió en personas extraordinarias tanto para mi vida académica como personal, compartiendo momentos de felicidad, adversidad y situaciones del día a día.

RESUMEN

En este trabajo se presenta el desarrollo de un sistema de visión artificial embebido con capacidad de inferencia en tiempo real, orientado a la detección de objetos y la prevención autónoma de colisiones en un robot móvil de alto torque. La arquitectura actual del robot, basada en una Jetson Nano y una cámara Kinect, resulta insuficiente para implementar algoritmos de aprendizaje profundo debido a sus limitaciones de procesamiento y falta de lógica de evasión. Para solucionar este problema, se propone la migración hacia una Jetson Orin Nano junto a una cámara de visión artificial Intel RealSense D415 y la integración de un nuevo sistema de visión compatible con modelos como YOLOv5, optimizados para plataformas aceleradas. El sistema se estructura sobre una arquitectura de comunicación entre

Arduino Mega, Raspberry Pi 5 con ROS 2 y la nueva plataforma Jetson Orin con la cámara RealSense, permitiendo la toma de decisiones autónomas a partir del análisis visual del entorno. Este proyecto busca aportar una solución eficiente y aplicable tanto en entornos educativos como en desarrollos de robótica inteligente.

Palabras clave: Visión Artificial, aprendizaje profundo, YOLOv5

ABSTRACT

This work presents the development of an embedded computer vision system with real-time inference capabilities, aimed at object detection and autonomous collision avoidance in a high-torque mobile robot. The robot's current architecture, based on a Jetson Nano and a Kinect camera, is insufficient for deploying deep learning algorithms due to processing limitations and the absence of avoidance logic. To address this issue, a migration is proposed toward a Jetson Orin Nano together with an Intel RealSense D415 vision camera, and the integration of a new vision system compatible with models such as YOLOv5, optimized for accelerated platforms. The system is built on a communication architecture involving an Arduino Mega, a Raspberry Pi 5 running ROS 2, and the new Jetson platform with the RealSense camera, enabling autonomous decision-making based on real-time visual environment analysis. This project aims to provide an efficient and applicable solution for both educational settings and intelligent robotics development.

Keywords: Computer Vision, Deep Learning, YOLOv5

CAPITULO I: INTRODUCCIÓN

1.1 Planteamiento del problema

La Universidad Técnica del Norte dispone de un robot móvil de alto torque desarrollado por exalumnos, quienes dejaron su diseño abierto para futuras mejoras tecnológicas y funcionales. Uno de los aspectos identificados para optimización es el sistema de visión artificial, el cual actualmente proporciona imágenes en tiempo real del entorno, pero carece de capacidad para influir en la toma de decisiones del robot. Esta limitación impide que el sistema reaccione ante obstáculos, lo que compromete la seguridad operativa, al no poder ejecutar maniobras de evasión ni detenerse automáticamente frente a una posible colisión [1,2].

El sistema actual presenta además deficiencias técnicas que afectan su desempeño. En particular, la comunicación entre la plataforma embebida Jetson Nano Developer Kit y la minicomputadora Raspberry Pi 5 es ineficiente debido a conflictos en la deserialización de datos, incompatibilidades de hardware y a la limitada capacidad de memoria RAM (2 GB) de la Jetson Nano. Estas condiciones impiden el desarrollo e implementación de algoritmos de visión más avanzados, afectando directamente el rendimiento del sistema y su capacidad de respuesta en tiempo real.

Ante este diagnóstico, se reconoce la necesidad de mejorar la arquitectura de cómputo y procesamiento del sistema de visión artificial del robot. Como solución, se propone sustituir la Jetson Nano por una Jetson Orin Nano, que cuenta con 8 GB de RAM y capacidades superiores de procesamiento, lo cual permitirá la implementación de algoritmos de aprendizaje profundo orientados al reconocimiento de objetos y toma de decisiones autónomas. Esta mejora no solo eliminará los problemas de comunicación actuales, sino que permitirá dotar al robot de mayor

inteligencia situacional, incrementando su robustez operativa y abriendo la posibilidad de nuevas aplicaciones inteligentes en navegación, detección y respuesta frente al entorno.

1.2 Objetivos

1.2.1 General

Desarrollar un sistema de visión artificial embebido basado en aprendizaje profundo para detección de objetos y análisis del entorno en tiempo real en un robot móvil de alto torque.

1.2.2 Específicos

- Analizar el sistema de visión artificial actual del robot móvil y las plataformas embebidas utilizadas en aplicaciones educativas.
- Diseñar un algoritmo de detección de objetos orientado a la prevención de colisiones.
- Implementar el sistema de visión embebido con capacidad de inferencia en tiempo real en la arquitectura del robot.

1.3 Alcance y delimitación

El presente proyecto comprende el diseño, implementación e integración de un sistema de visión artificial embebido basado en aprendizaje profundo, destinado a un robot móvil de alto torque previamente desarrollado. El nuevo sistema empleará una unidad de procesamiento con capacidad extendida de memoria y aceleración para inteligencia artificial, reemplazando la plataforma anterior, cuya limitada capacidad impedía ejecutar tareas de detección compleja en tiempo real y generaba problemas de compatibilidad con el sistema de control.

El sistema integrará una cámara digital compatible con procesamiento de imágenes embebido y sensores auxiliares para la detección del entorno, permitiendo adquirir información visual en tiempo real. Se desarrollará un módulo de detección de objetos basado en redes neuronales convolucionales, capaz de identificar obstáculos y generar respuestas preventivas, como detener o redirigir la trayectoria del robot para evitar colisiones.

Se establecerá una arquitectura funcional donde la unidad de procesamiento embebida actúe como nodo principal de visión e inferencia, comunicándose con el sistema de control encargado de ejecutar acciones de navegación. La lógica de control será optimizada para garantizar baja latencia y alta confiabilidad en la toma de decisiones autónomas.

La validación del sistema se realizará mediante pruebas experimentales en entornos reales, midiendo indicadores clave como precisión de detección, tiempo de respuesta, estabilidad del sistema y capacidad de evitar colisiones en distintos escenarios operativos.

1.4 Justificación

La presente investigación resulta de gran importancia para la formación del candidato a ingeniero en mecatrónica, al integrar conocimientos avanzados en visión artificial, sistemas embebidos y aprendizaje profundo, aplicados a una solución funcional y validada. Este proceso refuerza competencias clave en robótica, control y programación, consolidando la capacidad de desarrollar sistemas autónomos complejos.

Desde el ámbito educativo, el proyecto constituye un caso de estudio replicable, al documentar desde la selección de hardware hasta el diseño de algoritmos de deep learning e integración de

interfaces. Su implementación permitirá a futuros estudiantes comprender de forma estructurada el funcionamiento de sistemas inteligentes y su aplicación en entornos reales.

A nivel social e ingenieril, dotar al robot de capacidad para detectar y anticipar colisiones mejora sustancialmente su seguridad operativa, reduciendo riesgos tanto para el entorno como para el propio sistema. Esta capacidad fortalece la confianza en el uso de robots móviles en espacios compartidos, promoviendo la adopción segura de tecnologías con inteligencia artificial en contextos académicos, industriales y colaborativos.

CAPÍTULO II: MARCO TEÓRICO

2.1 Antecedentes

La integración de visión e inteligencia artificial en sistemas embebidos representa una tendencia creciente en el desarrollo de robots móviles autónomos. Diversas investigaciones demuestran la viabilidad de incorporar algoritmos de detección y decisión sobre plataformas con recursos limitados, mejorando la capacidad de navegación y evitando colisiones en entornos dinámicos.

En [3], se presenta el desarrollo de un robot aspiradora que emplea visión por computadora e inteligencia artificial para identificar y esquivar obstáculos. El sistema se construye tras una revisión bibliográfica, selección de componentes, y desarrollo de algoritmos de clasificación y segmentación de imágenes. La detección se realiza en tiempo real, y las decisiones de movimiento se basan en la interpretación visual del entorno, optimizando la autonomía del robot. Este enfoque demuestra la importancia de la integración coherente entre hardware, software de IA y mecánica del sistema.

Otra línea de trabajo relevante se enfoca en el reconocimiento de marcas viales dentro de plantas de manufactura. En [4], se implementa un sistema que identifica líneas pintadas en el suelo mediante visión artificial y redes neuronales profundas, utilizando OpenCV y YOLOv5 sobre Python. El sistema demuestra una precisión superior al 85% bajo condiciones óptimas de iluminación, validando la factibilidad de aplicar deep learning en hardware embebido. Esta línea apoya la viabilidad de ejecutar modelos avanzados en plataformas con restricciones computacionales.

En [5], se desarrolla un robot autónomo de fútbol basado en visión artificial y bajo costo, utilizando una plataforma embebida sencilla para detectar en tiempo real elementos clave como la

pelota. Aunque funcional, el sistema evidencia limitaciones relacionadas con el rendimiento computacional y la capacidad de procesamiento, lo cual justifica el uso de plataformas más avanzadas en proyectos que requieran mayor robustez y complejidad en sus decisiones visuales.

Una tesis presentada en [6] diseña un sistema de guiado autónomo basado en visión por computadora sobre una plataforma embebida con Raspberry Pi. El sistema realiza corrección de perspectiva, detección de bordes y cálculo de ángulo de orientación para mantener el seguimiento de trayectorias marcadas en el suelo. Aunque logra una navegación fluida, depende de técnicas tradicionales que pueden verse superadas por enfoques modernos basados en deep learning.

En [7] se implementa un robot móvil multitarea que integra visión artificial, posicionamiento y telemetría. La arquitectura combina una plataforma embebida para visión y una microcontroladora para el control de movimiento, comunicadas de manera bidireccional. El sistema ejecuta tareas simultáneas como navegación, visualización gráfica de datos y retorno automático a zona de carga, demostrando la factibilidad de combinar múltiples funcionalidades en una única plataforma robótica, base sobre la cual se puede incorporar aprendizaje profundo para mejorar la toma de decisiones.

La evolución de los sistemas de visión artificial en robótica móvil ha llevado al desarrollo de soluciones optimizadas para hardware de bajo consumo, permitiendo la ejecución de tareas complejas como la detección de objetos en tiempo real y la navegación autónoma en entornos dinámicos. En [8], se introduce xYOLO, un modelo de detección diseñado específicamente para operar en robots humanoides de fútbol sobre hardware de bajo rendimiento, demostrando que es posible mantener la precisión sin comprometer la eficiencia computacional. Complementariamente, el modelo Fostc3net, propuesto en [9], presenta una versión ligera de YOLOv5 con mejoras

estructurales que permiten una ejecución más ágil en plataformas embebidas. Además, en [10] se desarrolla un sistema de control visual basado en redes neuronales convolucionales combinado con controladores PID, lo que refuerza la aplicabilidad del aprendizaje profundo en la toma de decisiones de robots móviles, incluso con recursos limitados, alineándose con los objetivos de la presente investigación.

2.2 Bases Teóricas

2.2.1 Visión Artificial

La visión artificial permite a un robot "ver" y a la vez interpretar su entorno mediante imágenes digitales, transformando los píxeles en información útil, no solamente realiza el proceso de capturar una imagen, sino que la procesa, entiende y toma decisiones autónomas basadas en lo que ve. Es la base medular de por ejemplo un robot orientado a identificar obstáculos, calcular distancias y responder en milisegundos [11].

Este campo ha evolucionado exponencialmente en los últimos años debido al acceso a sensores asequibles como las cámaras RGB-D y gracias al desarrollo de algoritmos inteligentes algunos open sourse, teniendo una estructura alineada como en la Fig. 2.1. Plataformas embebidas como la Jetson Orin hacen posible que la visión artificial se ejecute en el propio robot, sin depender de servidores remotos o procesadores adicionales.

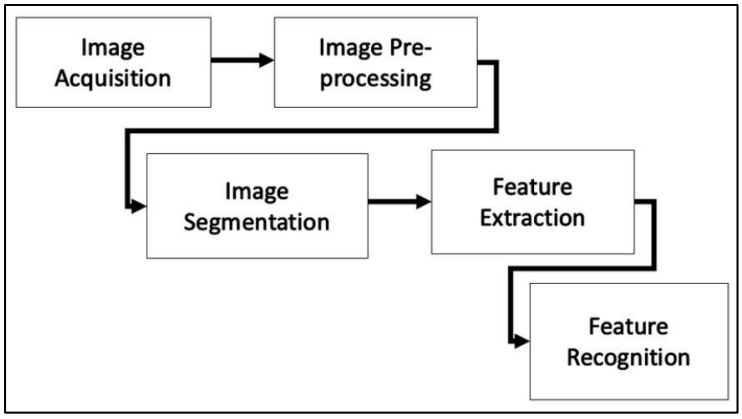


Fig. 2.1 Diagrama de flujo de visión artificial

2.2.1.1 Procesamiento de imágenes y extracción de características

Es la base donde comienza toda interpretación visual, en este proceso se incluye filtrado para eliminar ruido (filtros espaciales, medianos, gaussianos), también corrección geométrica (calibración intrínseca, corrección de distorsión radial/tangencial), y transformaciones espaciales como escalado y segmentación. Cada técnica mejora la calidad y procesamiento de la imagen para que los algoritmos posteriores trabajen con datos más confiables.

La extracción de características constituye el análisis más profundo; métodos como Harris (para esquinas), Canny (para bordes) y SIFT (para puntos clave fijos a escala e iluminación) permiten representar regiones críticas del entorno. Estas características son fundamentales para crear puntos clave, profundos y concisos que sobrevivan a cambios en la perspectiva o condiciones de luz [11].

El tiempo que lleva el procesamiento es necesario evaluarlo sobre la correcta ejecución del mismo, los métodos clásicos anteriormente mencionados hacen parten de las bases de procesamiento de imágenes.

2.2.1.2 Cámaras de profundidad (visión estéreo activa)

La visión estéreo activa como la que usa en cámaras dedicadas a la visión artificial, combina los patrones proyectados con dos sensores infrarrojos para calcular la disparidad entre estos. Esa disparidad se desarrolla y transforma en un mapa de profundidad que representa la distancia de cada píxel a la cámara [12].

La ventaja que nos brinda esta tecnología tiene como base su capacidad para operar en condiciones de iluminación adversas mientras proporciona precisión milimétrica en rangos de 0.3 a 10 m haciendo posible reconocer obstáculos tridimensionales en tiempo real [12].

La representación 3D ilustrada en la Fig. 2.2 resulta esencial en la arquitectura pues el robot no ve solo píxeles, sino puntos en el espacio con coordenadas reales, lo cual permite cálculos robustos de trayectoria, detección de obstáculos o evasión antes de una posible colisión.

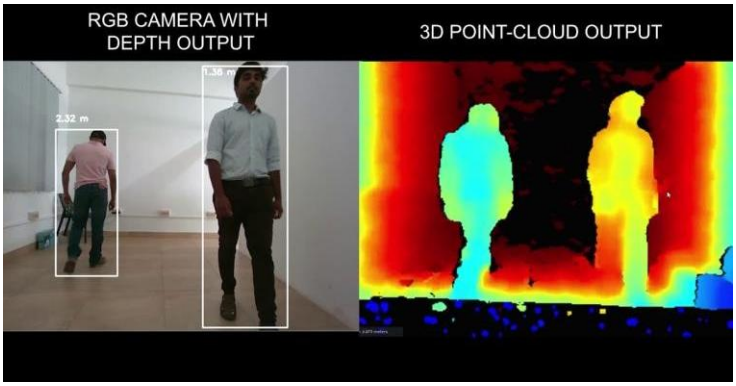


Fig. 2.2 Comparativa entre una imagen RGB y mapa de profundidad

2.2.2 Aprendizaje profundo embebido

El aprendizaje profundo embebido tiene como objetivo ejecutar modelos neuronales directamente en dispositivos con recursos limitados, el principal desafío está en lograr un equilibrio entre precisión y eficiencia energética, lo cual se consigue con técnicas como la cuantización, el pruning o la elección de arquitecturas eficientes [13]. Estos recursos permiten desplegar redes profundas en microcontroladores, GPUs móviles o plataformas integradas, sin depender de servidores externos.

Adicionalmente este enfoque trae consigo ventajas claras por ejemplo: menor latencia, mayor privacidad de los datos y una reducción significativa en el costo operativo [14]. Por lo general el flujo de trabajo usual consiste en entrenar el modelo en una estación potente, luego optimizarlo y finalmente trasladarlo al hardware embebido que ejecutará la inferencia en tiempo real.

