



UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACSISTEL

INGENIERÍA EN ELECTRÓNICA Y
AUTOMATIZACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR

**Implementación de un robot móvil terrestre
para la evaluación del estado de cultivos,
utilizando navegación por LiDAR y
clasificación mediante Visión Artificial**

Kleyner Fabricio Flores Cedeño

Dirigido por:

Ing. Junior Figueroa Olmedo, Mgtr.

La Libertad - 2026

DEDICATORIA

A mis padres, Fabián Flores y Ruth Cedeño, por su respaldo constante y el esfuerzo dedicado a mi formación personal y universitaria. Su guía, comprensión y sacrificio en cada etapa de este camino han sido determinantes para alcanzar esta meta; este resultado refleja el fruto de su apoyo incondicional.

A mi pareja, por su compañía fiel a lo largo de este arduo proceso. Gracias por ser mi refugio en los momentos de agotamiento, por escucharme incluso cuando mis palabras estaban llenas de algoritmos y robótica, y por celebrar cada pequeño avance como si fuera un triunfo propio. Tu amor y paciencia me dieron la fuerza para no rendirme.

A mi familia entera, por ser mi inspiración constante y recordarme que el esfuerzo siempre rinde sus frutos.

Kleyner Fabricio Flores Cedeño.

AGRADECIMIENTO

Quiero expresar mi más sincero y profundo agradecimiento a la Universidad Estatal Península de Santa Elena (UPSE), en especial a la Facultad de Sistemas y Telecomunicaciones (FACSISTEL), por abrirme las puertas y brindarme los espacios académicos y tecnológicos necesarios para formar mi perfil profesional. Ha sido un honor formarme en las aulas de esta prestigiosa institución.

A mis docentes, por compartir su conocimiento con dedicación y exigencia. Gracias a su guía y experiencia, no solo aprendí la teoría, sino que desarrollé el pensamiento crítico y analítico indispensable para afrontar los desafíos de la ingeniería. Un agradecimiento especial a mi tutor, el Ing. Junior Figueroa Olmedo, Mgtr., por su dirección, paciencia y confianza depositada durante el desarrollo de este trabajo de titulación.

A mis compañeros y amigos de carrera, con quienes compartí largas jornadas de estudio, horas frente a la computadora, momentos de frustración y grandes éxitos. Gracias por el compañerismo, los debates académicos y por hacer de esta etapa universitaria una experiencia inolvidable.

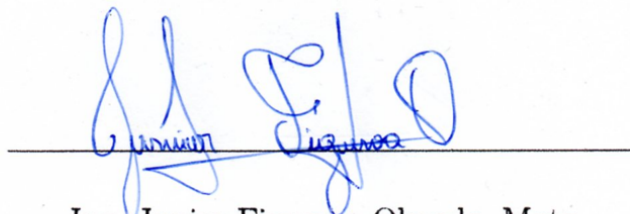
A mis padres y a mi pareja, por ser mi soporte emocional y económico. Gracias por creer en mí incluso cuando yo mismo dudaba, y por ser mi principal red de apoyo incondicional.

Finalmente, a la vida, por darme la salud, la perseverancia y la voluntad necesaria para alcanzar esta meta que hoy se convierte en una realidad.

Kleyner Fabricio Flores Cedeño.

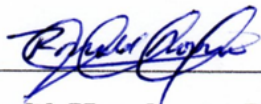
APROBACIÓN DEL TUTOR

En mi calidad de tutor del trabajo de titulación denominado: **“Implementación de un robot móvil terrestre para la evaluación del estado de cultivos, utilizando navegación por LiDAR y clasificación mediante visión artificial”**, elaborado por el estudiante **Kleyner Fabricio Flores Cedeño**, de la carrera de **Ingeniería en Electrónica y Automatización** de la Universidad Estatal Península de Santa Elena, declaro que luego de haber orientado, estudiado y revisado, lo apruebo en todas sus partes y autorizo al estudiante que inicie los trámites legales correspondientes.

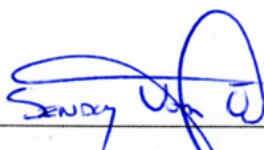


Ing. Junior Figueroa Olmedo, Mgtr.
Tutor

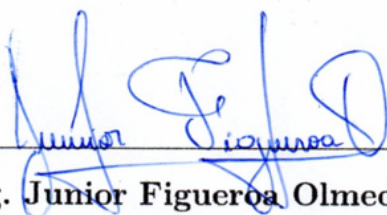
TRIBUNAL DE GRADO



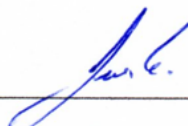
Ing. Ronald Humberto Rovira Jurado, Ph.D.
DIRECTOR DE CARRERA



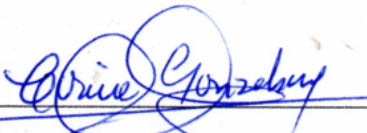
Ing. Sendey Vera González, Mgtr.
DOCENTE ESPECIALISTA



Ing. Junior Figueroa Olmedo, Mgtr.
TUTOR



Ing. Luis Chuquimarca Jiménez, Ph.D.
DOCENTE GUÍA UIC



Ing. Corina Gonzabay De La A, Mgtr.
SECRETARIA DEL TRIBUNAL

Resumen

Este trabajo presenta la implementación de un robot móvil terrestre para la evaluación automática del estado fitosanitario en cultivos de ciclo corto. Para superar las limitaciones del monitoreo manual de plagas, el sistema propuesto integra la plataforma Hiwonder JetTank ROS impulsada por una unidad de procesamiento NVIDIA Jetson Nano. La arquitectura de control opera mediante la separación funcional de dos capas: el sensor LiDAR ejecuta la navegación reactiva aplicando una ley de repulsión referenciada a los bordes de los surcos, mientras que la cámara RGB emplea el modelo YOLOv8 para contar y clasificar plantas en tiempo real al cruzar una línea virtual de validación. Debido a la pérdida de los especímenes vegetales orgánicos, la evaluación experimental en campo empleó réplicas sintéticas 3D dispuestas sobre el terreno. Las pruebas físicas confirmaron que el robot navega los surcos de forma autónoma y estable, procesando imágenes a una tasa de aproximadamente 25,7 FPS para la clasificación de las muestras. Los resultados demuestran que el sistema constituye un prototipo funcional con alto potencial para asistir el monitoreo manual en la agricultura.

Palabras clave: robótica móvil, agricultura de precisión, navegación reactiva, visión artificial, ROS.