También se ha demostrado que la cuantización de datos puede adaptarse a dispositivos con memoria muy limitada como microcontroladores de bajo consumo, manteniendo niveles de precisión aceptables y razonables [15]. Este tipo de despliegue resulta esencial para aplicaciones donde la latencia es crítica o el consumo de energía debe mantenerse al mínimo.

2.2.2.1 Fundamentos de las CNN (convolutional neural networks)

Las capas convolucionales permiten extraer patrones locales de una imagen aplicando filtros, mientras que las funciones de activación como Rectified Linear Unit (ReLu) introducen la no linealidad necesaria para modelar relaciones más completas y a la vez más complejas [14]. Después aparece el pooling que es el encargado de reducir la resolución espacial, lo que ayuda a controlar el número de parámetros para que este no se vuelva en extremo pesado y mejora la generalización.

Cuando se quiere implementar esto en un entorno embebido el reto medular está en mantener el rendimiento sin comprometer los recursos, para ello se aplican técnicas de Pruning (eliminación de datos descartables), reducción de precisión (como el uso de enteros en lugar de flotantes) y el diseño de arquitecturas ligeras, que permiten ejecutar CNNs en plataformas con restricciones serias [13].

La cuantización es un ejemplo que ha demostrado ser efectiva para ejecutar redes ya entrenadas en microcontroladores con núcleos Cortex-M o incluso en GPUs embebidas de bajo consumo [15]. Estas técnicas no solo reducen el tamaño del modelo sino que a la vez aceleran la inferencia y bajan el consumo energético.

2.2.2.2 YOLOv8 y detección de objetos en una etapa

Un detector de objetos produce categorías, etiquetas y localizaciones, son usualmente señaladas mediante el uso de cajas delimitadoras, tienen su distinción y particularidad de que a diferencia de una clasificación global de imagen en esta varias instancias o clases pueden coexistir dentro del mismo cuadro. You Only Look Once (*YOLO*) propone un método de detección en una sola etapa donde la imagen es procesada por la red para obtener las clases, cajas delimitadoras y niveles de confianza en cada detección cuyo enfoque permite reducir de manera considerable la cantidad de etapas necesarias durante el procesamiento y lo hace particularmente útil en aplicaciones de monitoreo contínuo.

YOLOv8 desarrollado por Ultralytics es miembro de esta familia de detectores y tiene disponibilidad en diferentes variantes permitiendo adaptar el modelo según la capacidad y necesidad de procesamiento disponible así como su aplicación concreta, entre sus características destaca su arquitectura orientada a mejorar la detección y localización de objetos pero manteniendo

una armonía entre precisión y velocidad, además al ser una herramienta de código abierto y respaldado por una comunidad activa, esta puede ser utilizada tanto con modelos previamente entrenados como puntos de partida para desarrollar modelos personalizados con un propio dataset [24].

La caja delimitadora que esta utiliza para demarcar los objetos representa aproximadamente la extensión visible de los mismos y puede definirse mediante coordenadas de sus extremos o también a través de su centro, ancho y alto, adicionalmente a esta información se añade un nivel de confianza que permite estimar qué tan segura es la predicción señalada por el detector.

2.2.3 Arquitectura ROS 2

ROS 2 (Robot Operating System 2) es una plataforma de software diseñada para ayudar de manera considerable a facilitar el desarrollo de sistemas robóticos distribuidos y confiables, a diferencia de ROS 1 su arquitectura fue reconstruida desde cero para adaptarse a entornos donde se requiere comunicación en tiempo real, seguridad y proyección a una mayor escalabilidad.

Una de sus características centrales es el uso de DDS (Data Distribution Service) como sistema de comunicación el cual permite conectar nodos en distintos dispositivos Fig. 2.3 gestionando el envío y recepción de datos de forma eficiente. En esta arquitectura los nodos se comunican mediante publicadores y suscriptores, además de servicios y acciones, lo que permite estructurar el sistema en módulos independientes y reutilizables [17].

ROS 2 también tiene la capacidad de admitir sistemas operativos en tiempo real y múltiples lenguajes de programación, como Python y C++, ampliando de manera considerable sus

aplicaciones tanto en el ámbito industrial como en el educativo [16]. Herramientas como RViz y rosbag complementan su funcionalidad permitiendo la visualización y el registro de datos experimentales [17].

Básicamente ROS 2 proporciona una base sólida para el desarrollo de robots inteligentes y autónomos, gracias a su arquitectura abierta, su comunidad activa y su enfoque en el rendimiento distribuido.

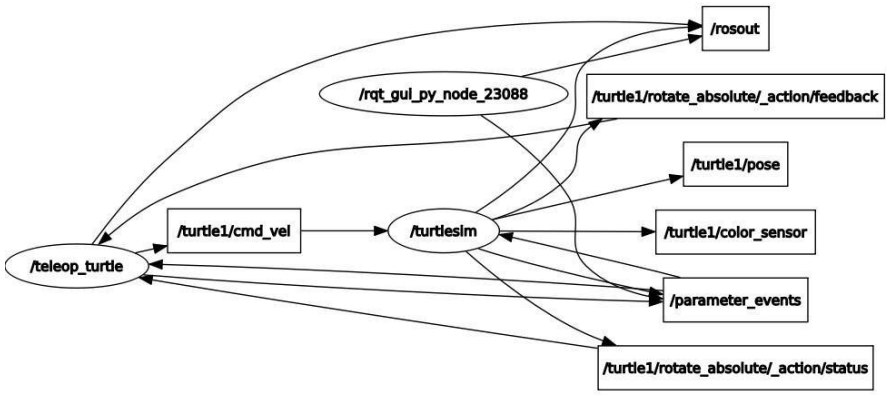


Fig. 2.3 Gráfico de nodos y tópicos en ROS 2.

2.2.4 Hardware empleado

2.2.4.1 Cámara Intel RealSense D415

La cámara Intel RealSense D415 es un sensor de profundidad especializado que combina una cámara RGB con una cámara infrarroja estéreo, permitiendo la percepción tridimensional del entorno. Utiliza una técnica de visión estéreo activa asistida por proyector infrarrojo, herramienta la cual permite ofrecer alta precisión en la obtención de datos de profundidad incluso en condiciones

de iluminación variable. Esta tecnología permite que el robot capte en tiempo real imágenes con datos espaciales útiles para la navegación y la detección de obstáculos [18].



Fig. 2.4 Cámara Intel RealSense D415

Dentro de sus principales características técnicas se incluye una resolución de hasta 1280×720 píxeles para la imagen RGB, una tasa de cuadros de hasta 90 FPS, y un rango de detección de profundidad de 0.3 m hasta 10 m, dependiendo del entorno y configuración. Su estrecho campo de visión (FOV) proporciona mayor precisión para tareas de visión enfocada en robótica móvil, adicionalmente es compatible con la plataforma ROS 2 mediante el paquete *realsense2_camera* lo cual facilita su integración con sistemas embebidos modernos como la Jetson Orin Nano[18].

Gracias a su capacidad de generar nubes de puntos en tiempo real y su soporte activo por parte de Intel y la comunidad de código abierto, se convierte en una herramienta adecuada para proyectos de navegación asistida en tiempo real con aprendizaje profundo.

2.2.4.2 Jetson Orin Nano

La Jetson Orin Nano Developer Kit, desarrollada por NVIDIA es una plataforma de cómputo embebido de alto rendimiento diseñada para ejecutar modelos de inteligencia artificial, a diferencia

de su antecesora Jetson Nano, esta versión cuenta con 8 GB de memoria RAM LPDDR5 [19], lo que la hace adecuada para tareas de inferencia en tiempo real basadas en redes neuronales convolucionales, como las implementadas en detección de objetos y prevención de colisiones en robótica móvil.

Este developer kit integra una GPU NVIDIA Ampere con 1024 núcleos CUDA y 32 núcleos Tensor, un procesador ARM Cortex-A78AE de 6 núcleos, y ofrece compatibilidad con múltiples interfaces de entrada/salida como USB 3.2, M.2 Key E, y cámara MIPI CSI. El rendimiento de esta tarjeta permite procesar simultáneamente datos de sensores RGB-D [19], generar nubes de puntos, ejecutar inferencias y establecer comunicación directa con otros módulos del sistema robótico, todo en tiempo real.



Fig. 2.5 Jetson Orin Nano Developer Kit 8Gb

Además, es compatible con el entorno JetPack 6 de NVIDIA, el cual incluye soporte nativo para bibliotecas de aprendizaje profundo como TensorRT, cuDNN y OpenCV, optimizando así el rendimiento del sistema de visión artificial embebido. Esta plataforma constituye el núcleo del

nuevo sistema propuesto para la prevención de colisiones, gracias a su balance entre bajo consumo energético y alta capacidad de cómputo.

2.2.5 Drivers y librerías

Un driver es un software que sirve como un intermediario entre el sistema operativo y el hardware, este permite que la plataforma de cómputo embebida reconozca y controle un periférico de manera óptima, sin drivers el sistema no podría interpretar las señales del dispositivo ni interactuar con él de forma eficiente, mucho menos en tiempo real.

Una librería es un conjunto de funciones y herramientas previamente realizadas o recopiladas que facilitan el desarrollo de software. Proporciona métodos optimizados sean de desarrollador u open source dirigidos para realizar tareas específicas y especializadas, como el procesamiento de imágenes y/o la comunicación con dispositivos, aunque tienden a cumplir funciones distintas, ambos son esenciales en la ejecución completa. El driver garantiza que el hardware sea accesible para el sistema y por otro lado la librería simplifica la implementación de algoritmos y la gestión de datos, asegurando un rendimiento eficiente y una integración armoniosa adecuada en aplicaciones avanzadas.

2.2.6 Modelos preentrenados y personalización de redes de detección

Existen modelos preentrenados que no son más que redes neuronales que han sido previamente ajustadas y personalizadas utilizando conjuntos de datos bastante robustos y que han aprendido a reconocer características visuales de manera general por ejemplo formas, bordes, texturas o patrones asociados a diferentes categorías de objetos. Este conocimiento previamente entrenado permite reutilizar su conocimiento previo y establecerlo como punto de partida para la creación de

nuevos modelos de visión personalizados, ahorrando de manera significativa la tarea de entrenar una red completamente desde cero.

En aplicaciones como la detección de objetos la personalización de un modelo se basa en adaptar una arquitectura entrenada anteriormente e integrarla a un conjunto de clases específico, para esto se emplea un nuevo dataset compuesto por imágenes o parámetros alineados con el objetivo de interés del nuevo modelo, durante este entrenamiento los parámetros de la red se ajustan para mejorar este concepto propio mientras que se mantiene el conocimiento visual adquirido durante el preentrenamiento del modelo base.

En detectores de una etapa como los pertenecientes a la familia YOLO, las estructuras previamente desarrolladas pueden utilizarse como fuente medular para el desarrollo de modelos orientados a categorías particulares, una vez finalizado el entrenamiento, el resultante de los modelos representan una versión especializada del detector capaz de generar cajas delimitadoras, clases y niveles de confianza para los nuevos conjuntos de datos, es decir, una misma arquitectura puede adaptarse a diferentes campos de aplicaciones sin la necesidad de modificar su estructura inicial.

2.2.7 Controlador para la cámara Intel RealSense D415

Para integrar de manera correcta la cámara Intel RealSense D415 en un entorno ROS 2 se emplea el controlador oficial *realsense2_camera*, desarrollado y mantenido por Intel. Este paquete permite el acceso completo a las funcionalidades de la cámara, incluyendo la publicación de imágenes RGB [21], mapas de profundidad, nubes de puntos y parámetros de calibración en múltiples tópicos de ROS 2.

El nodo *realsense2_camera* también brinda a los usuarios soporte para sincronización de datos, filtrado de profundidad, alineamiento de imágenes y configuración dinámica, su uso es aplicado ampliamente y compatible con dispositivos con plataforma NVIDIA lo que lo convierte en una herramienta robusta para aplicaciones en robótica móvil. La instalación se realiza a través de los repositorios oficiales de Intel o mediante rosdep.

2.2.8 Entrenamiento de modelos de IA para evitar colisiones

El entrenamiento de modelos de detección permite que una red neuronal aprenda y reconozca patrones visuales orientados a diferentes elementos que conforman una escena, en este proceso la red ajusta constantemente los parámetros a partir de un conjunto de ejemplos preparados con anterioridad en busca de identificar características similares en las nuevas imágenes [25]. Para realizar el entrenamiento es necesario disponer de imágenes que entren en la categoría de su entorno en el cual se va a trabajar, en modelos de detección no basta solamente con identificar la clase del objeto sino también establecer su ubicación dentro de la imagen usando anotaciones que permitan a la red tener una relación directa entre lo visual y su determinada posición [26]. La calidad y variedad influyen de manera significativa y directa sobre la capacidad del modelo en responder de manera satisfactoria o insuficiente ante los escenarios presentados y utilizados durante su entrenamiento [28].

2.2.8.1 Dataset, anotación y particiones

Un dataset utilizado para detección de objetos se encuentra conformado por un conjunto de imágenes y sus respectivas anotaciones o etiquetas, tienen una diferencia notoria con la clasificación pues en ella generalmente se asigna una categoría. En la detección se requiere además indicar la

región exacta de donde se encuentra cada objeto, para este propósito suelen emplearse cajas delimitadoras que relacionan una categoría con una posición específica dentro de la escena [26].

Dentro de la familia YOLO los objetos anotados se representan mediante un identificador de clase además de las coordenadas, una misma imagen es capaz de contener diferentes objetos pertenecientes a diferentes categorías. Las herramientas destinadas a la preparación de conjunto de datos como Roboflow brindan las herramientas para realizar procesos de anotación mediante las cajas delimitadoras, organizar imágenes y generad diferentes versiones de un conjunto antes de ser utilizadas para el entrenamiento [27].

Para organizar el proceso de aprendizaje los datos usualmente se dividen en categorías destinadas al entrenamiento, validación y prueba. En la primera se utiliza para ajustar parámetros dentro de la red, en la segunda se permite observar el comportamiento del modelo durante su etapa de desarrollo y finalmente la prueba permite analizar su desempeño y su respectivo ajuste de ser necesario [25] [26].

2.2.9 Gestión de decisiones en tiempo real y comunicación entre nodos

Una parte crítica del sistema propuesto es la toma de decisiones en tiempo real basada en la información visual recibida. Para ello, se empleará un nodo de lógica de evasión que suscriba los tópicos generados por el modelo de inferencia (como */detections* o */bounding_boxes*) y analice si existe un objeto en trayectoria de colisión [23].

Si se detecta una amenaza, el nodo envía comandos de detención o redirección al nodo de navegación. Esta comunicación se basa en mensajes estándar de ROS 2 (*geometry_msgs/Twist* para

movimiento y *std_msgs/Bool* para paros). El sistema debe ser validado en condiciones reales, evaluando latencia, precisión de detección y tasa de falsos positivos, asegurando que las decisiones se tomen antes del impacto.

CAPÍTULO III: MARCO METODOLÓGICO

3.1 Enfoque y modelo de investigación

En el marco del presente proyecto de integración curricular, se desarrolla una investigación de tipo aplicada con la finalidad de brindar una solución concreta al problema identificado. Este problema está relacionado con la creación de un sistema de visión por computadora destinado a un robot móvil con alto torque, cuya finalidad es dotarlo de percepción clara sobre el entorno en el que operará.

Para alcanzar este objetivo, se adopta una metodología de investigación documental, recurriendo a fuentes secundarias como textos especializados, publicaciones académicas y artículos científicos. Este método permite recolectar, seleccionar y organizar información clave que respalde tanto el desarrollo del estudio como la elaboración del informe técnico.

Paralelamente, se aplica un enfoque descriptivo que permite caracterizar las variables involucradas y analizar diferentes dimensiones del entorno relacionado con el sistema. La combinación de estos métodos posibilita avanzar con bases sólidas, al contar con datos relevantes y bien estructurados. Finalmente, el proceso culmina con una fase de investigación experimental, en la cual se realizan pruebas y ajustes al sistema. Esta etapa permite evaluar el comportamiento del sistema en condiciones controladas y observar cómo impacta en otras variables dependientes, mediante un análisis detallado de su desempeño.

3.2 Diseño de la Investigación

Para lograr los objetivos planteados, la investigación se divide en tres fases correlacionadas, cada una con actividades que validan y complementan el avance del proyecto.

3.2.1. Fase 1: Análisis del sistema de visión artificial y plataformas embebidas

Para esta fase se revisa a fondo el estado actual del robot y su entorno de cómputo, con el objetivo de sentar las bases de los desarrollos posteriores.