Abstract

This work presents the implementation of a terrestrial mobile robot for automatic plant health assessment in short-cycle crops. To address the limitations of manual pest monitoring, the proposed system integrates the Hiwonder JetTank ROS platform powered by an NVIDIA Jetson Nano processing unit. The control architecture operates through the functional separation of two layers: the LiDAR sensor executes reactive navigation using a repulsion law referenced to the furrow edges, while the RGB camera employs the YOLOv8 model for real-time counting and classification as plants cross a virtual validation line. Due to the loss of organic crop specimens, the experimental field evaluation employed 3D synthetic plant replicas placed over the terrain. Field tests confirmed that the robot navigates the rows autonomously and stably, processing images at a rate of approximately 25.7 FPS to classify the synthetic samples. These results demonstrate that the system constitutes a functional prototype with high potential to assist manual monitoring in agriculture.

Keywords: mobile robotics, precision agriculture, reactive navigation, computer vision, ROS.

Índice

Índice de figuras	8
Índice de tablas	10
1. Introducción	14
1.1. Justificación	14
1.2. Panorama actual	15
1.3. Objetivos	16
1.3.1. Objetivo general	16
1.3.2. Objetivos específicos	16
1.4. Fundamentos teóricos	17
1.4.1. Cultivos de ciclo corto y sus plagas	17
1.4.2. Redes neuronales convolucionales	18
1.4.3. Robot móvil terrestre (diferencial)	18
1.4.4. Localización y mapeo en robótica móvil	19
1.5. Marco contextual	20
2. Métodos y diseño experimental	22
2.1. Métodos	22
2.2. Descripción general	24
2.3. Componentes físicos	26
2.3.1. Estructura, dimensiones y peso	26
2.3.2. Unidad de procesamiento embebido (NVIDIA Jetson Nano)	27
2.3.3. Sensor LiDAR (YDLIDAR G4)	28
2.3.4. Cámara de profundidad 3D	29
2.3.5. Sistema de locomoción: Encoders, servo y orugas	30
2.3.6. Concentrador (hub) USB	31
2.3.7. Tarjeta de expansión ROS (Expansion Board)	32
2.3.8. Batería de polímero de litio (LiPo)	33
2.4. Componentes lógicos	34
2.5. Diseño experimental	35
2.5.1. Diseño y caracterización cinemática de la plataforma	36
2.5.2. Diseño eléctrico	36
2.5.3. Diseño del sistema de control reactivo y supervisión de seguridad	36
2.5.4. Diseño del sistema de comunicación	41
2.5.5. Diseño de la interfaz gráfica de monitoreo	42

2.5.6.	Entrenamiento del modelo de visión artificial y optimización <i>edge computing</i>	42
2.6.	Desarrollo experimental	47
2.6.1.	Implementación del chasis robótico	47
2.6.2.	Configuración del entorno físico de pruebas	48
2.6.3.	Implementación del sistema de alimentación	50
2.6.4.	Implementación del sistema de control y mitigación de interferencia lumínica	51
2.6.5.	Implementación del sistema de comunicación	52
2.6.6.	Implementación de la interfaz gráfica embebida	53
2.6.7.	Configuración de parámetros cinemáticos	54
2.6.8.	Configuración del entorno de depuración y despliegue autónomo	59
2.6.9.	Mitigación de conflictos de red y optimización del enlace inalámbrico	60
2.6.10.	Configuración de la sensibilidad visual	60
3.	Resultados	62
3.1.	Requerimientos y restricciones del sistema	62
3.2.	Rendimiento de la telemetría y latencia	63
3.3.	Calibración de navegación LiDAR y compensación cinemática	64
3.4.	Métricas de evaluación del modelo de inferencia embebido	65
3.5.	Fusión de sensores y mitigación de interferencia solar	67
3.6.	Análisis de trayectorias y limitaciones físicas del terreno	67
3.7.	Despliegue de la interfaz embebida	70
4.	Conclusiones y recomendaciones	72
4.1.	Conclusiones	72
4.2.	Recomendaciones	74
	Referencias	76
	Anexos	79

Índice de figuras

1.	Ejemplo visual de plagas comunes en cultivos de ciclo corto. .	17
2.	Arquitectura general de una red neuronal convolucional (CNN). .	18
3.	Representación gráfica de la cinemática diferencial y postura del robot.	19
4.	Esquema conceptual del proceso del SLAM. Fuente: Elaboración propia.	20
5.	Esquema general de los componentes y fases operativas del sistema. Fuente: Autoría propia.	24
6.	Diagrama de los subsistemas mecánicos, de control, sensado y actuación del robot.	26
7.	Vista general de la estructura y chasis de aluminio.	27
8.	Microcomputador NVIDIA Jetson Nano.	28
9.	Vista del sensor LiDAR YDLIDAR G4 montado.	29
10.	Cámara de profundidad RGB-D instalada en el frontal.	30
11.	Clasificación de tipos de ruedas y sistemas de locomoción en robótica móvil.	31
12.	Detalle del motor 520, orugas y servomotor PTZ.	31
13.	Concentrador (hub) USB integrado en la plataforma.	32
14.	Tarjeta de expansión ROS y gestión de potencia.	33
15.	Paquete de batería LiPo de 6000 mAh y 11.1 V.	34
16.	Diagrama de bloques de la arquitectura de control con separación funcional.	41
17.	Estructura del dataset personalizado en la plataforma Roboflow. .	43
18.	Mosaico de lotes de entrenamiento (Train Batch) en YOLOv8. .	45
19.	Curvas de pérdida y precisión mAP@50 del entrenamiento YOLOv8n.	46
20.	Diagrama del flujo de optimización de PyTorch a TensorRT. .	46
21.	Vista superior del robot navegando autónomamente dentro de la hilera de cultivo. Fuente: Autoría propia.	49
22.	Vista a nivel del suelo evidenciando la estrechez del surco y el reto de navegación autónoma. Fuente: Autoría propia.	50
23.	Lanzadores de escritorio para el despliegue autónomo en la pantalla táctil de la Jetson Nano. Fuente: Autoría propia. . . .	60
24.	Matriz de confusión del modelo optimizado best.engine.	66
25.	Colapso del mapa SLAM por derrape de orugas.	68
26.	Trayectoria manual sin derrape.	69
27.	Trayectoria autónoma con elongación por deslizamiento.	70

28.	Interfaz gráfica de monitoreo embebida en la pantalla táctil de la Jetson Nano. Fuente: Autoría propia.	71
29.	Entorno de escritorio de la Jetson Nano con los lanzadores de autoarranque.	103
30.	Terminal del sistema confirmando la calibración de la IMU y el inicio del escaneo LiDAR.	104
31.	Carga del motor de inferencia TensorRT y despliegue de la GUI táctil previo al inicio del tránsito.	104
32.	Finalización de la misión en campo y presentación del balance fitosanitario en la interfaz.	105
33.	Código QR - Repositorio de GitHub.	106
34.	Código QR - Dataset de imágenes en Roboflow.	107
35.	Código QR - Carpeta de videos demostrativos en Google Drive.	108

Índice de tablas

I.	Especificaciones físicas del chasis y estructura.	27
II.	Especificaciones del microcomputador Jetson Nano.	28
III.	Especificaciones del sensor LiDAR YDLIDAR G4.	29
IV.	Especificaciones de la cámara de profundidad RGB-D.	30
V.	Especificaciones de los actuadores y tracción.	31
VI.	Especificaciones del concentrador de expansión de datos. . . .	32
VII.	Especificaciones de la tarjeta de expansión.	33
VIII.	Especificaciones del sistema energético.	34
IX.	Resumen consolidado de parámetros de configuración del sistema. Fuente: Elaboración propia.	55
X.	Requerimientos y restricciones de diseño del sistema	63
XI.	Comparativa de rendimiento de los protocolos de telemetría visual	64
XII.	Métricas de rendimiento de la red neuronal multiclase optimizada en la Jetson Nano	66

Nomenclatura y Abreviaturas

A continuación, se presentan las abreviaturas y los símbolos matemáticos utilizados en este trabajo de investigación:

Símbolos y variables matemáticas del sistema robótico

Símbolo	Significado
I	Corriente eléctrica
V	Voltaje o diferencia de potencial eléctrico
v	Velocidad lineal del centro de masa del robot
x, y	Coordenadas de posición bidimensional en el marco cartesiano global
ψ	Orientación azimutal (<i>yaw</i>) del chasis respecto al marco global
θ	Ángulo de actitud de cabeceo (<i>pitch</i>) medido por la IMU
ϕ	Ángulo de actitud de balanceo (<i>roll</i>) medido por la IMU
ω	Velocidad angular de rotación del vehículo
$\dot{\varphi}_i$	Velocidad angular de rotación de la rueda o banda motriz i

Acrónimos y abreviaturas tecnológicas (Parte I)

Sigla	Significado
CNN	Red neuronal convolucional (<i>Convolutional Neural Network</i>)
CPU	Unidad central de procesamiento (<i>Central Processing Unit</i>)
CSI	Interfaz serie de cámara (<i>Camera Serial Interface</i>)
DC	Corriente continua (<i>Direct Current</i>)
FOV	Campo de visión (<i>Field of View</i>)
FP32	Formato en coma flotante de precisión simple de 32 bits (<i>Floating Point 32-bit</i>)
FPS	Cuadros por segundo (<i>Frames Per Second</i>)
GPU	Unidad de procesamiento gráfico (<i>Graphics Processing Unit</i>)
GUI	Interfaz gráfica de usuario (<i>Graphical User Interface</i>)
HSV	Espacio de color matiz, saturación y valor (<i>Hue, Saturation, Value</i>)
I ² C	Circuito inter-integrado (<i>Inter-Integrated Circuit</i>)
IDE	Entorno de desarrollo integrado (<i>Integrated Development Environment</i>)
IMU	Unidad de medición inercial (<i>Inertial Measurement Unit</i>)
IP	Protocolo de Internet (<i>Internet Protocol</i>)
LiDAR	Detección y localización por luz (<i>Light Detection and Ranging</i>)
LiPo	Batería basada en polímero de litio (<i>Lithium Polymer</i>)
mAP	Precisión media promedio (<i>Mean Average Precision</i>)

Acrónimos y abreviaturas tecnológicas (Parte II)

Sigla	Significado
ONNX	Formato abierto de intercambio de redes neuronales (<i>Open Neural Network Exchange</i>)
PD	Regulador proporcional-derivativo
PID	Regulador proporcional-integral-derivativo
PTZ	Sistema de posicionamiento de paneo, inclinación y acercamiento (<i>Pan-Tilt-Zoom</i>)
PWM	Modulación por ancho de pulsos (<i>Pulse Width Modulation</i>)
RAM	Memoria de acceso aleatorio (<i>Random Access Memory</i>)
RGB	Espacio de color rojo, verde y azul (<i>Red, Green, Blue</i>)
RGB-D	Sensor visual de canales de color y mapa de profundidad (<i>Red, Green, Blue, Depth</i>)
ROS	Sistema operativo robótico (<i>Robot Operating System</i>)
SFTP	Protocolo seguro de transferencia de archivos (<i>Secure File Transfer Protocol</i>)
SLAM	Localización y mapeo simultáneos (<i>Simultaneous Localization and Mapping</i>)
SSH	Protocolo de intérprete de comandos seguro (<i>Secure Shell</i>)
ToF	Principio de medición de tiempo de vuelo (<i>Time of Flight</i>)
UART	Transmisor-receptor asíncrono universal (<i>Universal Asynchronous Receiver-Transmitter</i>)
WSL	Subsistema de Windows para Linux (<i>Windows Subsystem for Linux</i>)
YOLO	Modelo de detección en una sola etapa (<i>You Only Look Once</i>)

1. Introducción

El presente capítulo introduce el desarrollo e implementación de un sistema robótico basado en el Hiwonder JetTank ROS para la evaluación del estado fitosanitario de cultivos de ciclo corto. En las siguientes secciones se expone la justificación que respalda la relevancia técnica y la importancia práctica de automatizar el monitoreo agrícola frente a los desafíos actuales. Asimismo, se analiza el estado actual de las tecnologías de robótica y visión computacional aplicadas a la agricultura. Por otra parte, se definen los objetivos general y específicos que guían la investigación, se proporciona la fundamentación teórica necesaria para comprender los sistemas involucrados y, finalmente, se contextualiza el desarrollo del proyecto dentro de la realidad agrícola de la provincia de Santa Elena.

1.1. Justificación

El control fitosanitario constituye un desafío crítico para la producción agrícola: la proliferación incontrolada de plagas reduce los volúmenes de cosecha y deteriora la calidad del producto, generando pérdidas económicas severas en el sector [1]. Esta problemática se intensifica debido a la necesidad de incrementar la productividad agrícola ante el acelerado crecimiento demográfico global, sin comprometer la sostenibilidad del suelo.

Las metodologías tradicionales basadas en la inspección visual humana resultan ineficientes en campo. Su baja frecuencia operativa, la vulnerabilidad al error de apreciación y la imposibilidad de abarcar grandes extensiones derivan en diagnósticos tardíos. Esta limitación fomenta la aplicación excesiva y poco focalizada de insumos químicos, lo cual incrementa la degradación ambiental en los agroecosistemas [2].