Actividad 1.1: Revisión técnica del hardware, software y capacidad de procesamiento

Se levantará un inventario de los componentes físicos (cámaras, sensores, unidad de cómputo) y del software (sistemas operativos, librerías, frameworks) que usa actualmente el robot.

Se evaluará su rendimiento y limitaciones.

Actividad 1.2: Estudio comparativo de arquitecturas embebidas y marcos de trabajo

Se investigarán otras plataformas de cómputo embebido (por ejemplo Jetson Nano/Orin,

Raspberry Pi, Coral Edge TPU) y frameworks de visión artificial (OpenCV, TensorFlow Lite, PyTorch Mobile). Se compararán en términos de velocidad de inferencia, consumo energético y facilidad de integración en entornos educativos.

3.2.2. Fase 2: Diseño del algoritmo de detección de objetos

Se procede a realizar la estructura lógica de inferencia para la identificación temprana de obstáculos.

Actividad 2.1: Desarrollo del flujo lógico y modelado de inferencia

Se definirá un diagrama de flujo que detalle cada paso del procesamiento de la imagen (preprocesado, segmentación, clasificación, post-procesado) y la arquitectura del modelo de inferencia.

Actividad 2.2: Selección y entrenamiento de red neuronal convolucional

Se elegirá una arquitectura adecuada y se entrenará con un dataset especializado en navegación autónoma de robots móviles. Se ajustarán hiperparámetros para maximizar precisión y velocidad.

3.2.3. Fase 3: Implementación y validación en tiempo real

En la etapa final se integran todos los componentes y se someten a pruebas en condiciones reales.

Actividad 3.1: Integración del modelo entrenado en la plataforma embebida

Se desplegará el modelo optimizado en la unidad de cómputo del robot. Se verificará su correcto enlace con los controladores de cámara y motores.

Actividad 3.2: Validación experimental de detección y maniobra evasiva

Se realizarán pruebas de campo donde el robot deba detectar obstáculos y modificar su trayectoria en tiempo real. Se registrarán métricas de latencia, tasa de aciertos y eficacia de las maniobras evasivas.

CAPÍTULO IV: RESULTADOS

4.1 Características generales del sistema implementado

El sistema final se implementa como una capa de percepción e intervención asistida sobre el robot móvil diferencial de alto torque. La cámara Intel RealSense D415 entrega imagen RGB y profundidad, por otro lado la Jetson Orin Nano ejecuta ROS 2 deduce y ejecuta la decisión evitando por completo el uso de la Raspberry Pi 5. El control normal del robot se mantiene manual mediante la APK/Bluetooth previamente diseñada, mientras la capa visual interviene ante el riesgo de colisión a través de la comunicación directa de la Jetson con el Arduino que se mantiene como el encargado de enviar los pulsos eléctricos para la ejecución o no de giro de las ruedas del robot. Los componentes anteriormente señalados se reflejan en la Tabla 4.1.

La aplicación desarrollada conserva la finalidad del trabajo y su correcta ejecución, el robot diferencial de alto torque proporciona la plataforma de validación permitiendo comprobar si la información visual se convierte en una intervención física.

El resultado principal refleja una cadena funcional Fig.4.1 con percepción, interpretación y actuación asistida, con procesamiento local en la plataforma embebida.

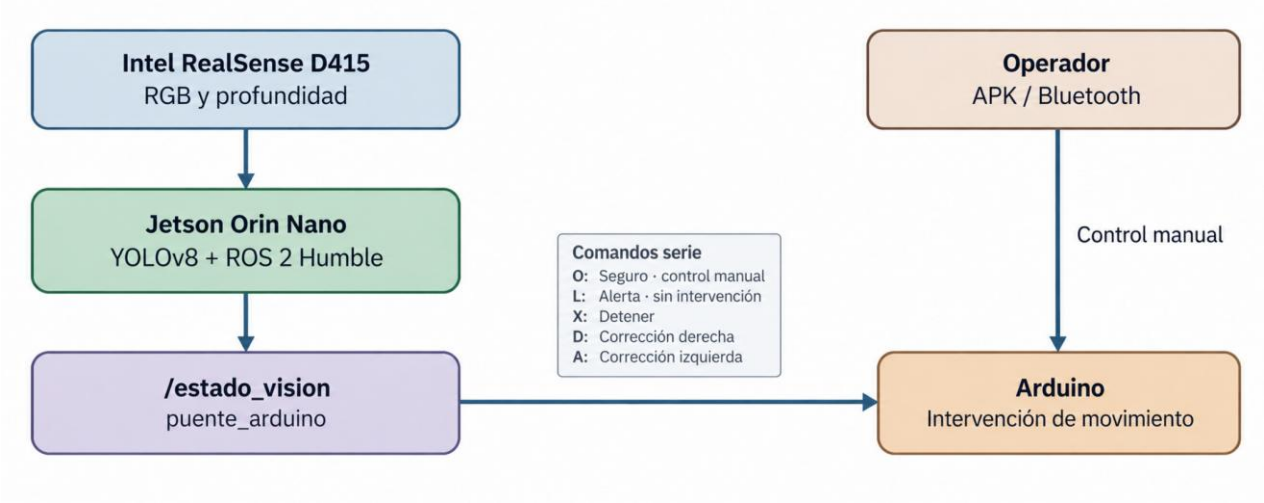


Fig. 4.1 Arquitectura funcional del sistema de visión y su aplicación asistida al robot.

Tabla 4.1 Componentes y funciones de la aplicación final.

Componente	Función
Intel RealSense D415	Adquisición RGB y profundidad
Jetson Orin Nano	Inferencia local y ejecución ROS 2
YOLOv8n.pt	Detección de clases generales
best_personalizado.pt	Detección de door, wall, furniture y window
/estado_visión	Comunicación del estado perceptivo
puente_arduino / Arduino	Traducción funcional e intervención
APK/Bluetooth	Conducción manual habitual

4.2 Construcción del conjunto de datos

El conjunto de datos se construye con escenas asociadas a un entorno de operación habitual no necesariamente un ambiente de laboratorio. La versión final contiene 301 imágenes, distribuidas así: 241 para entrenamiento, 40 para validación y 20 para prueba. Las clases finales utilizadas por el modelo personalizado son *door, wall* y *window*. La clase *furniture* es excluida del modelo final

por su comportamiento ambiguo durante la evolución del proyecto además que tendía a sobreponerse sobre etiquetas previamente entrenadas en el modelo base a partir del cual se realizó el nuevo modelo.

4.2.1 Recopilación y evolución del conjunto

Las imágenes disponibles Fig. 4.2 muestran elementos de un entorno cotidiano, muros, paredes, puertas, ventanas o estructuras en general, el conjunto se organiza para complementar las categorías generales del detector base con estructuras que intervienen en la interpretación del espacio, la selección de esas clases responde al entorno de aplicación general y a la necesidad de conservar la detección general previamente funcional.

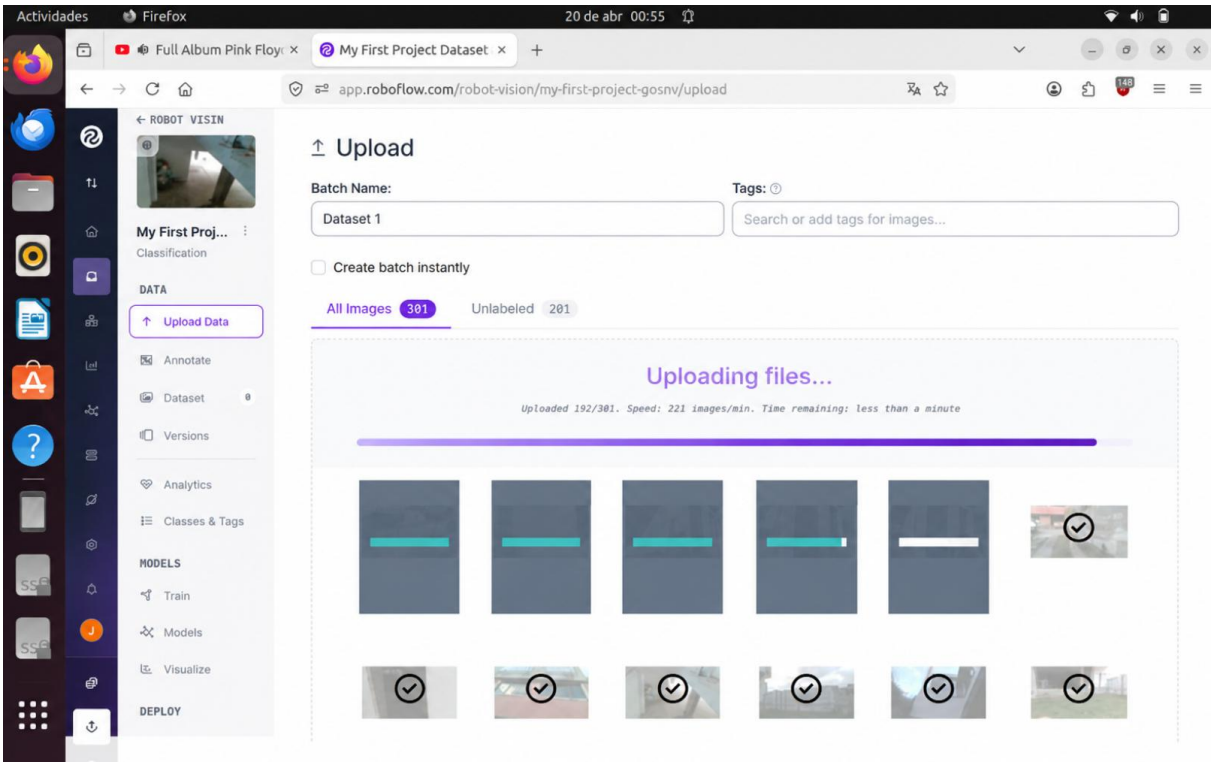


Fig. 4.2 Carga inicial de imágenes en Roboflow durante la construcción del dataset.

4.2.2 Etiquetado y ejemplos representativos

El etiquetado se realiza en Roboflow, asignando clases a regiones delimitadas, en la Fig. 4.3 conserva el panel de anotación y la presencia histórica de la clase *furniture*. El registro permite identificar la herramienta, imagen y los nombres de categorías durante esa etapa. La existencia de una etiqueta manual muestra la fase temprana del entrenamiento.



Fig. 4.3 Intefaz de anotación de Roboflow y catálogo histórico de clases.

La Fig. 4.4 presenta una puerta rodeada de pared, en el ejemplo muestra la diferencia entre un elemento con un contorno delimitado y una superficie que ocupa el fondo, en un detector basado en cajas; *wall* puede cubrir una parte considerable del cuadro, mientras *door* ocupa una región más concentrada, esta diferencia de escala influye en la localización puesto que el centro de una caja extensa no necesariamente representaría la parte del obstáculo más próxima al robot.

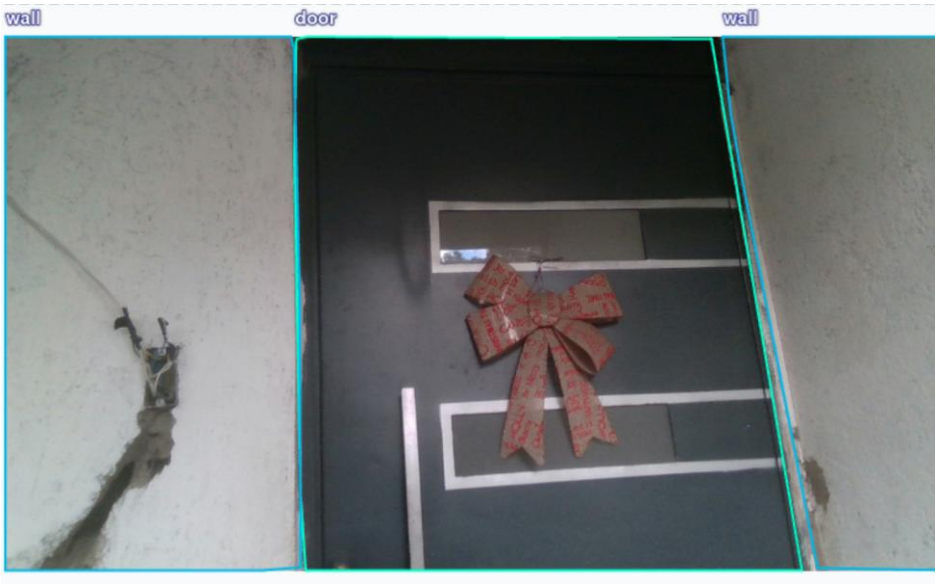


Fig. 4.4 Anotación de door y wall en una escena de puerta y pared.

La Fig. 4.5 muestra una escena exterior con varias ventanas y muros. Las ventanas pequeñas en el fondo y superficies cercanas se encuentran dentro de una misma imagen, este ejemplo aporta a la diversidad de escala pero también expone una gran pista a la aplicación. Así una detección lejana puede ser correcta pero no necesariamente ser un obstáculo que debe tomar un papel protagónico en algún tipo de maniobra entonces se tuvo variedad en ejemplos de profundidad y cercanía además de hacer hincapié en la clase.



Fig. 4.5 Anotación de ventanas y muros con diferentes tamaños aparentes.

4.2.3 Catálogo final y exlusión de furniture

El catálogo final contiene las clases *door, wall y window*. Las tres categorías describen estructuras del entorno no contempladas en el modelo YOLOv8 al ser difíciles de interpretar e incluso su ambigüedad poniendo como ejemplo una puerta de cristal por lo tanto se realizó esta recopilación complementando al modelo, por otro lado la exclusión de *furniture* se debió a la imprecisión y a la sobreposición que esta tomaba con clases de mobiliario ya reconocidas en el modelo base dejando de detectar como ejemplo a *chair*, *couch* u objetos similares.

La Fig. 4.6 refleja una escena que aún incluye *furniture*, evidenciando el cambio del catálogo inicial y su condición es meramente documental, la decisión final se toma como una delimitación del modelo personalizado mas no como una mejora de precisión por lo tanto esta característica propia del modelo no se ve afectada en ningún nivel.

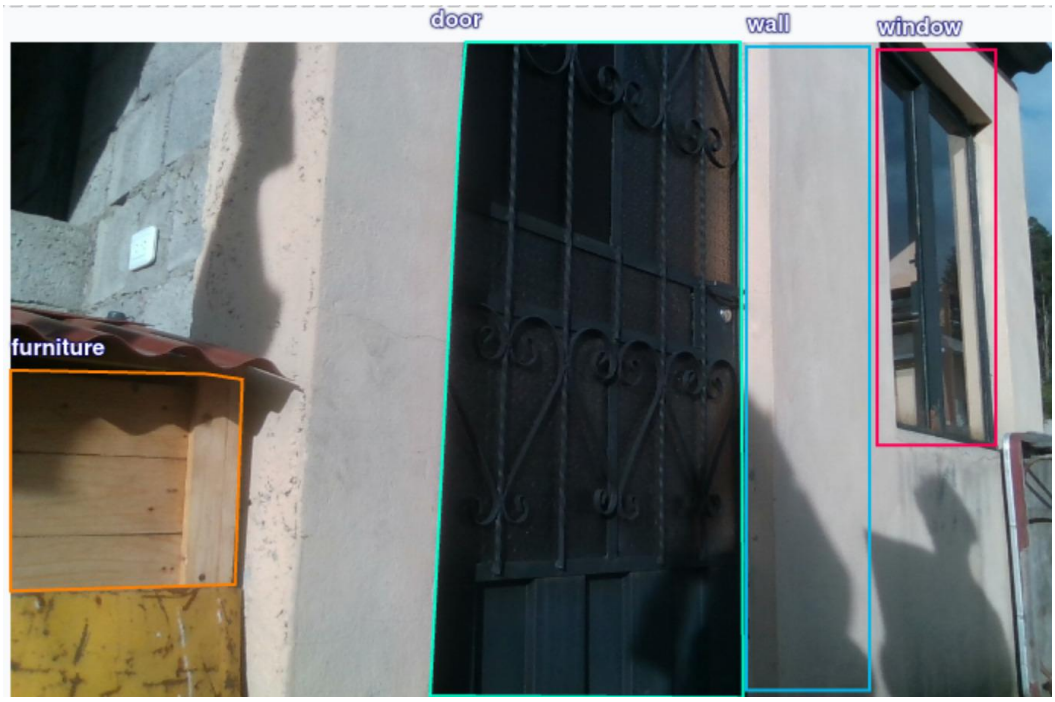


Fig. 4.6 Registro de las cuatro clases antes de la exclusión de furniture.