Para mitigar estas deficiencias, la integración de sistemas robóticos móviles equipados con sensores LiDAR (Light Detection and Ranging) y algoritmos de visión artificial permite automatizar la inspección del cultivo. Esta arquitectura posibilita la detección temprana de patógenos en campo, facilita la aplicación localizada de tratamientos y optimiza el esfuerzo manual en las parcelas [3].

En Ecuador, los cultivos de ciclo corto (como arroz, maíz y hortalizas) sustentan la economía de las comunidades rurales. No obstante, las variaciones microclimáticas regionales favorecen la rápida propagación de agentes fitófagos, impactando directamente la rentabilidad de los pequeños y medianos productores. Por consiguiente, este trabajo propone el desarrollo de una plataforma robótica autónoma para la supervisión y mitigación de pérdidas fitosanitarias, impulsando la transición hacia la agricultura de precisión en el

país [4].

1.2. Panorama actual

El despliegue de la robótica y la inteligencia artificial en el sector agroindustrial cuenta con precedentes comerciales y académicos que respaldan la viabilidad de este tipo de proyectos. En la integración de visión por computadora y aprendizaje profundo, destaca el desarrollo de Blue River Technology (adquirida por John Deere en 2017) a través de su sistema “See and Spray”. Esta plataforma procesa imágenes mediante algoritmos avanzados para aplicar herbicidas exclusivamente sobre las malezas identificadas, reduciendo el consumo de insumos químicos [5].

En el ámbito de la navegación autónoma y el control cinemático en campo, firmas como Naio Technologies comercializan el robot Oz para el deshierbe de hortalizas y viveros [6]; por su parte, FarmWise despliega unidades autónomas capaces de diferenciar cultivos de malezas mediante visión artificial para ejecutar su remoción física [7]. Adicionalmente, existen enfoques complementarios orientados hacia la recolección automatizada mediante visión 3D, como el sistema desarrollado por Abundant Robotics [8], y hacia la teledetección fitosanitaria con imágenes aéreas multiespectrales, como la propuesta por Taranis [9].

Desde la perspectiva de las plataformas de prueba e investigación técnica, el Husky UGV de Clearpath Robotics representa un estándar ampliamente documentado en la literatura y sirve como marco de referencia para la unidad Hiwonder JetTank ROS empleada en este trabajo. Pese a tratarse de un vehículo de mayor escala y potencia, comparte fundamentos conceptuales relevantes, como el uso de ROS (Robot Operating System) como middleware y la implementación de estrategias de navegación reactiva basadas en LiDAR para el desplazamiento en terrenos irregulares [10]. Asimismo, plataformas cuadrúpedas como Spot de Boston Dynamics han comenzado a implementarse en la supervisión de cultivos y el sector ganadero, lo que evidencia la flexibilidad de los sistemas autónomos en entornos agrícolas heterogéneos [11].

1.3. Objetivos

1.3.1. Objetivo general

Implementar un sistema autónomo de supervisión fitosanitaria sobre la plataforma robótica Hiwonder JetTank ROS, mediante navegación reactiva basada en LiDAR y visión artificial con YOLOv8 en procesamiento embebido, para la clasificación y el conteo de plantas durante el recorrido en cultivos de ciclo corto.

1.3.2. Objetivos específicos

- Implementar un sistema de visión artificial basado en aprendizaje profundo (YOLOv8) optimizado para la plataforma embebida, capaz de clasificar el estado fitosanitario (sano, plaga y estrés) y contar las plantas detectadas durante el recorrido.
- Desarrollar un algoritmo de navegación reactiva basado en LiDAR, con supervisión de actitud mediante IMU y apoyo de odometría para las maniobras de cambio de hilera, procurando el recorrido centrado entre surcos y la detención segura ante bloqueos.
- Integrar la cámara RGB, el LiDAR y la IMU en una arquitectura modular con separación funcional de responsabilidades, favoreciendo la operación continua de la navegación y la percepción visual en la tarjeta embebida.
- Desarrollar una interfaz gráfica de usuario embebida para la visualización en tiempo real del estado operacional del robot, así como del conteo y la clasificación fitosanitaria.

1.4. Fundamentos teóricos

La base teórica de este proyecto se sustenta en tres áreas: la agronomía, la visión por computadora y la robótica móvil. La convergencia de estas disciplinas articula los requerimientos fitotécnicos con sistemas autónomos de inspección en campo. La plataforma robótica gestiona la locomoción continua a lo largo de los surcos, mientras que el módulo de visión computacional analiza las características morfológicas del dosel foliar en tiempo real para identificar estados de estrés o proliferación de plagas. A continuación, se detallan los principios operativos de cada componente y su integración funcional en el sistema.

1.4.1. Cultivos de ciclo corto y sus plagas

Los cultivos de ciclo corto se caracterizan por presentar un ciclo biológico acelerado, lo que permite cosechas en períodos relativamente breves, generalmente inferiores a un año. Especies como la lechuga o el maíz son fundamentales para la producción rápida de alimentos; no obstante, presentan una elevada susceptibilidad a factores bióticos y abióticos [12]. Las plagas representan una amenaza significativa: insectos como los pulgones [13], la mosca blanca [14], los trips [15] y los minadores de hojas [16] debilitan las plantas al succionar su savia, afectar su desarrollo fotosintético o actuar como vectores de enfermedades virales, reduciendo así la calidad y la cantidad de los cultivos. Para ilustrar la vulnerabilidad de estas plantaciones y proporcionar contexto visual sobre las amenazas a detectar, la Fig. 1 muestra ejemplos de las principales plagas que afectan el desarrollo de las hojas.



Fig. 1. Ejemplo visual de plagas comunes en cultivos de ciclo corto. Fuente: [13–16].

1.4.2. Redes neuronales convolucionales

Para abordar esta problemática de manera automatizada, se emplea la visión artificial [17], la cual permite la identificación y clasificación temprana de estas amenazas [18]. En este ámbito destacan las redes neuronales convolucionales (CNN), un modelo de aprendizaje profundo diseñado para procesar datos en forma de cuadrícula, como las imágenes. Las CNN identifican patrones y características jerárquicas (bordes, texturas o colores) de manera autónoma, lo que permite a los sistemas clasificar el estado de las plantas (sanas, con plagas o con estrés) basándose en las capturas visuales del entorno agrícola [19]. Para facilitar la comprensión de cómo se procesa la información visual capa por capa, la Fig. 2 detalla la estructura lógica de este modelo.

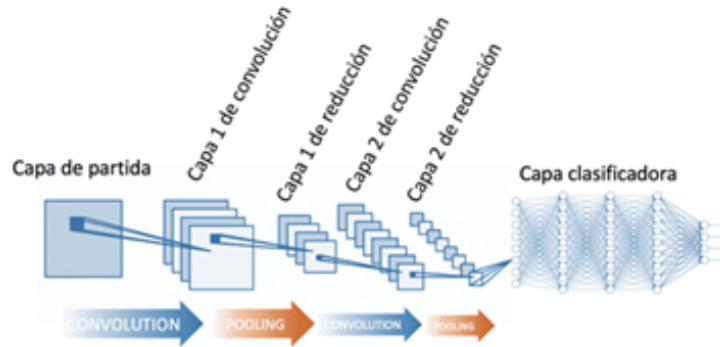


Fig. 2. Arquitectura general de una red neuronal convolucional (CNN) para procesamiento de imágenes. Fuente: [20].

1.4.3. Robot móvil terrestre (diferencial)

Este sistema de análisis visual se monta sobre un robot móvil terrestre, el cual es una máquina electromecánica capaz de desplazarse y reaccionar ante su entorno usando sensores y actuadores. El proyecto emplea una configuración de tracción diferencial tipo oruga. En este diseño, la velocidad y la dirección del chasis se obtienen variando la velocidad de giro de los motores laterales de forma independiente. El cálculo de este movimiento se basa en la cinemática diferencial, que relaciona las velocidades de giro de las orugas con la posición y la orientación del robot en el terreno [21]. En la Fig. 3 se muestra el modelo geométrico y las variables de esta locomoción, los cuales sirven de base para las ecuaciones de control utilizadas en el sistema.

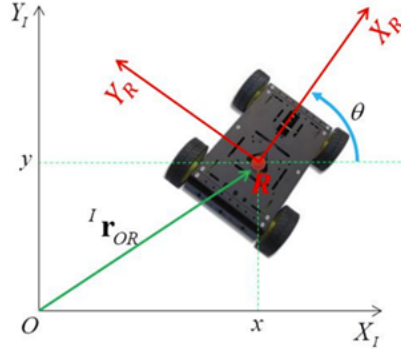


Fig. 3. Representación gráfica de la cinemática diferencial y postura del robot. Fuente: [22].

1.4.4. Localización y mapeo en robótica móvil

Para el levantamiento topográfico y la representación espacial del entorno agrícola, diversas investigaciones emplean técnicas como SLAM (*Simultaneous Localization and Mapping*). Este algoritmo procesa los datos capturados por el sensor LiDAR para construir un mapa de ocupación del terreno no estructurado y, simultáneamente, estimar la postura del robot dentro de dicho mapa. En este proyecto, el recorrido autónomo entre los surcos se ejecuta mediante un algoritmo de navegación reactiva basado en LiDAR, con supervisión de actitud mediante IMU (Inertial Measurement Unit) y apoyo de odometría para las maniobras de cambio de hilera. La técnica SLAM, por su parte, resulta fundamental en la fase de exploración previa, validación geométrica y cartografía del cultivo. Diversas investigaciones sobre SLAM y robótica móvil resaltan la importancia de combinar percepción, localización y mapeo para operar en entornos agrícolas no estructurados [23]. Para una mejor comprensión del flujo de datos en el proceso de mapeo, la Fig. 4 detalla el esquema conceptual correspondiente.

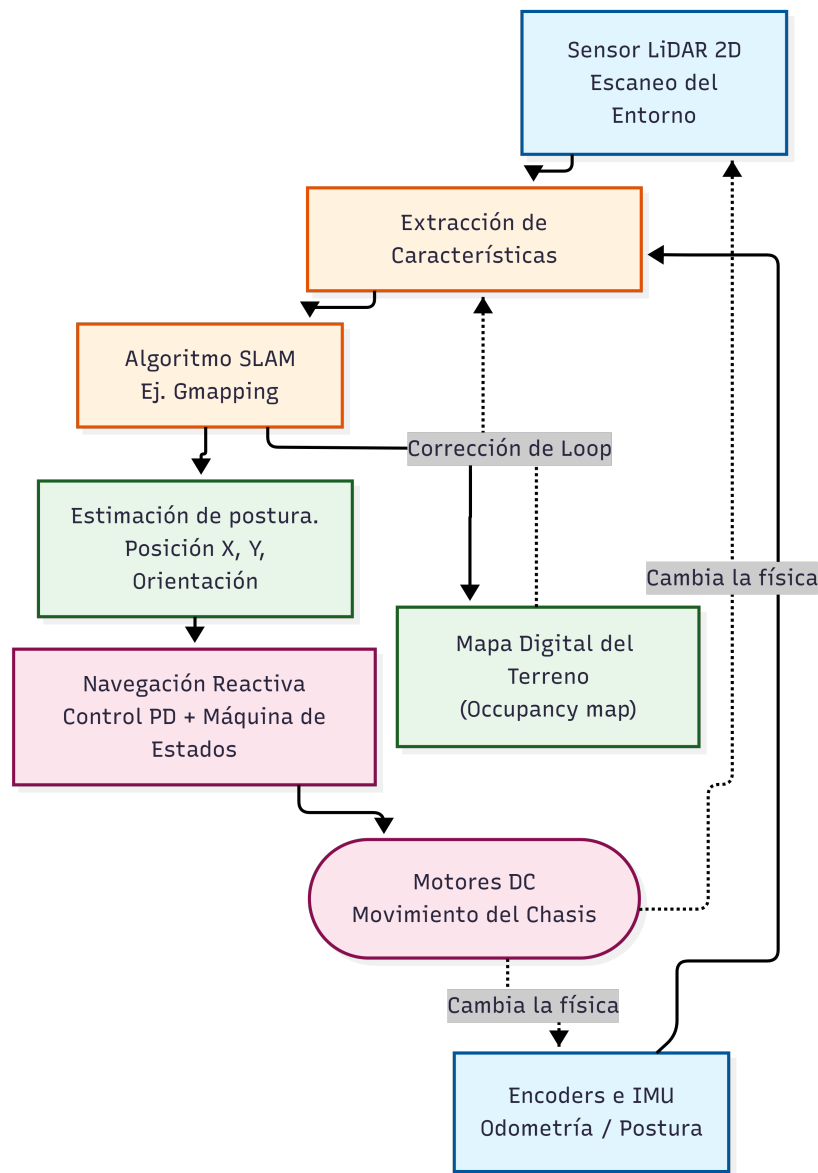


Fig. 4. Esquema conceptual del proceso del SLAM. Fuente: Elaboración propia.