Roboflow permite organizar las imágenes para crear el modelo final personalizado. La separación del conjunto asigna imágenes al aprendizaje y reserva otras para prueba del detector como muestra la Tabla 4.2. Se presenta esta distribución como resultado de la preparación previa al entrenamiento.

Tabla 4.2 Componentes y funciones de la aplicación final.

Partición	Imágenes	Porcentaje derivado
Entrenamiento	241	80.07%
Validación	40	13.29%
Prueba	20	6.64%
Total	301	100%

4.3 Entrenamiento del modelo personalizado

4.3.1 Configuración y ejecución mediante CPU

El entrenamiento utiliza YOLOv8 mediante Ultralytics y se toma yolov8s.pt como punto de partida para el modelo personalizado, la ejecución completó 50 épocas con imgsz=640 (tamaño de imagen) y batch=16 (tamaño de lote), se realiza en CPU después de que se intenta utilizar la GPU originando: *CUDA Error 101: invalid divice ordinal*, sin embargo el fallo de la GPU no influye en la calidad del modelo entrenado solamente en el tiempo que este toma en realizarse aun así siendo relativamente bajo gracias a las especificaciones de la CPU.

Tabla 4.3 Configuración del entrenamiento.

Parámetro	Valor
Modelo inicial	yolov8s.pt
Épocas	50
imgsz	640 píxeles
batch	16
Dispositivo	CPU
Procesador	AMD Ryzen 7 260w/ Radeon 780M Graphics
Duración estimada	0.816 horas
Modelo desplegado	best_personalizado.pt
Clases finales	door, wall, window

En la Tabla 4.3 se distingue los parámetros de entrenamiento de los de adquisición, imgsz=640 corresponde al tamaño de entrada que maneja el detector, mientras que 640x480x30 corresponde al perfil de la cámara en la aplicación, pese a que pueden parecer similares tienen operaciones diferentes, el procesamiento del detector puede adaptar el cuadro a su entrada sin que eso cambie el perfil configurado del sensor.

La Fig. 4.7 registra épocas tempranas y memoria GPU indicada como 0G, mientras que la Fig. 4.8 muestra una etapa posterior al entrenamiento, ambas permiten evidenciar el avance de optimización y variación de pérdidas y métricas.

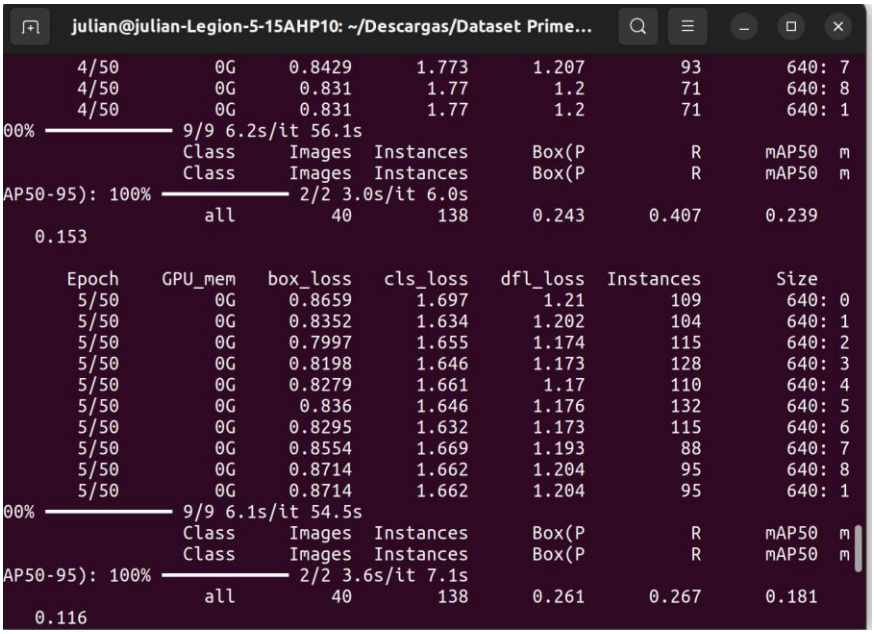


Fig. 4.7 Registro del entrenamiento en las épocas 4 y 5.

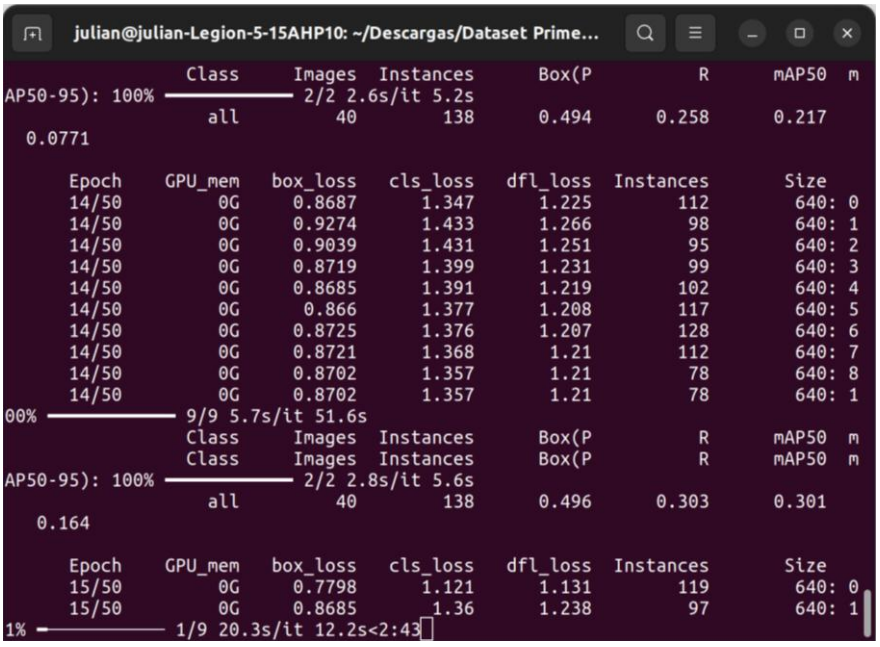


Fig 4.8 Registro del entrenamiento en las épocas 14 y 15.

El registro de cierre informa 0.816 horas y los archivos *best.pt* y *last.pt*. El nombre del modelo personalizado finalizado es *best_personalizado.pt.*

4.3.2 Métricas del modelo final

Las métricas esenciales disponibles en *best_personalizado.pt* son P=0.601, R=0.492, mAP50=0.495 y mAP50-95=0.389 explicados en la Tabla 4.4. Mientras en la Fig. 4.9 muestra los resultados obtenidos incluida la clase descartada *furniture*.

Tabla 4.4 Métricas del modelo final.

Métrica	Valor	Dominio
(P) Precisión	0.601	Validación del detector
(R) Sensibilidad	0.492	Validación del detector
(mAP50) Precisión media promedio	0.495	IoU 0.50
(mAP50-95) Precisión media promedio	0.389	IoU 0.50-0.95

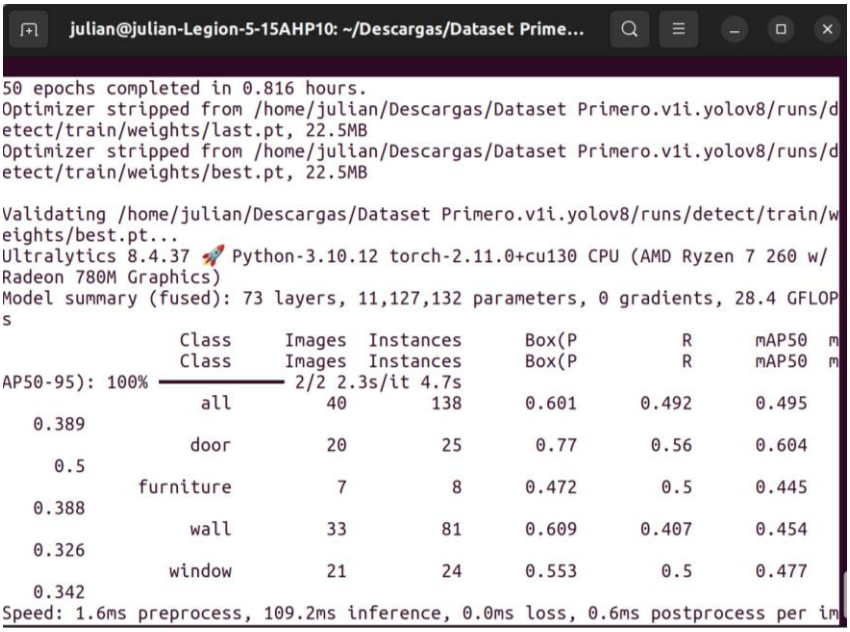


Fig. 4.9 Registro de cierre del entrenamiento y métricas.

La precisión de 0.601 indica un desempeño moderado en el control de falsas detecciones bajo el procedimiento de validación utilizado, no equivale a una probabilidad universal de que cada etiqueta emitida sea correcta, sino que su interpretación depende del conjunto evaluado y de los criterios de correspondencia entre cajas. En la aplicación asistida las falsas detecciones pueden generar intervenciones innecesarias, por lo que una precisión de este nivel exige delimitar el alcance.

La sensibilidad de 0.492 evidencia que la recuperación de objetos de referencia es más limitada que la precisión, el indicador advierte que la detección dentro del cuadro por sí misma no garantiza reconocer todos los elementos que podrían obstaculizar al robot.

El mAP50 de 0.495 resume el desempeño con un criterio de solapamiento de 0.50. El mAP50-95 de 0.389 es menor porque tiene exigencias más estrictas que de la localización, la diferencia de 0.106 muestra la sensibilidad del resultado a la precisión espacial de las cajas mas no representa

una caída de 10.6 % en la tasa de maniobras. Una localización menos ajustada también puede afectar el centro elegido para consultar profundidad, especialmente cuando la caja incluye fondo o superficies a diferentes distancias.

Estas métricas validan la obtención de un modelo detector funcional, pero no una detección de alto rendimiento en cualquier escenario, la contribución de la tesis incluye su integración con profundidad y actuación sobre la plataforma, no únicamente el valor de mAP.

4.3.3 Inferencia y límites de interpretación

La Fig. 4.10 muestra el funcionamiento del detector general nativo en una escena de interior en fase de prueba, con elementos como *person*, *cell pone* y *bed*, así se pudo optar por la elección del modelo base pues no solamente nos brindaba clases, sino que también nos brindaba un nivel de confiabilidad de los elementos.

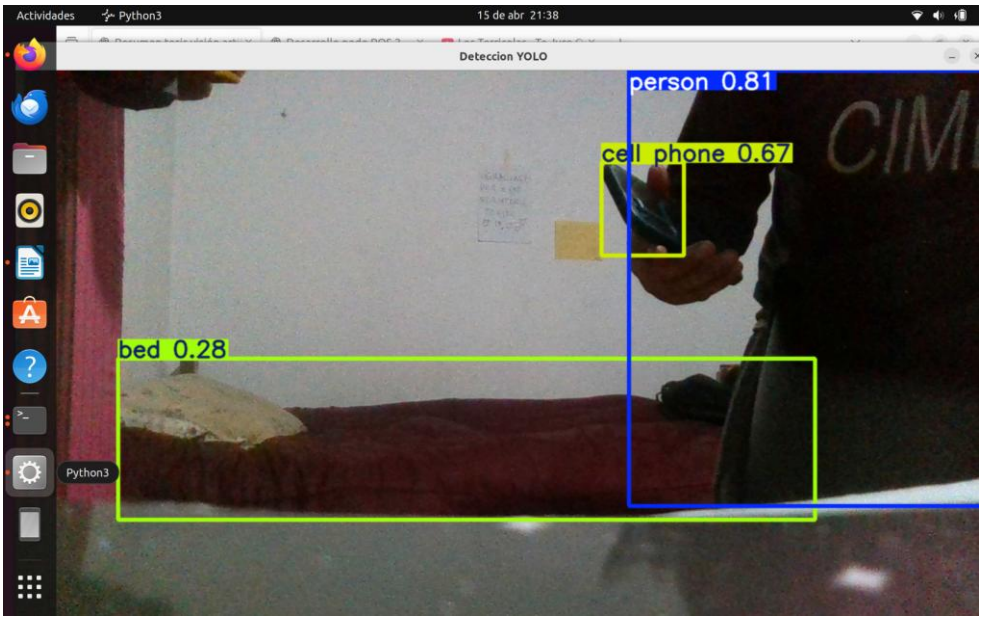


Fig. 4.10 Detecciones del modelo base en una escena interior.

La validación del modelo trabaja con etiquetas referenciales, la visualización muestra predicciones sobre una escena, el ODS contiene mediciones y respuestas físicas. El modelo nativo y el propio pueden trabajar en conjunto para sustentar el diseño del algoritmo y su aplicación, con el apoyo del dataset elaborado se puede tener una validación apropiada con respecto a su interpretación.

4.4 Migración de archivos y modelos a la Jetson Orin Nano

Para la ejecución embebida se organiza el proyecto bajo el nombre de: *~/tesis_julian*. Las rutas más sobresalientes con respecto al proyecto son: *~/tesis_julian/modelos/yolov8n.pt*, *~/tesis_julian/modelos/best_personalizado.pt* y *~/tesis_julian/ros2_ws_tesis*. Esta organización permite separar el sistema actual de otros módulos ya incluidos en la Jetson, la ubicación de los archivos así como las versiones se encuentran disponibles en el repositorio GitHub adjuntado en los Anexos de la tesis.

4.4.1 Organización del workspace y modelos

En la Tabla 4.5 se encuentran documentadas las correspondientes rutas de los archivos, la estructura separa los pesos de los modelos y el paquete ROS 2.

En la laptop que se utiliza en el desarrollo se ocupa *~/ros2_ws*, mientras que en la Jetson Orin se crea *~/tesis_julian/ros2_ws_tesis* identificando el espacio de trabajo, se hace esta distinción para explicar el por qué una ruta válida en el desarrollo podría llegar a fallar si se cambiase de equipo o de directorio de ejecución.

Tabla 4.5 Organización de los archivos

Ubicación	Función
~/ros2_ws	Workspace inicial en laptop
~/ros2_ws/src/primer_paquete/primer_paquete/lector_camara.py	Nodo inicial
~/tesis_julian/ros2_ws_tesis	Workspace de aplicación Jetson
~/tesis_julian/ros2_ws_tesis/src/primer_paquete/primer_paquete/fusion_realsense_nav.py	Nodo de visión integrado
/home/jetson/tesis_julian/modelos/yolov8n.pt	Modelo base
/home/jetson/tesis_julian/modelos/best_personalizado.pt	Modelo personalizado
primer_paquete / puente_arduino	Ejecutable del enlace funcional

4.4.2 Diferencias entre desarrollo e inferencia embebida

El entorno de desarrollo centra su trabajo en la preparación de datos y entrenamiento mediante la CPU, por otro lado, la Jetson concentra su trabajo en la adquisición, inferencia, decisión y comunicación hacia el Arduino. Hacer estas funciones por separado ofrece la oportunidad de trasladar los modelos ya entrenados sin necesidad de ejecutar nuevamente el proceso de aprendizaje durante las pruebas del robot, las mediciones de RAM, GPU y temperatura registradas en la hoja de cálculos corresponden únicamente al sistema embebido, por lo tanto, la duración de entrenamiento de la laptop no se utiliza para describir el rendimiento de inferencia de la Jetson.

4.4.3 Herramientas de software empleadas

La implementación del sistema requiere de diferentes herramientas de software destinadas a diferentes ámbitos como la adquisición de información, ejecución y prueba de modelos de detección, procesamiento de imágenes y comunicación de componentes, en la Tabla 4.6 resume los principales recursos empleados.

Tabla 4.6 Herramientas de software utilizadas

Herramienta	Función
Ros 2 Humble	Comunicación entre nodos
Python	Desarrollo y ejecución del sistema
Realsense2_camera	Acceso a imagen RGB y profundidad
Rclpy	Creación y manejo de nodos ROS 2
cv_bridge	Conversión de imágenes ROS-OpenCV
Ultralytics	Entrenamiento e inferencia con YOLOv8
NumPy	Manejo de datos y martices
OpenCV	Procesamiento y visualizacón de imágenes
PySerial	Comunicación serial con Arduino

4.5 Integración de la cámara Intel RealSense D415 en ROS 2

La cámara se ejecuta mediante el paquete de *realsense2_camera* con perfiles RGB y profundidad configurados en 640x480 a 30 cuadros por segundo. Las mediciones registradas en la hoja de cálculo indican como resultado una frecuencia RGB entre 29.31 y 29.74 Hz y una frecuencia de profundidad entre 25.57 y 29.44 Hz mostrado concordancia con los 30 cuadros señalados.