1.5. Marco contextual

Ubicada en la franja costera de Ecuador, la provincia de Santa Elena sustenta su dinamismo económico en el turismo, la pesca y una actividad agrícola en progresiva expansión. Aunque la región destaca por la producción rural y periurbana de maíz, mango, uvas y hortalizas, los cultivos de ciclo

corto —especialmente la lechuga— han adquirido relevancia comercial. A pesar de tratarse de una especie adaptada originalmente a climas fríos, su producción se ha establecido en el entorno semiárido peninsular mediante invernaderos y sistemas hidropónicos para abastecer el mercado local [24]. No obstante, la consolidación del sector enfrenta barreras asociadas con la baja adopción tecnológica, infraestructuras limitadas y la incidencia recurrente de fitopatógenos [25].

El impacto de las plagas en la productividad agrícola constituye un factor crítico a escala global y regional, calculándose reducciones de rendimiento de entre el 20 % y el 40 % anual por la acción de insectos y patógenos, cifra con repercusión directa en los ingresos de los productores de la zona [26]. Frente a la escasez de herramientas para la supervisión continua en campo, el procesamiento digital de imágenes y el aprendizaje automático ofrecen un marco técnico efectivo para reconocer patrones morfológicos anómalos de forma anticipada [27].

La integración de la navegación reactiva basada en LiDAR y la visión computacional plantea un cambio de paradigma para la inspección agrícola local. El despliegue de una plataforma robótica móvil orientada a este fin busca automatizar la detección de patologías y optimizar la logística en parcela, disminuyendo la aplicación indiscriminada de agroquímicos y mitigando la carga ambiental [28]. Asimismo, la transición hacia esquemas de monitoreo automatizado promueve la rentabilidad y la sostenibilidad operativa en el agro peninsular [29].

En este escenario, el presente trabajo articula una propuesta tecnológica adaptada a las condiciones productivas de Santa Elena. Al enmarcarse en los principios de la Agricultura 4.0 y la digitalización del campo, el desarrollo busca fortalecer la capacidad técnica de los productores locales e impulsar la innovación tecnológica en la región [30].

2. Métodos y diseño experimental

2.1. Métodos

La metodología aplicada corresponde a un enfoque de desarrollo tecnológico experimental, centrado en el diseño de sistemas embebidos, la validación en entornos controlados y la evaluación cuantitativa del desempeño. El proceso metodológico se estructuró en los siguientes ejes operativos:

- **Diseño e integración de hardware/software:** Consistió en la adaptación funcional de la plataforma Hiwonder JetTank. En esta etapa se realizó la interconexión física de sensores y actuadores con la unidad de procesamiento, así como la configuración de los nodos y scripts de ROS para la gestión de comunicaciones, adquisición sensorial y control del vehículo.
- **Validación individual de sensores y actuadores:** Previo a la implementación del sistema integrado, se efectuaron pruebas independientes del LiDAR, la cámara, la IMU, los encoders, los motores de tracción y el servomotor PTZ (Pan-Tilt-Zoom). Esta etapa permitió verificar el funcionamiento, la estabilidad y la compatibilidad de cada dispositivo con la plataforma ROS, además de calibrar parámetros básicos antes de integrar la navegación reactiva y el módulo de visión artificial.
- **Ensayos experimentales en campo:** Se ejecutaron pruebas físicas recorriendo surcos que replican hileras de cultivos de ciclo corto. Durante los ensayos se evaluó la estabilidad de la navegación, la respuesta del robot ante variaciones del entorno y la capacidad del sistema visual para detectar, clasificar y contar plantas durante el recorrido.
- **Evaluación cuantitativa del desempeño:** Se procesaron los datos obtenidos por el módulo de visión artificial. La confiabilidad del modelo se evaluó mediante métricas de detección y clasificación, tales como precisión, sensibilidad, mAP@50 y latencia de inferencia. Adicionalmente, se verificó la ausencia de conteos duplicados durante las pruebas de campo.

Variables del diseño experimental

Las pruebas operativas se configuraron bajo el control y monitoreo de los siguientes parámetros:

- *Variables independientes:* Configuración cinemática del robot (velocidad lineal de cruce y límites de velocidad angular), parámetros de la ley de control reactivo, resolución de entrada de la cámara, umbral de confianza del modelo de visión y parámetros de configuración del motor de inferencia embebido.
- *Variables dependientes:* Tasa de procesamiento visual (FPS y latencia), precisión, sensibilidad y mAP@50 del clasificador, número de conteos duplicados, estabilidad del recorrido y consistencia de la cartografía SLAM evaluada en la fase exploratoria.

Recopilación de datos y fuentes de error

La adquisición de datos se realizó durante los ensayos mediante los sensores del robot. La cámara registró imágenes del área foliar, mientras que el LiDAR proporcionó mediciones de distancia para la navegación y la caracterización del entorno del surco. Durante las pruebas se identificaron tres principales fuentes de error:

- *Deslizamiento de las orugas:* Al desplazarse sobre tierra suelta, las orugas pueden patinar. Este fenómeno afecta la estimación odométrica y la consistencia del mapa SLAM; por ello, la navegación final se apoyó principalmente en la información del LiDAR.
- *Variabilidad en la radiación solar:* Los cambios bruscos de iluminación modifican la apariencia visual del follaje y pueden dificultar la clasificación. Para mitigar este efecto, el dataset incluyó capturas bajo distintas condiciones de luz y se ajustó un umbral de confianza para la detección.
- *Inestabilidad en la alimentación eléctrica:* Los picos de consumo de los motores pueden generar caídas de tensión. Para reducir este riesgo, se separó la alimentación de la unidad de procesamiento de la etapa de potencia.

Instrumentos, equipos y software

Para el desarrollo de la investigación se empleó la siguiente infraestructura técnica:

- *Hardware:* Plataforma robótica Hiwonder JetTank con NVIDIA Jetson Nano, sensor LiDAR 2D, cámara frontal (hardware RGB-D, utilizando finalmente el canal RGB para inferencia), motores DC con encoders magnéticos de efecto Hall, IMU integrada en la tarjeta de expansión y servomotor PTZ para la cámara.

- *Software y entorno de desarrollo:* Sistema operativo Ubuntu, ROS como middleware de comunicación, Python 3, OpenCV, NumPy, TensorRT para inferencia embebida y Tkinter para la interfaz gráfica. El desarrollo se apoyó en WSL y conexiones SSH para la edición y depuración remota, mientras que el despliegue final se ejecutó localmente en la Jetson Nano.

2.2. Descripción general

El prototipo desarrollado consiste en un robot móvil terrestre orientado a la inspección fitosanitaria de cultivos de ciclo corto. El sistema articula hardware robótico, procesamiento de visión computacional y una estrategia de navegación reactiva basada en el sensor LiDAR, con el objetivo de lograr desplazamiento autónomo sobre surcos agrícolas con mínima intervención humana. Adicionalmente, se empleó SLAM como herramienta exploratoria de cartografía y validación del hardware, aunque no como método principal de navegación.

La plataforma base es la Hiwonder JetTank ROS, adaptada técnica y funcionalmente para los ensayos experimentales. La dinámica operativa se organiza alrededor de un flujo de datos estructurado en tres bloques funcionales: adquisición sensorial, procesamiento central y control cinemático con visualización del estado. La interacción entre las etapas de hardware y software en la parcela se esquematiza en la Fig. 5.

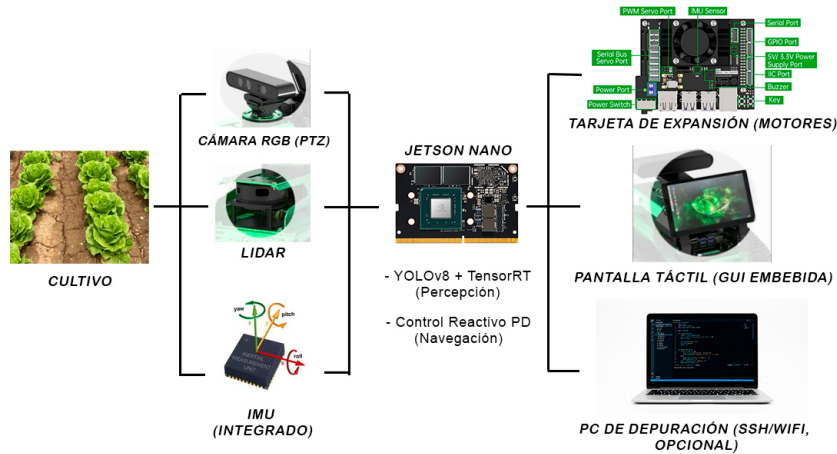


Fig. 5. Esquema general de los componentes y fases operativas del sistema. Fuente: Autoría propia.

- **Adquisición de datos sensoriales:** La captura de información del entorno se ejecuta mediante sensores exteroceptivos. Un sensor LiDAR

2D barre el entorno para medir distancias y detectar los límites del surco. En la etapa exploratoria, estos datos también se utilizaron para construir mapas de ocupación mediante SLAM. En paralelo, la cámara frontal captura imágenes RGB del cultivo. Aunque el hardware corresponde a una cámara RGB-D, el canal de profundidad no se empleó en la etapa final de inferencia debido a la saturación causada por la radiación solar directa.

- **Procesamiento central de datos:** La totalidad de las lecturas converge en la unidad embebida NVIDIA Jetson Nano. Durante esta fase, las imágenes capturadas en formato BGR se redimensionan, se convierten a RGB y se normalizan mediante arreglos de NumPy antes de la inferencia. Posteriormente, el modelo de visión artificial procesa cada imagen para clasificar las plantas detectadas en tres categorías: sana, plaga y estrés.
- **Control cinemático y visualización del estado:** El sistema opera con separación funcional de responsabilidades. La dirección del vehículo se gobierna mediante una *ley de repulsión* reactiva calculada a partir de las distancias laterales medidas por el LiDAR, con el fin de mantener el robot centrado dentro del surco. La percepción visual, por su parte, se emplea para clasificar y contar plantas durante el avance, sin intervenir directamente en la ley de control de trayectoria. Para reducir falsos conteos por objetos lejanos o vegetación fuera del surco, únicamente se validan las detecciones cuya coordenada inferior se encuentra en la mitad inferior de la imagen. Además, se aplica un periodo de inactivación temporal entre conteos consecutivos. La función que activa el contador C se define mediante la siguiente relación:

$$C = \begin{cases} 1, & \text{si } y_{\text{bottom}} > Y_{\text{lim}} \wedge \Delta t_{\text{last}} > T_{\text{cd}} \\ 0, & \text{en caso contrario,} \end{cases} \quad (1)$$

donde Y_{lim} corresponde al umbral espacial de validación en la imagen, y_{bottom} representa la coordenada inferior del objeto detectado, Δt_{last} es el tiempo transcurrido desde el evento anterior y T_{cd} es el periodo de inactivación temporal configurado en 4 s para las pruebas finales.. Si en un mismo cuadro existen varias detecciones, el contador considera únicamente la detección válida de mayor puntuación; las detecciones restantes se descartan para evitar múltiples registros simultáneos. Esta formulación permite contabilizar la incidencia fitosanitaria sin interferir en la trayectoria del robot y reduce el riesgo de sobreconteo.

Esta separación funcional entre navegación y percepción permite que el robot mantenga una operación estable y continua durante los recorridos, consolidando el esquema operativo detallado en las siguientes secciones.

2.3. Componentes físicos

El hardware del sistema está conformado por la plataforma robótica móvil Hiwonder JetTank, la cual ha sido seleccionada por su arquitectura modular y su capacidad de procesamiento embebido, ideal para tareas de agricultura de precisión. Los componentes físicos se dividen funcionalmente en diversos subsistemas. Para proporcionar una representación visual estructurada de cómo se distribuyen e integran estos elementos en la plataforma, la Fig. 6 ilustra el diagrama general de la arquitectura de hardware del robot.

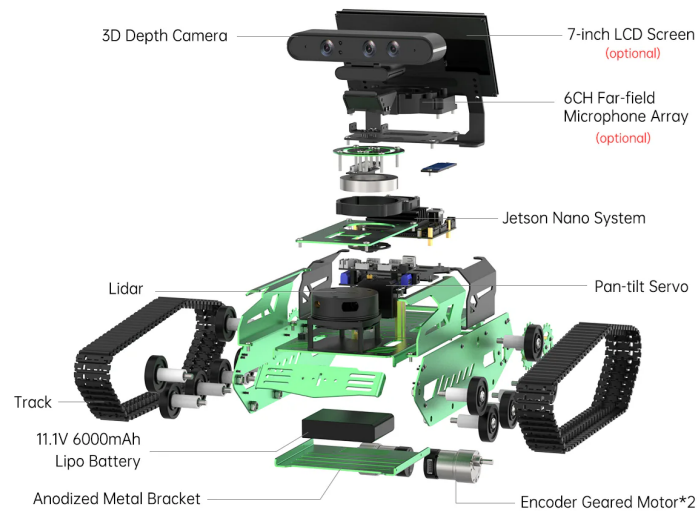


Fig. 6. Diagrama de los subsistemas mecánicos, de control, sensado y actuación del robot. Fuente: [31].