4.5.1 Adquisición RGB y profundidad

El paquete *realsense2_camera* publica los flujos: *camera/camera/color/image_raw* y */camera/camera/depth/image_rect_raw*. La imagen RGB alimenta el proceso de inferencia y el mapa de profundidad añade el componente de valores de distancia. En la Fig. 4.11 se muestra una ventana de detección con distancias en milímetros asociadas a distintas cajas, los centros marcados, así como los textos permiten reconocer la introducción del dato espacial a la visualización obtenida.

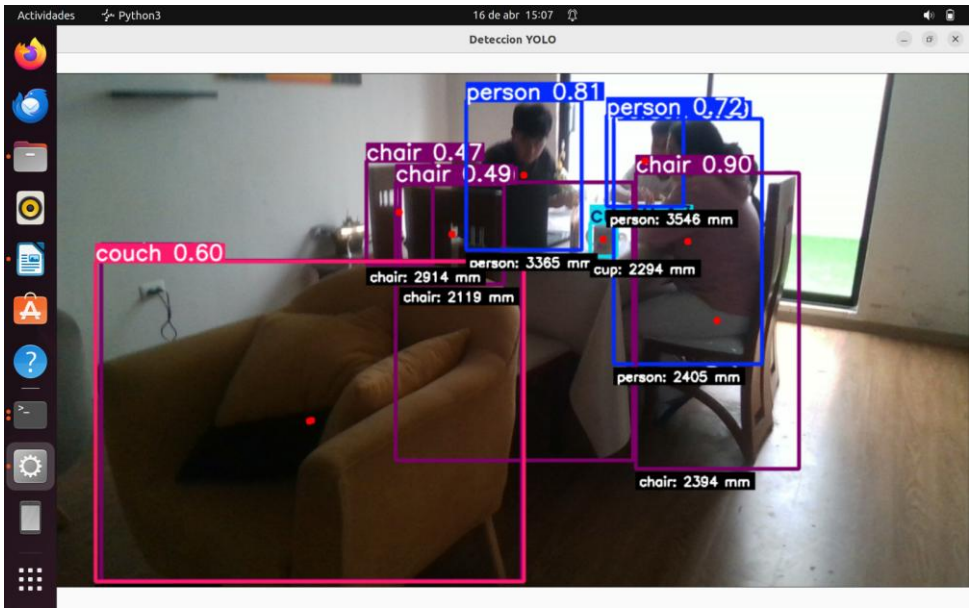


Fig. 4.11 Detección general con puntos de centro y distancias en milímetros.

4.5.2 Consulta de profundidad y asociación con cajas

Para obtener la distancia asociada a cada detección se recurre a utilizar el centro de la caja delimitadora generada por el modelo, pero a partir de este punto se consulta una vecindad de hasta 5 x 5 píxeles sobre la imagen de profundidad siempre considerando únicamente los valores positivos disponibles.

La distancia final se obtiene utilizando la mediana de estos valores, reduciendo la dependencia de una única lectura puntual y añadiendo un valor representativo de profundidad para cada detección.

En resumen, para una caja delimitadora definida por (x_1, y_1) y (x_2, y_2), el punto central se determina mediante Ec. 4.1 y Ec. 4.2.

Ecuación 4.1 Coordenada horizontal del centro de la caja delimitadora

$$C_x = \frac{(x_1 + x_2)}{2}$$

Ecuación 4.2 Coordenada vertical del centro de la caja delimitadora

$$C_y = \frac{(y_1 + y_2)}{2}$$

4.6 Fusión de modelos y lógica de navegación

4.6.1 Complementariedad entre modelo base y personalizado

La solución al modelo creado combina yolov8n.pt para las categorías generales mientras que best_personalizado.pt para las clases anteriormente mencionadas: *door*, *wall*, y *window*. Ambos procesan el cuadro RGB y sus detecciones se reúnen para la evaluación espacial final.

La Fig. 4.12 resume la secuencia desde adquisición de datos hasta publicación de acción, las cajas permiten localizar regiones y sus centros mientras que la profundidad permite ordenar los obstáculos válidos por cercanía, la zona horizontal da paso a la intervención. En otras palabras, la clase aporta interpretación del entorno, pero los estados se obtienen de la combinación de distancias y zona, esta organización establecida conserva el detector base sin exigirle al modelo personalizado aprender nuevamente todas sus categorías.

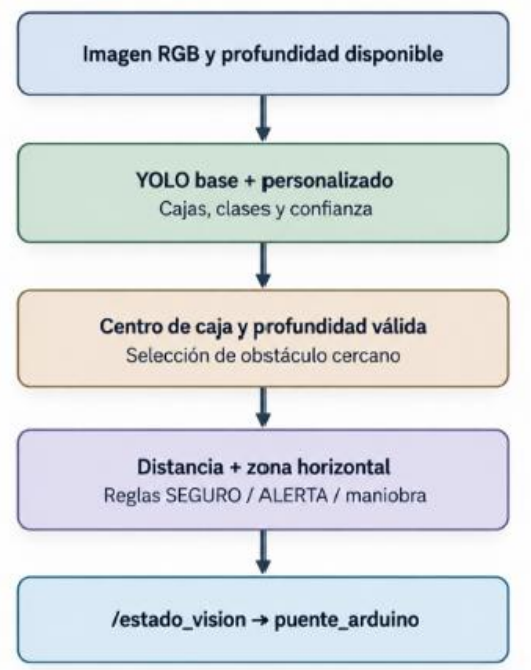


Fig. 4.12 Flujo de detección, profundidad y decisión del sistema integrado.

4.6.2 Definición de zonas y prioridad de cercanía

La imagen se divide horizontalmente en tercios, para un ancho *w*, un centro *cx* menor que *w/3* se clasifica a la izquierda, para un centro mayor que *2w/3* a la derecha, los restantes se consideran como centrales. Los límites forman parte de una regla de implementación y no a una medición del ancho físico del robot; la zona indica la posición relativa dentro de la imagen y no una trayectoria libre calculada en el espacio tridimensional.

Entre las detecciones con profundidad válida se toma en cuenta como prioridad a la más cercana, esta elección permite que el estado del robot responda al obstáculo más cercano dentro de las detecciones consideradas y evitar la posible colisión u obstáculo más relevante.

4.6.3 Estados y umbrales

La Tabla 4.6 muestra las fronteras con desigualdades para poder evitar ambigüedad en las lecturas.

Tabla 4.7 Reglas de profundidad y comandos.

Distancia d (mm)	Zona	Estado	Comando
$d < 400$	Cualquiera	DETENER	X
$400 \leq d < 700$	Izquierda	GIRO DERECHA	D
$400 \leq d < 700$	Centro	DETENER	X
$400 \leq d < 700$	Derecha	GIRO IZQUIERDA	A

$700 \leq d < 1000$	Cualquiera	ALERTA	L
$d \geq 1000$	Cualquiera	SEGURO	O

DETENER tiene prioridad por debajo de 400 mm cuando existe una distancia válida, entre 400 y 700 mm el centro provoca la detención si es el valor más bajo mientras que si es en el umbral de estos 2 los lados provocan un giro contrario. Esa regla es local y reactiva es decir no verifica si existe o no una salida totalmente limpia o libre ni tampoco calcula la trayectoria futura sino que su utilidad experimental se limita a las maniobras registradas, pero habiendo algún tipo de obstáculo al otro lado y al ser la distancia menor a 400 mm este se detendrá por lo tanto su objetivo de cuidar la integridad del robot se cumple. SEGURO significa que la regla no requiere intervención en ese instante, pero de igual manera tampoco certifica la ausencia completa y libre de obstáculos.

4.7 Ejecución del sistema en la Jetson Orin Nano

La ejecución final se realiza con el uso de tres componentes funcionales: cámara *RealSense*, nodo *fusion_realsense_nav* y nodo *puente_arduino*. El nodo de visión publica */estado_visión*, mientras que *puente_arduino* recibe el estado, lo traduce a comandos funcionales para el Arduino y este interviene sobre el movimiento del robot.

4.7.1 Ejecución y observación del nodo de visión

La Fig. 4.13 muestra el nodo identificado como *lector_camara_fusion*, asociada al ejecutable *fusión_realsense_nav*, se pueden observar clases, distancias, zonas y estados desde *~/tesis_julian/ros2_ws_tesis*. La captura contiene diferentes categorías incluidas algunas generales del modelo base pero la aparición de una etiqueta en terminal no es suficiente para confirmar que

esa clase sea correspondiente al 100% al objeto físico dentro de la escena pero sí permite comprobar que existe un flujo produciendo observaciones y decisiones en la Jetson.

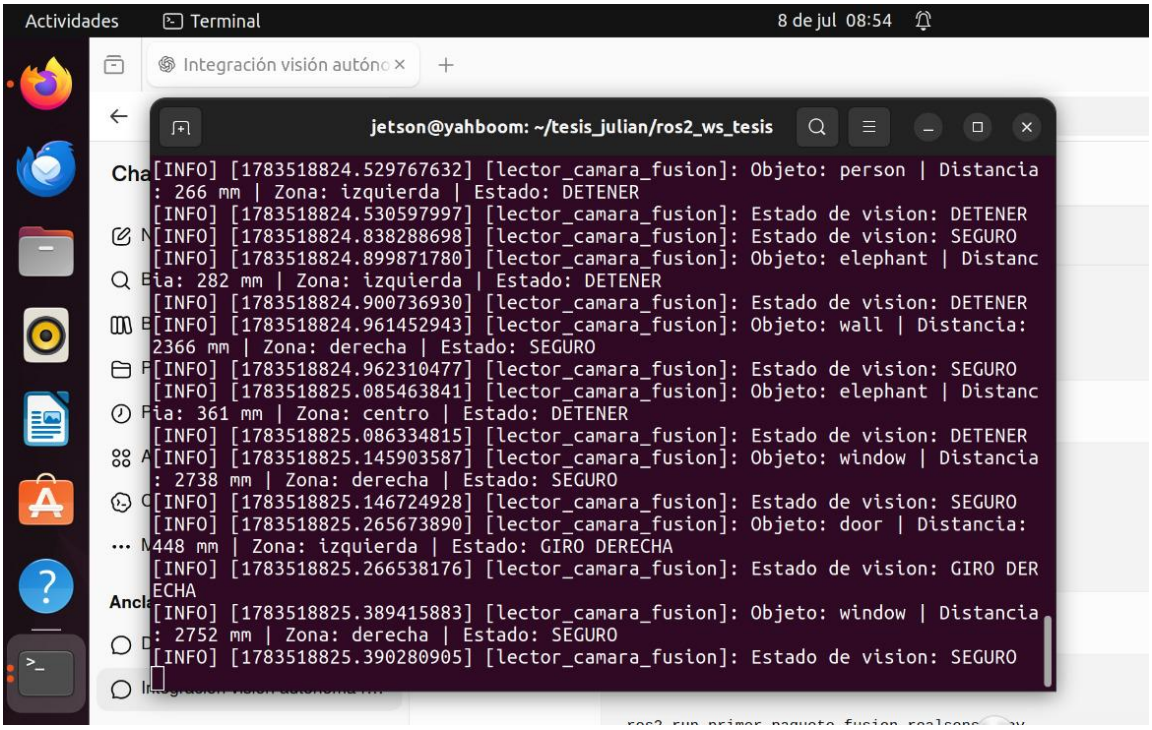


Fig. 4.13 Registro de distancias, zonas y estados en la Jetson

La ejecución sin una ventana gráfica explícita permite observar el sistema mediante terminal en el proceso de integración. Los elementos distinguibles son: *fusion_realsense_nav* que se utiliza para lanzar el software, *lector_camara_fusion* que identifica el nodo en los mensajes y *fusion_realsense_nav.py* que contiene el desarrollo en Python.

4.7.2 Integración funcional con puente_arduino

La Fig 4.14 presenta de manera simultánea registros de cámara, visión y puente. Se pueden identificar los estados SEGURO y ALERTA con su respectiva traducción a O y L evidenciando que la señal de salida llega al componente encargado de comunicarla al Arduino.

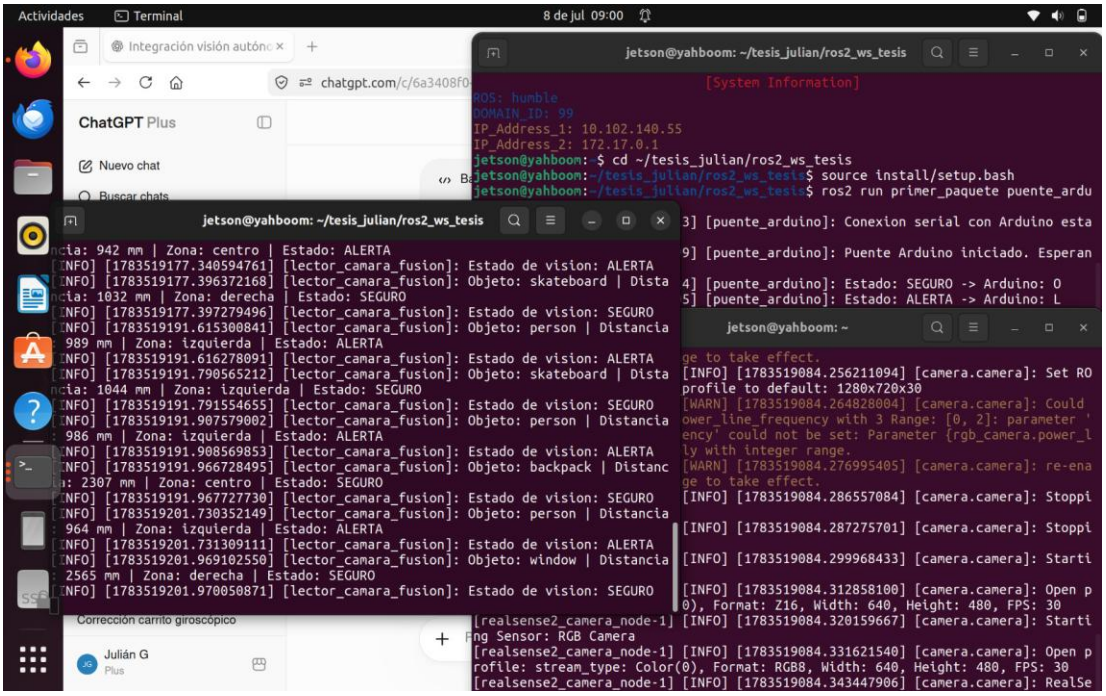


Fig. 4.14 Ejecución conjunta de cámara, visión y puente en la validación final.

La Fig 4.15 contiene los estados del *puente_arduino*, es decir se centra exclusivamente en la comunicación con este, se puede observar los estados y la ejecución de la asignación de la acción dentro del Arduino, en este caso como ejemplos podemos citar los visualizados en la Tabla 4.8.

Tabla 4.8 Estado y señal enviada hacia Arduino

ESTADO	ARDUINO
GIRO DERECHA	D
ALERTA	L
SEGURO	O

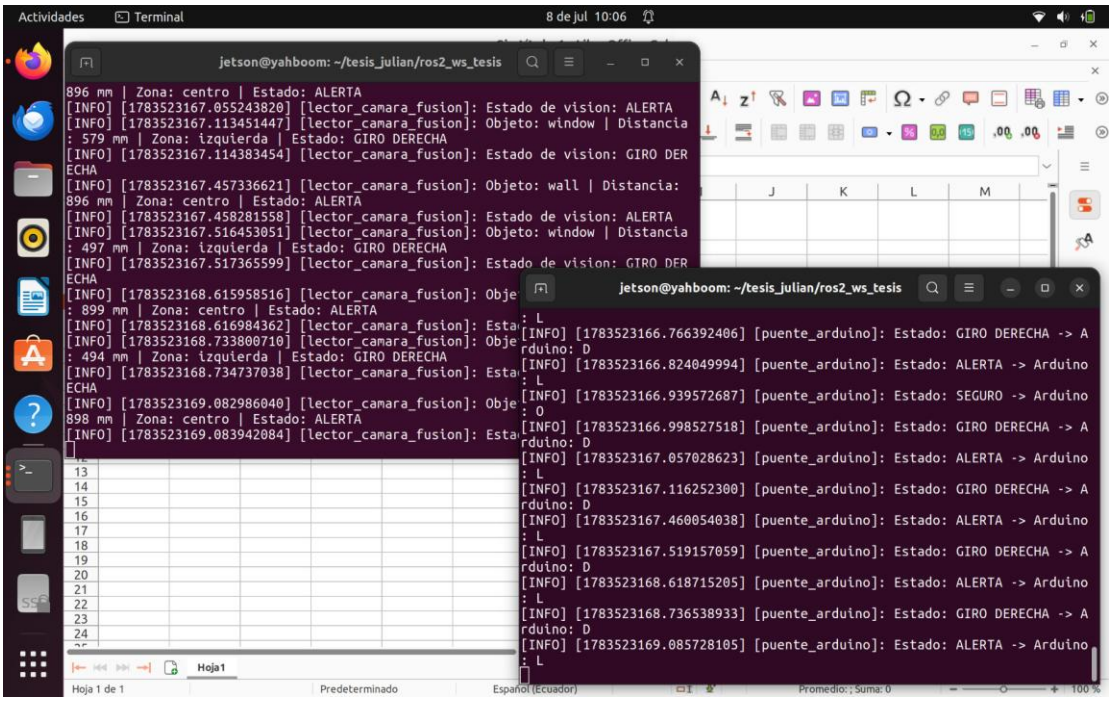


Fig. 4.15 Visualización de la ejecución *puente_arduino*

La Fig. 4.16 por otro lado muestra la ejecución de *lector_camara_fusion*, evidenciando el estado de la visión, la etiqueta del objeto en cuestión, la distancia hacia este, la zona en la cual se encuentra ubicado y una acción a realizarse, se utilizan algunos de estos ejemplos para mostrar su visualización en la terminal mediante la Tabla 4.9.

Tabla 4.9 Ejemplos de visualización

Objeto	Distancia (mm)	Zona	Estado
person	448	izquierda	GIRO DERECHA
cat	583	centro	ALERTA

door	620	derecha	GIRO IZQUIERDA

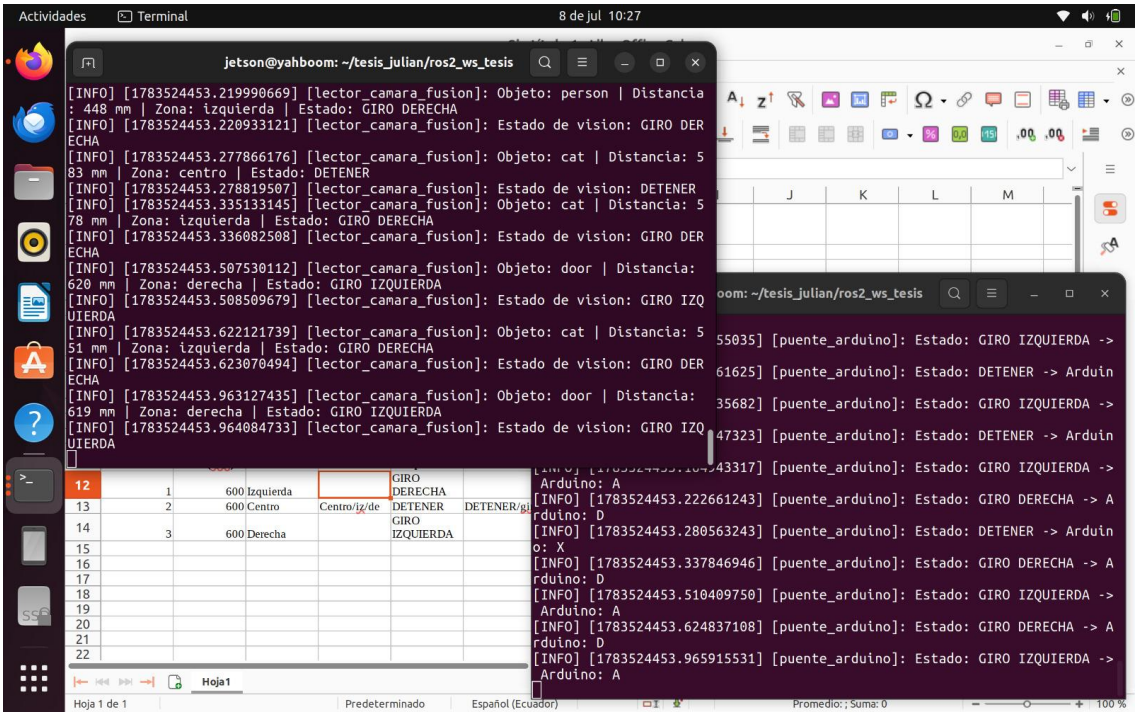


Fig. 4.16 Visualización de la ejecución lector_camara_fusion

La evidencia conjunta confirma de manera contundente el enlace funcional no mostrado en la arquitectura visual inicial, el nodo de visión publica */estado_vision* mientras que *puente_arduino* recibe y traduce las señales, Arduino interviene de manera directa en el movimiento y la conducción habitual continúa siendo mediante la APK/Bluetooth desarrollada previamente.

4.8 Revisión del ecosistema previo del robot

La revisión llevada a cabo sobre el sistema previo identifica funciones de percepción, telemetría y odometría. Los registros asocian *nodo_base_py* con lectura de RPM y publicación de */rpm_data*, convertidos.py con */wheel_velocities* y odometry_node.py con */odom*. Son registros previos

existentes en la Jetson que continúan como parte del robot, pero no tienen ninguna inferencia en la tesis desarrollada ni viceversa.

El análisis original evidencia las limitaciones obvias del sistema de visión antecesor: Jetson Nano, Kinect y Raspberry Pi 5. La solución final es reemplazar esos componentes con: Jetson Orin Nano e IntelRealsense D415, siendo la primera la razón por la cual la Raspberry Pi 5 no fuera necesaria en este proceso.

La conservación del control manual previamente realizado fue una decisión por mantener una armonía dentro de las mejoras realizables al robot. La visión añade asistencia preventiva sobre una plataforma ya construida, no sustituye la arquitectura mecánica ni desarrolla una navegación autónoma completa, habiendo declarado esto la delimitación del trabajo puede servir como base para futuros desarrollos.

4.9 Estado final, pruebas y análisis de resultados

La validación final se obtiene mediante pruebas de distancias, zonas, transición de estados y rendimiento. Datos recolectados de forma práctica con la utilización de tablas son la fuente directa de los resultados dentro de un ambiente de trabajo común para el robot como se observa en la Fig. 17.

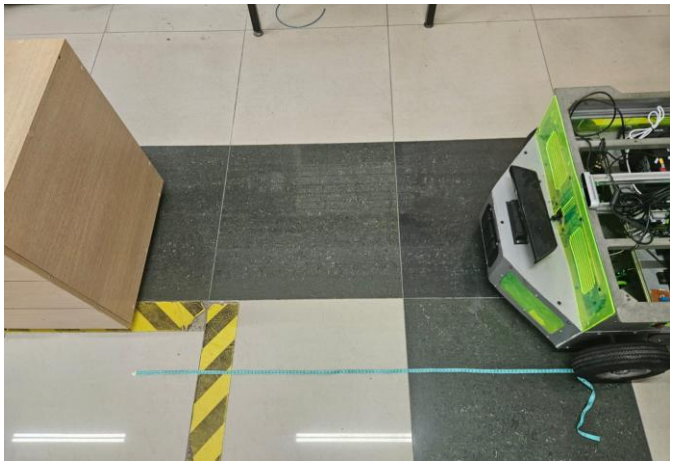


Fig. 4.17 Toma de datos realizada dentro del entorno de trabajo.

4.9.1 Organización y alcance de la evidencia experimental

Los registros contienen cuatro bloques: siete condiciones de distancia, tres posiciones espaciales, nueve registros de transición correspondientes a tres repeticiones y nueve parámetros de rendimiento.

4.9.2 Distancia real frente a profundidad RealSense

Las Tablas 4.10 y 4.11 contienen los siete casos con respecto a la zona central, en 1200, 900 y 600 mm, los estados registrados mantienen coherencia con lo esperado, las condiciones de 1000 y 700 mm incluyen lecturas cercanas a los umbrales: 997/1001 y 699/702 mm, dando una explicación razonable de los estados vecinos en estos. Mientras que en 400 y 300 mm tienden a señalar DETENER/SEGURO siendo este no correspondiente a lo esperado, pero tomando en cuenta que a partir de menos de 400 mm el lector de profundidad tiende a fallar y enviar valores erróneos.

Tabla 4.10 Pruebas de distancia con valores registrados.

Prueba	Real (mm)	Objeto	RealSense (mm)	Zona
1	1200	caja	1195	Centro
2	1000	caja	997/1001	Centro
3	900	caja	898	Centro
4	700	caja	699/702	Centro
5	600	caja	601	Centro
6	400	caja	403	Centro
7	300	caja	305/304	Centro

Tabla 4.11 Pruebas con estados y aciertos.

Prueba	Estado esperado	Estado obtenido	Correcto
1	SEGURO	SEGURO	Sí
2	Límite SEGURO/ALERTA	SEGURO/ALERTA	Sí
3	ALERTA	ALERTA	Sí
4	Límite ALERTA/DETENER	ALERTA/DETENER	Sí
5	DETENER	DETENER	Sí
6	Límite DETENER	DETENER/SEGURO	No

7	DETENER	DETENER/SEGURO	No

RealSense D415 presenta mediciones adecuadas entre 600 y 1200 mm, teniendo en cuenta también los cambios esperados en los umbrales de 700 y 1000 mm, pero las principales inconsistencias se observan a 400 y 300 mm, donde aparecen estados erróneos relacionados netamente a la pérdida de confiabilidad del sensor a corta distancia.

En la Fig. 4.18 se presenta cada valor de profundidad que aparecen en las celdas incluidos los dos valores críticos separados por una barra, la línea discontinua indica una práctica igualdad entre lo real y lo medido, la proximidad de los puntos en la diagonal describe las lecturas realizadas y evidencia la inestabilidad con los estados en distancias cortas.

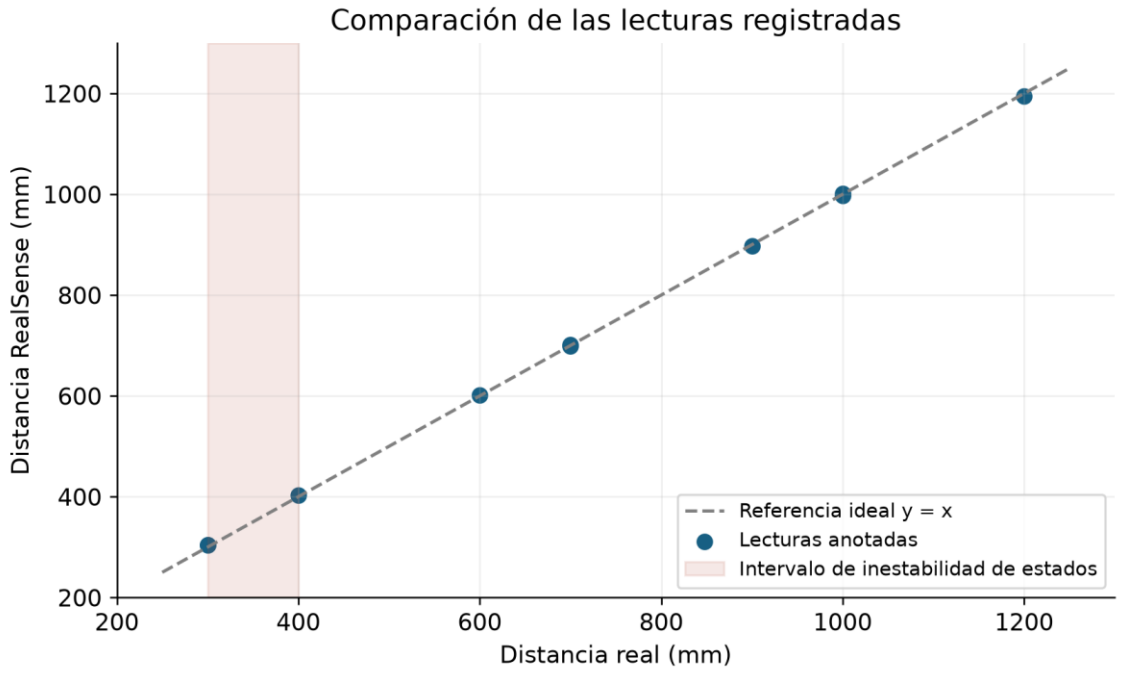


Fig. 4.18 Distancia real y valores de profundidad registrados.

4.9.3 Error de distancia y comportamiento de las mediciones

En base a las distancias registradas en las pruebas anteriores se procede a calcular la diferencia entre el valor obtenido por la Intel RealSense D415 y la distancia real de referencia, para cada lectura se determina el error Ec. 4.3 el error absoluto Ec. 4.4 y el error porcentual Ec. 4.5 reflejados en la Tabla 4.12. Cuando una misma condición presenta dos valores registrados ambos se analizan de manera individual.

Ecuación 4.3 Error

$$e = d_{RS} - d_{real}$$

Ecuación 4.4 Error absoluto

$$EA = |d_{RS} - d_{real}|$$

Ecuación 4.5 Error porcentual

$$EP = 100 \, . EA/d_{real}$$

Tabla 4.12 Errores obtenido a partir de las lecturas de profundidad.

Prueba	Real (mm)	RS (mm)	Error (mm)	EA (mm)	EP (%)
1	1200	1195	-5	5	0.417
2	1000	997	-3	3	0.300
2	1000	1001	+1	1	0.100

3	900	898	-2	2	0.222
4	700	699	-1	1	0.143
4	700	702	+2	2	0.286
5	600	601	+1	1	0.167
6	400	403	+3	3	0.750
7	300	305	+5	5	1.667
7	300	304	+4	4	1.333

Los errores absolutos obtenidos se encuentran entre 1 y 5 mm, demostrando que las lecturas registradas permanecen bastante aproximadas a las distancias reales utilizadas como referencia, en las anotaciones comprendidas entre 600 y 1200 mm el error promedio es de aproximadamente 2.14 mm pero en las lecturas de 300 y 400 mm prácticamente se duplica aumentando a 4.00 mm.

La Fig. 4.19 refleja esta diferencia de manera gráfica, aunque el error numérico se mantiene reducido en todos los valores las mediciones realizadas a menor distancia presentan una mayor separación con respecto a la referencia.

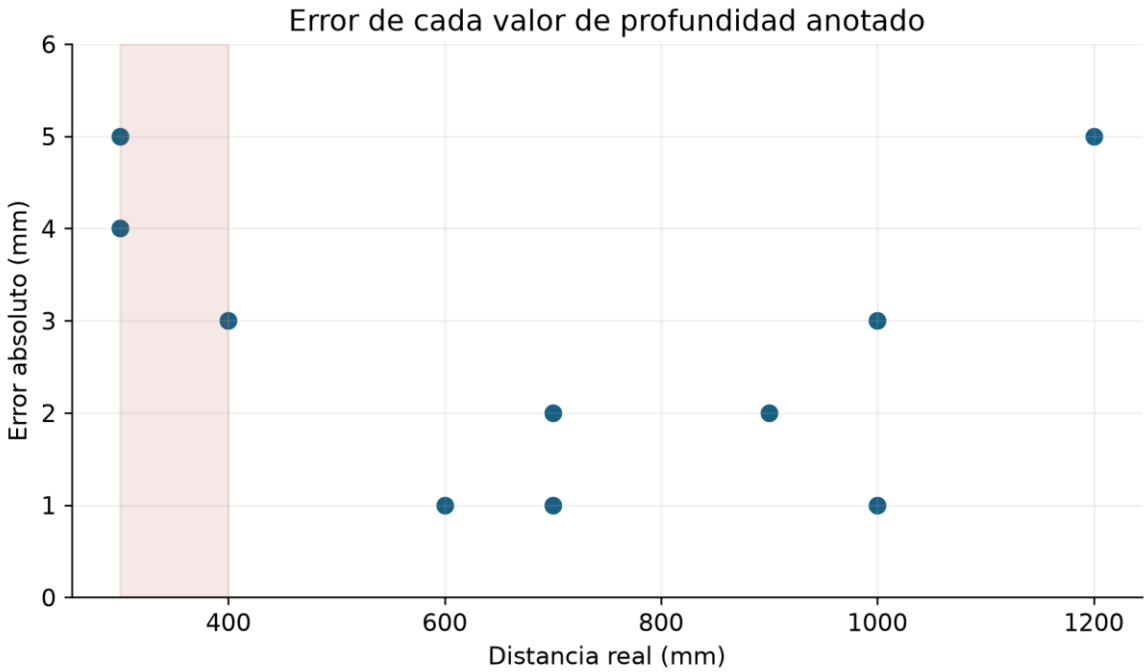


Fig. 4.19 Error absoluto de cada valor de distancia anotado.

Para obtener el indicador general del conjunto de lecturas se calcula el error absoluto medio (MAE), dando como resultado un valor de 2.70 mm para los diez registros obtenidos. Este valor resume la diferencia promedio entre la distancia real medida y la proporcionada por la cámara RealSense en el entorno.

Cabe aclarar que el comportamiento del sistema no depende solamente de que la distancia numérica sea cercana al valor real. En las pruebas de 300 y 400 mm se observaran cambios entre los estados DETENER y SEGURO pese a que algunas lecturas permanecen en concordancia. Por esta razón, el comportamiento a corta distancia se analiza posteriormente también desde la respuesta de la lógica de decisión y no únicamente desde el error de profundidad.

4.9.4 Pruebas por zonas y respuesta física

Tabla 4.13 Pruebas a 600 mm con ubicación y estados.

Prueba	Posición real	Zona detectada	Estado esperado	Estado obtenido
1	Izquierda	Izquierda	GIRO DERECHA	GIRO DERECHA
2	Centro	Centro / Izquierda / Derecha	DETENER	DETENER / GIRO IZQUIERDA / GIRO DERECHA
3	Derecha	Derecha	GIRO IZQUIERDA	GIRO IZQUIERDA

Tabla 4.14 Comando esperado y respuesta física.

Prueba	Comando esperado	Respuesta física
1	D	GIRO DERECHA
2	X	DETENCIÓN
3	A	GIRO IZQUIERDA

Con el afán de comprobar el comportamiento del sistema según la posición horizontal del obstáculo se realizan tres pruebas en un umbral de importancia en este caso de 600 mm reflejadas en las Tablas 4.13 y 4.14, se coloca el obstáculo en las zonas: izquierda, centro y derecha. La lógica implementada funciona estableciendo un guro en sentido contrario cuando el obstáculo se presenta en una zona lateral y una detención cuando aparece en la zona central.

En la prueba correspondiente a izquierda se registra una distancia de 588mm identificando la zona correctamente y como resultado de la lógica se genera el estado GIRO DERECHA enviando el comando D efectivamente girando a la derecha.

Mientras que en la prueba correspondiente a derecha se registró una distancia de 618 mm, la zona es reconocida satisfactoriamente y de igual manera generando la lógica del estado GIRO IZQUIERDA enviando el comando A y ejecutando el giro a la izquierda.

El comportamiento más particular y digno de ser tomado en cuenta es en la condición central, durante esta prueba se registraron valores de profundidad de 605 y 597 mm, pero lo que más destacó es que la zona detectada alterna entre centro, izquierda y derecha. Esta variación produce cambios repentinos y rápidos entre los estados de detección y giro provocando un avance con pequeñas correcciones laterales de ambos lados similar a un titubeo o temblor del robot pero a medida de que el robot se aproximó al obstáculo, la detección central se confirma generando la orden de DETENER correspondiente al comando X deteniendo al robot.

4.9.5 Transiciones entre estados

Para documentar el cambio dentro del sistema al aproximarse y alejarse de un obstáculo se realizaron tres repeticiones sobre los umbrales particularmente críticos dentro del cambio de estados: SEGURO, ALERTA Y DETENER, conforme con la lógica implementada la transición los resultados obtenidos se muestran en las Tablas 4.15 y 4.16.

Tabla 4.15 Transiciones registradas en las pruebas y distancias

Repetición	Transición evaluada	Teórica (mm)	Observada (mm)
1	SEGURO → ALERTA	1000	990
1	ALERTA → DETENER	700	701
1	DETENER → ALERTA/SEGURO	700/1000	689-1003
2	SEGURO → ALERTA	1000	993
2	ALERTA → DETENER	700	698
2	DETENER → ALERTA/SEGURO	700/1000	701/1005
3	SEGURO → ALERTA	1000	1002
3	ALERTA → DETENER	700	706
3	DETENER → ALERTA/SEGURO	700/1000	702-992

Tabla 4.16 Estados y resultado de la lógica

Repetición	Estado antes	Estado después	Correcto
1	SEGURO	ALERTA	Sí
1	ALERTA	DETENER	Sí
1	DETENER	ALERTA - SEGURO	Sí
2	SEGURO	ALERTA	Sí

2	ALERTA	DETENER	Sí
2	DETENER	ALERTA – SEGURO	Sí
3	SEGURO	ALERTA	Sí
3	ALERTA	DETENER	Sí
3	DETENER	ALERTA - SEGURO	Sí

Las transiciones registradas se producen de manera satisfactoria cerca a los umbrales establecidos de 1000 mm, para el cambio de SEGURO a ALERTA y 700 mm para ALERTA a DETENER. En las tres repeticiones se observa pequeñas variaciones pero que fluctúan dentro de estos valores con desviaciones medias absolutas de 6.33 mm y 3.00 mm respectivamente.

Durante el alejamiento del obstáculo también se puede verificar la transición inversa es decir pasar de DETENER a ALERTA y posteriormente a SEGURO a medida que va incrementando la distancia. Con todo esto señalado, las pruebas muestran una correspondencia satisfactoria entre lo establecido en la lógica y lo realizado en la práctica.

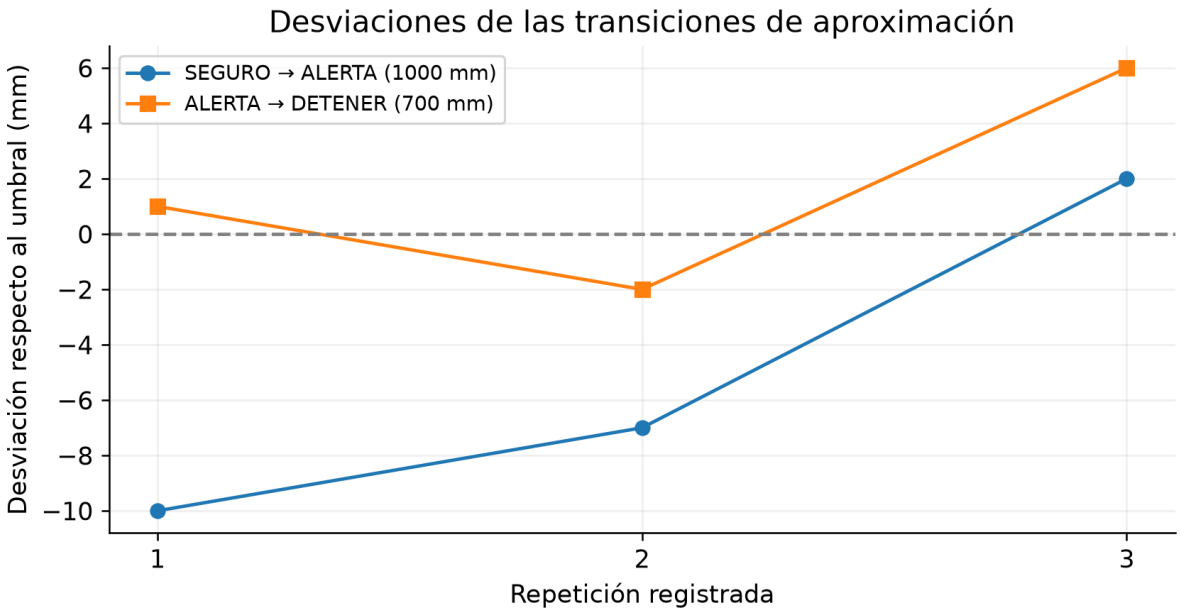


Fig. 4.20 Desviaciones de las transiciones de aproximación.

La Fig. 4.20 permite visualizar estas diferencias con respecto a los umbrales programados en la lógica, evidencian que los cambios de estados se mantienen cerca de las referencias establecidas.

4.9.6 Limitación física y operativa entre 300 y 400 mm.

En las pruebas hechas a 400 y 300 mm se registran valores de profundidad muy cercanos a las distancias reales, pero a la vez el sistema presenta variaciones entre los estados DETENER y Seguro, indicando que, pese a que la cámara todavía puede entregar lecturas de métricas acertadas, el comportamiento de la percepción de la lógica no es estable cuando el obstáculo se encuentra demasiado cerca.

Esta variación va de la mano con las limitaciones propias en la medición de profundidad a corta distancia del modelo Intel RealSense D415, en estas condiciones suelen presentarse pérdidas o

variaciones en los valores de profundidad que utiliza la lógica de decisión afectando de manera directa el estado del sistema.

Razón por la cual el intervalo comprendido entre los 300 y 400 mm se considera una zona de menor confiabilidad operativa en lo referente a navegación y percepción de profundidad. La regla de detención establecida para distancias inferiores a 400 mm es completamente válida pues si bien cambia de estado de manera constante el robot tiende a detenerse por completo.

4.9.7 Frecuencias de publicación y capacidad de operación

Durante las pruebas se registran frecuencias de publicación que corresponden a la imagen RGB, la información de profundidad y */estado_vision*, la cámara RGB está prácticamente estable en 30 Hz, mientras que la profundidad presenta valores de entre 25.57 y 29.44 Hz, por otra parte */estado_vision* se mantiene entre 17.87 y 18.38 Hz como refleja en la Tabla 4.17.

Tabla 4.17 Frecuencias registradas durante la ejecución del sistema

Parámetro	Valor (Hz)	Observación
Publicación de /estado_vision	17.81 – 18.38	Publicación estable del estado de decisión
Cámara RGB	29.31 – 29.74	Funcionamiento cercano a los 30 Hz configurados
Cámara de profundidad	25.57 – 29.44	Frecuencia variable pero satisfactoria

La Fig. 4.21 permite comparar de forma directa estas frecuencias, observando que los flujos de RGB y de profundidad operaran a una frecuencia superior a la publicación de */estado_vision*, esta

diferencia tiene coherencia con el procesamiento adicional realizado por el sistema al tener dos modelos de visión artificial además de generar una decisión.

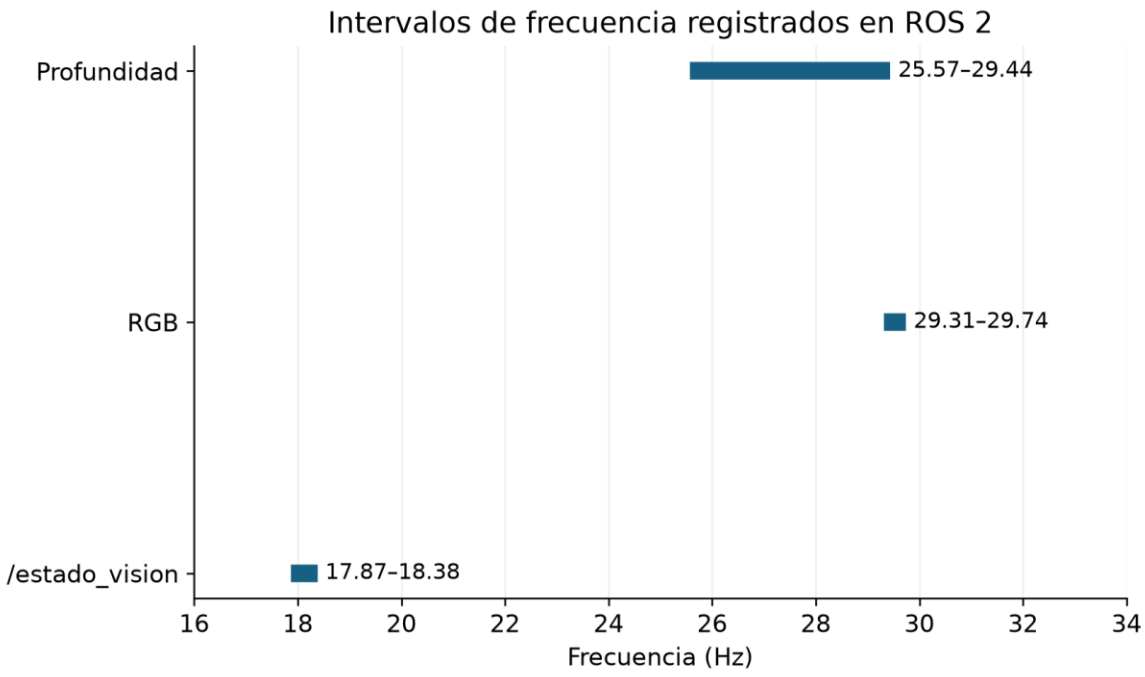


Fig. 4.21 Intervalos registrados de frecuencia RGB, profundidad y /estado_vision

La muestra de */estado_vision* se mantiene alrededor de 18 Hz equivalente a un nuevo estado cada 0.05 – 0.06 segundos aproximadamente, en la ejecución de las pruebas la respuesta del sistema se percibe de manera inmediata ignorando este posible problema sin retardos evidentes en la inferencia del robot.

4.9.8 Memoria, GPU, temperaturas y potencia

Durante la ejecución del sistema de visión se lleva a cabo un monitoreo de los principales recursos utilizados por la Jetson Orin Nano mediante tegrastats, considerando uso de memoria

RAM, memoria swap, actividad de GPU, temperaturas de CPU y GPU además de la potencia de entrada reportada.

Tabla 4.18 Recursos y condiciones de operación registrados

Parámetro	Valor	Unidad	Observación
Uso de memoria RAM	2991 / 7620	Mb	Aproximadamente 39.2% de utilización
Uso de swap	0 / 12002	Mb	No se registra uso de memoria de intercambio
Uso de GPU / GR3D	29 – 98	%	Actividad variable en la ejecución
Temperatura CPU	73.0 – 74.1	°C	Comportamiento térmico estable
Temperatura GPU	73.1 – 74.0	°C	Valores cercanos a los registrados en CPU
Voltaje	11.27 – 11.48	W	Potencia de entrada durante la ejecución

Los resultados de la Fig. 4.22 y la Tabla 4.18 muestran que la ejecución del sistema solamente utiliza cerca del 39% de la capacidad de la memoria disponible sin la necesidad de recurrir a memoria swap, indicando que durante las condiciones evaluadas la plataforma es suficientemente capaz de ejecutar simultáneamente los componentes principales del sistema de visión.

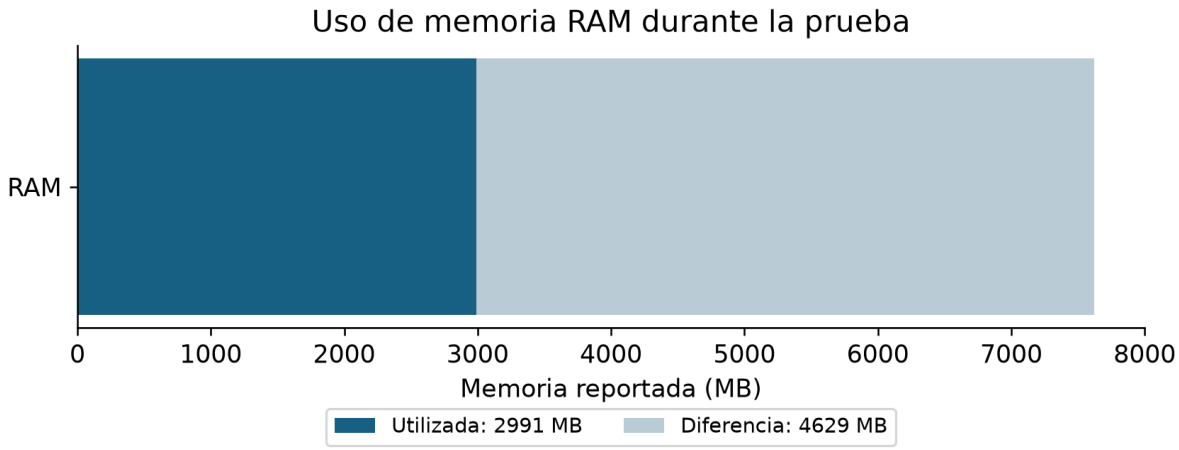


Fig. 4.22 Uso de memoria RAM

La actividad de la GPU presenta algunas variaciones a lo largo de la ejecución alcanzando valores elevados durante ciertos momentos de procesamiento, confirmando la participación de todos los recursos gráficos de la Jetson Orin Nano en la inferencia de decisiones y el funcionamiento integrado del sistema.

Con respecto al comportamiento térmico Fig. 4.23 la CPU y la GPU se mantienen en temperaturas similares cercanas a los 73 – 74 °C sin la presencia de interrupciones o caídas a lo largo de la realización de las pruebas, al igual que la potencia de entrada *voltaje* permanece cercana a 11W evidenciando poca variación.

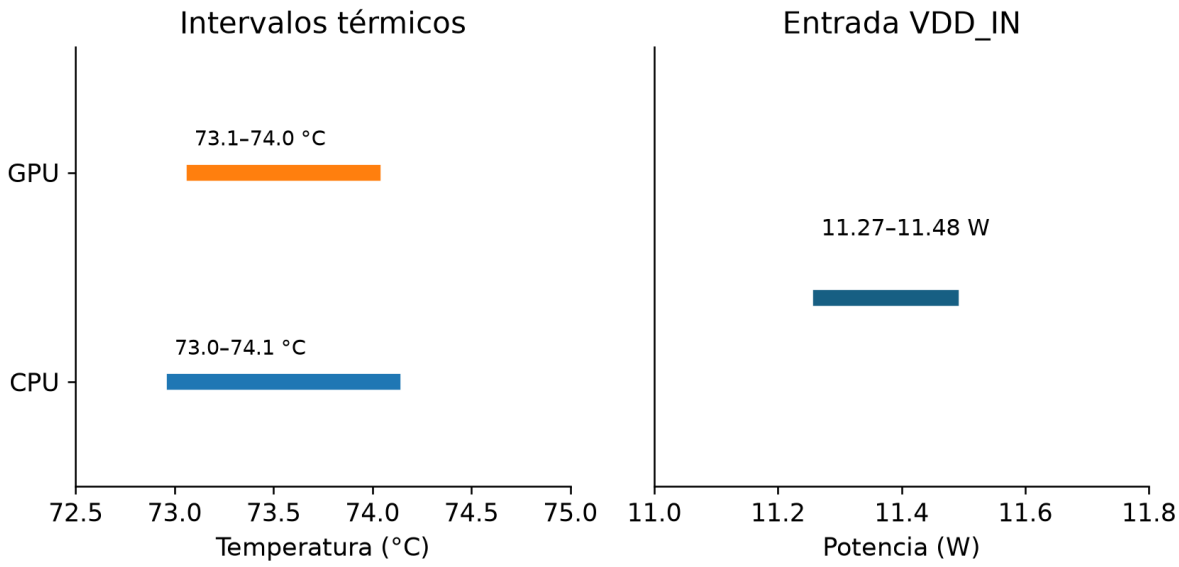


Fig. 4.23 Rangos de temperatura y potencia en la ejecución

Los valores registrados muestran que la Jetson Orin Nano ejecuta de manera satisfactoria la adquisición de imágenes, inferencia, procesamiento y comunicación del sistema mientras mantiene un uso relativamente moderado de memoria y un comportamiento térmico estable durante la ejecución de las pruebas.

4.9.9 Evaluación de errores

Se realiza una evaluación final al modelo bajo diferentes condiciones de iluminación. Se identifican posibles errores de reconocimiento alertando la influencia de las condiciones ambientales sobre la capacidad de detección del sistema ilustrado en la Tabla 4.19.

Los datos se registrados se toman mediante 90 simulaciones dirigidas a las tres clases implementadas: door, wall y window. Se evalúa de manera distribuida entre iluminación normal, iluminación baja y contraluz.

Tabla 4.19 Evaluación de reconocimiento bajo diferentes condiciones de iluminación

Iluminación	Objeto	Muestras	Reconocidos Correctamente	Otra etiqueta	Sin detección
Normal	door	10	7	1	2
Normal	wall	10	7	0	3
Normal	window	10	5	1	4
Baja	door	10	5	1	4
Baja	wall	10	4	2	4
Baja	window	10	5	1	4
Contraluz	door	10	3	2	5
Contraluz	wall	10	5	1	4
Contraluz	window	10	3	1	6
Total		90	44	10	36

Se obtiene una tasa de reconocimiento correcto del 63.33% en condiciones de iluminación normal, en iluminación baja disminuye al 46.67% y finaliza con 36.67% a contraluz.

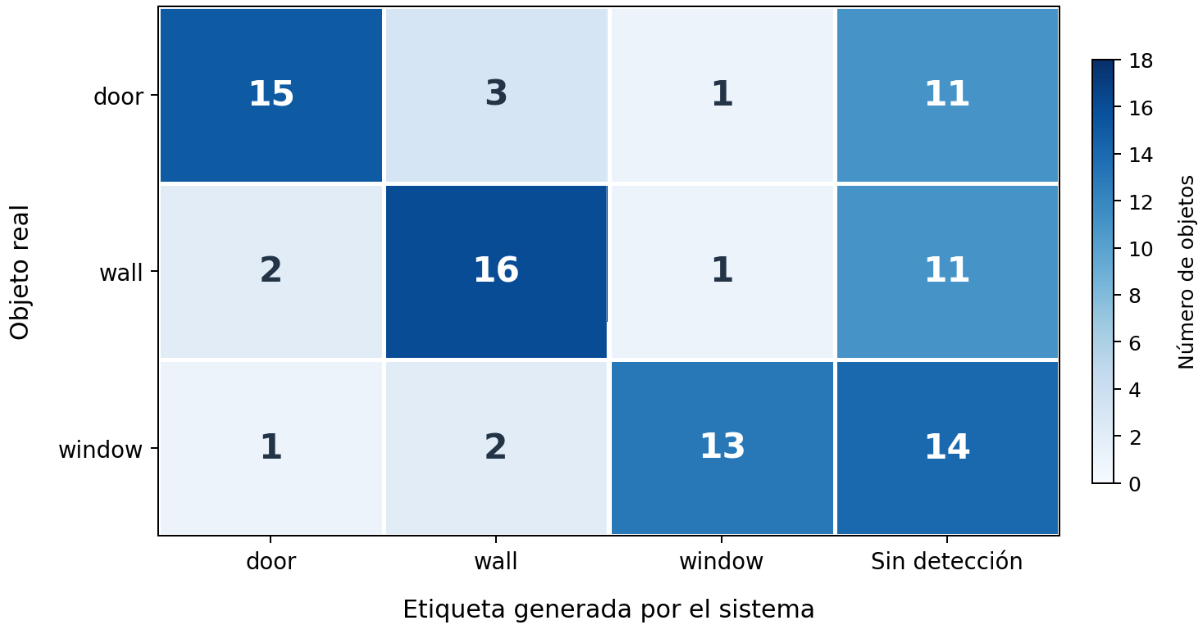


Fig. 4.24 Matriz de confusión del modelo personalizado

La matriz de confusión presentada en la Fig. 4.24 muestra un mayor acierto de clasificaciones correctas en la diagonal. Adicionalmente presenta 10 confusiones entre categorías y 36 elementos sin detección, acorde con la sensibilidad de 0.492 registrada en la validación del detector.

CAPITULO V: CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Se desarrolló un sistema de visión artificial embebido basado en aprendizaje profundo aplicado al robot móvil de alto torque, integrando la cámara Intel RealSense D415, la plataforma Jetson Orin Nano, ROS 2 y el sistema de control previamente realizado para el Arduino, el nuevo sistema permitió incorporar percepción RGB-D, detección de objetos así como una lógica de decisión capaz de intervenir físicamente sobre el movimiento del robot frente a la presencia de obstáculos manteniendo el sistema de control manual ya existente mediante una APK/Bluetooth.

El análisis de la arquitectura inicial permitió identificar restricciones de la plataforma Jetson Nano y la necesidad de disponer de mayores recursos de procesamiento para la ejecución de modelos de visión artificial, en base a este análisis se integró la Jetson Orin Nano como unidad principal de procesamiento y la Intel RealSense D415 como sensor RGB-D, concentrado en la plataforma embebida las tareas de adquisición, inferencia, decisión y comunicación directa con el Arduino.

Se diseñó un algoritmo de detección orientado a la prevención de colisiones mediante la combinación de un modelo base YOLOv8 y un modelo personalizado entrenado. La información visual fue complementada con datos de profundidad de la RealSense D415 y una lógica basada en distancia y posición horizontal de objetos, permitiendo generar estados de SEGURO, ALERTA, DETENER y maniobras de giro de acuerdo a las condiciones detectadas.

La implementación del sistema en la Jetson Orin Nano logró que la información obtenida del entorno se traduzca en acciones físicas de detección y giro frente a la presencia de obstáculos. Las pruebas realizadas permitieron comprobar el funcionamiento conjunto de la percepción y la lógica de decisión en armonía con la plataforma robótica previamente desarrollada.

5.2 Recomendaciones

Se recomienda ir ampliando progresivamente el conjunto de datos que se utilizan en el modelo personalizado, añadiendo una mayor variedad de escenas, perspectivas, diferentes iluminaciones y características de elementos estructurales, esto permitiría hacer un dataset más robusto sumando mayor capacidad al detector y mejorando las métricas obtenidas del modelo.

Es conveniente fortalecer o a su vez delimitar el comportamiento del sistema en distancias muy cortas, particularmente en el intervalo entre 300 y 400 mm, donde durante las pruebas prácticas se observó particular variación entre los estados DETENER y SEGURO, pudiendo establecer un estado de detención absoluta cuando estos estados tengan una alta frecuencia de cambios.

Es recomendable ampliar la validación del sistema mediante pruebas en diferentes escenarios y condiciones de operación incluyendo cambios en la ubicación, así como el tipo y cantidad de obstáculos, además de incorporar una medición instrumental del tiempo transcurrido entre la detección y la respuesta física del robot permitiendo complementar los resultados obtenidos para una mayor fidelidad de la precisión del robot.

5.3 Trabajo a futuro

Se propone de manera enfática continuar evolucionando el modelo de detección añadiendo nuevas clases y escenarios que amplíen la comprensión del entorno intentando especializarlo en el

ambiente de trabajo del robot, buscando que el sistema no se limite a los elementos utilizados en este desarrollo temprano.

Finalmente, la cámara RGB-D y la plataforma de procesamiento instaladas permiten proyectar nuevas aplicaciones dentro del campo de visión artificial sobre el mismo sistema, como reconocimiento de espacios, seguimiento de objetos o personas, análisis tridimensionales del entorno y asistencia de futuras tareas de navegación inteligente, ampliando las posibilidades de manera exponencial manteniendo la misma arquitectura actual.

Referencias

[1] E. S. Mediavilla Borja, "Robot móvil de alto torque: desarrollo electrónico," trabajo de titulación en Ing. Mecatrónica, Univ. Técnica del Norte, Ibarra, Ecuador, 2025. [Online]. Available: https://repositorio.utn.edu.ec/handle/123456789/17015

[2] P. S. Hinojosa Rojas, "Robot móvil de alto torque: visión artificial," trabajo de titulación en Ing. Mecatrónica, Univ. Técnica del Norte, Ibarra, Ecuador, 2025. [Online]. Available: https://repositorio.utn.edu.ec/handle/123456789/17002

[3] M. E. Castiblanco-Gutiérrez and C. A. Lozada-Ávila, "Diseño e implementación de un robot aspiradora utilizando algoritmos de visión computacional mediante redes neuronales," trabajo de grado, Univ. Católica de Colombia, Bogotá, Colombia, 2023. [Online]. Available: https://repository.ucatolica.edu.co/entities/publication/8db82220-7d01-4dc1-a530-92165abae3e4

[4] L. R. Bañuelos Hernández, “Aplicación de redes neuronales y visión artificial para la navegación autónoma de un robot móvil,” tesis de maestría en Sistemas Computacionales, Inst. Tecnológico de Chihuahua II, Tecnológico Nacional de México, Chihuahua, México, 2023. [Online]. Available: https://rinacional.tecnm.mx/handle/TecNM/8165

[5] A. M. Pardo Coronel and J. E. Vera Castro, “Diseño e implementación de un robot soccer autónomo mediante visión artificial a través del sistema embebido Raspberry Pi Zero,” trabajo de titulación en Ing. Electrónica, Univ. Politécnica Salesiana, Guayaquil, Ecuador, 2023. [Online]. Available: https://dspace.ups.edu.ec/handle/123456789/26387

[6] S. A. Doti, “Sistema de guiado de un robot móvil mediante visión artificial,” informe de práctica profesional supervisada, Univ. Nacional Arturo Jauretche, Florencio Varela, Argentina, 2021. [Online]. Available: https://rid.unaj.edu.ar/handle/123456789/2174

[7] R. K. Agudo Ube and F. P. Gómez Zambrano, “Diseño e implementación de robot móvil multitarea con visión artificial y programación en lenguaje Python para la Universidad Politécnica Salesiana,” trabajo de titulación, Univ. Politécnica Salesiana, Guayaquil, Ecuador, 2022. [Online]. Available: https://dspace.ups.edu.ec/handle/123456789/23566

[8] D. Barry, M. Shah, M. Keijsers, H. Khan, and B. Hopman, “xYOLO: A model for real-time object detection in humanoid soccer on low-end hardware,” 2019, arXiv:1910.03159, doi: 10.48550/arXiv.1910.03159.

[9] D. Ma, S. Li, B. Dang, H. Zang, and X. Dong, “Fostc3net: A lightweight YOLOv5 based on the network structure optimization,” 2024, arXiv:2403.13703, doi: 10.48550/arXiv.2403.13703.

[10] R. Farkh, M. T. Quasim, K. Al jaloud, S. Alhuwaimel, and S. T. Siddiqui, “Computer vision-control-based CNN-PID for mobile robot,” *Comput. Mater. Contin.*, vol. 68, no. 1, pp. 1065–1079, 2021, doi: 10.32604/cmc.2021.016600.

[11] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY, USA: Pearson, 2018.

[12] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2004, doi: 10.1017/CBO9780511811685.

[13] W. Roth et al., “Resource-efficient neural networks for embedded systems,” *J. Mach. Learn. Res.*, vol. 25, no. 50, pp. 1–51, 2024. [Online]. Available: https://www.jmlr.org/papers/v25/18-566.html

[14] Y. Deng, “Deep learning on mobile devices—A review,” 2019, arXiv:1904.09274, doi: 10.48550/arXiv.1904.09274.

[15] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, “Quantization and deployment of deep neural networks on microcontrollers,” *Sensors*, vol. 21, no. 9, Art. no. 2984, 2021, doi: 10.3390/s21092984.

[16] T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal, and G. Fettweis, “Latency analysis of ROS2 multi-node systems,” 2021, arXiv:2101.02074, doi: 10.48550/arXiv.2101.02074.

[17] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” *Sci. Robot.*, vol. 7, no. 66, Art. no. eabm6074, May 2022, doi: 10.1126/scirobotics.abm6074.

[18] Intel Corporation, *Intel RealSense Product Family D400 Series Datasheet*, rev. 016, document 337029-016, Jul. 2023. [Online]. Available: https://www.realsenseai.com/wp-content/uploads/2023/07/Intel-RealSense-D400-Series-Datasheet-July-2023.pdf

[19] NVIDIA Corporation. "NVIDIA Jetson Orin." NVIDIA. Accessed: Oct. 21, 2026. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

[20] OpenVINO Toolkit Contributors. "Monodepth estimation with OpenVINO." OpenVINO Documentation, ver. 2023.3. Accessed: Oct. 21, 2026. [Online]. Available: https://docs.openvino.ai/2023.3/notebooks/201-vision-monodepth-with-output.html

[21] RealSense. "ROS wrapper for RealSense cameras." GitHub. Accessed: Oct. 21, 2026. [Online]. Available: https://github.com/realsenseai/realsense-ros

[22] Ultralytics. "YOLOv5." GitHub. Accessed: Oct. 21, 2026. [Online]. Available: https://github.com/ultralytics/yolov5

[23] Open Robotics. "Topics." ROS 2 Documentation: Humble. Accessed: Oct. 21, 2026. [Online]. Available: https://docs.ros.org/en/humble/Concepts/Basic/About-Topics.html

[24] Ultralytics. "Explore Ultralytics YOLOv8." Ultralytics Docs. Accessed: Oct. 21, 2026. [Online]. Available: https://docs.ultralytics.com/models/yolov8/

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[26] Ultralytics. "Object detection datasets overview." Ultralytics Docs. Accessed: Oct. 21, 2026. [Online]. Available: https://docs.ultralytics.com/datasets/detect/

[27] Roboflow. “Introduction to Roboflow Annotate.” Roboflow Docs. Accessed: Oct. 21, 2026. [Online]. Available: https://docs.roboflow.com/datasets/annotate/annotate/annotation-tools

[28] A. Torralba and A. A. Efros, “Unbiased look at dataset bias,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Colorado Springs, CO, USA, 2011, pp. 1521–1528, doi: 10.1109/CVPR.2011.5995347.

ANEXOS

Anexo A.

Códigos y scripts disponibles en: https://github.com/Jjgranja/sistema-vision-artificial-robot-movil