A continuación, se detallan las características operativas y especificaciones técnicas de todos los elementos mecánicos y electrónicos del robot, desglosados según su función en el sistema:

2.3.1. Estructura, dimensiones y peso

El chasis es el esqueleto de la plataforma móvil, diseñado para soportar las vibraciones del terreno agrícola y alojar de manera segura la carga útil (batería, sensores y procesador). Está fabricado con materiales industriales de

alta resistencia para garantizar la durabilidad ante condiciones ambientales moderadas, como se puede apreciar en la Fig. 7 y en la Tabla I.

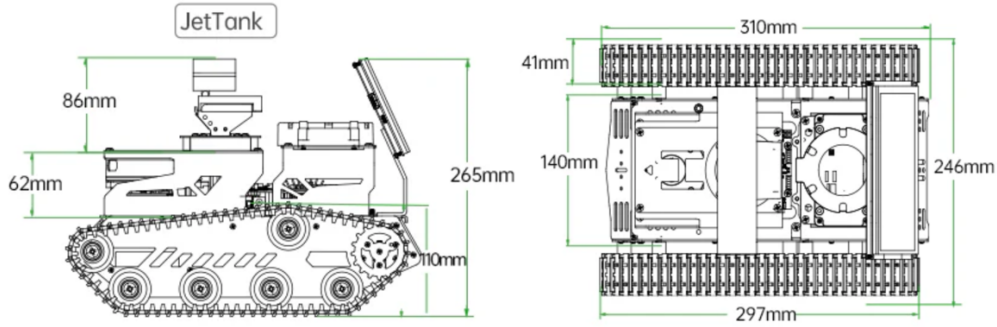


Fig. 7. Vista general de la estructura y chasis de aluminio. Fuente: [31].

Tabla I. Especificaciones físicas del chasis y estructura. Fuente: [31].

Parámetro	Especificación
Material principal	Aleación de aluminio duro (superficie anodizada)
Dimensiones (largo x ancho x alto)	$310 \times 246 \times 265$ mm
Peso neto del producto	2.9 kg
Capacidad de carga útil	Optimizado para sensores y electrónica a bordo

2.3.2. Unidad de procesamiento embebido (NVIDIA Jetson Nano)

Este módulo constituye el núcleo de cómputo principal del sistema, encargado de coordinar la arquitectura lógica y el procesamiento central. En la configuración *Advanced Kit*, este procesador embebido aloja el sistema operativo Ubuntu, gestiona el *middleware* ROS y ejecuta las redes neuronales de aprendizaje profundo para la inferencia visual. En la Fig. 8 se ilustra la disposición física de la tarjeta, mientras que la Tabla II resume sus especificaciones técnicas principales.

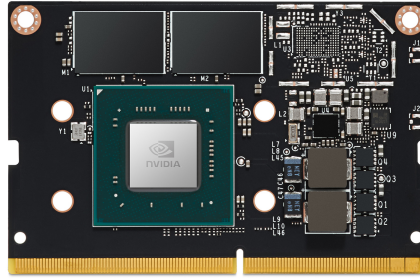


Fig. 8. Microcomputador NVIDIA Jetson Nano. Fuente: [32].

Tabla II. Especificaciones del microcomputador Jetson Nano. Fuente: [32].

Parámetro	Especificación
GPU (Gráficos)	Arquitectura Maxwell de 128 núcleos
CPU (Procesador)	ARM Cortex-A57 Quad-core
Memoria RAM	4 GB LPDDR4
Almacenamiento base	Tarjeta TF / MicroSD de 32 GB

2.3.3. Sensor LiDAR (YDLIDAR G4)

El sensor LiDAR es el componente principal del subsistema exteroceptivo. Su función es emitir un láser infrarrojo que rebota en los obstáculos y cultivos, calculando el tiempo de vuelo (ToF) para generar un mapa bidimensional del entorno. Cabe destacar que, aunque el sensor posee un rango de escaneo nativo de 360 grados, su campo de visión operativo en la plataforma se encuentra limitado físicamente a 180 grados debido a la oclusión generada por la estructura del chasis. En la Fig. 9 se puede apreciar el sensor montado y en la Tabla III sus especificaciones.

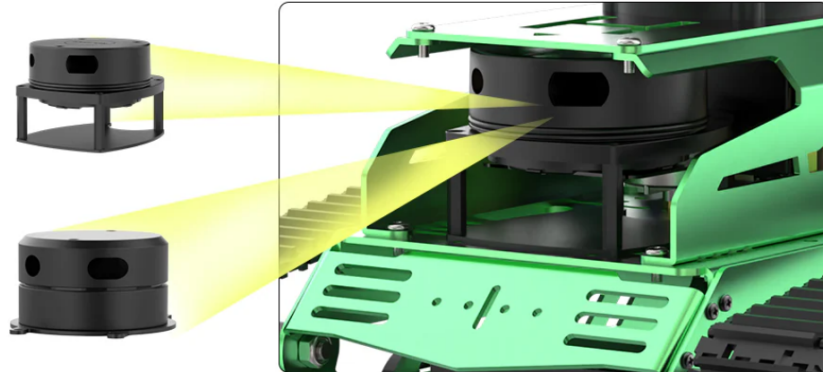


Fig. 9. Vista del sensor LiDAR YDLIDAR G4 montado. Fuente: [31].

Tabla III. Especificaciones del sensor LiDAR YDLIDAR G4. Fuente: [31].

Parámetro	Especificación
Modelo	EAI YDLIDAR G4
Rango de escaneo	360° (limitado a 180° por chasis)
Distancia máxima de medición	Hasta 16 metros
Frecuencia de escaneo	5 – 12 Hz
Frecuencia de muestreo	9000 muestras por segundo
Aplicación principal	Localización, mapeo (SLAM) y evasión de obstáculos

2.3.4. Cámara de profundidad 3D

Situada en la parte frontal superior del robot, esta cámara de alta definición constituye el sensor visual del modelo de inteligencia artificial. Captura el flujo de video necesario para procesar las características morfológicas de las hojas y diagnosticar plagas. En la Fig. 10 se ilustra el montaje frontal de la cámara, mientras que sus especificaciones técnicas se detallan en la Tabla IV.

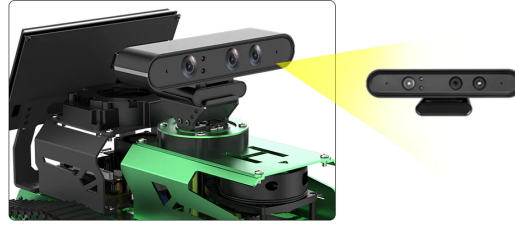


Fig. 10. Cámara de profundidad RGB-D instalada en el frontal. Fuente: [31].

Tabla IV. Especificaciones de la cámara de profundidad RGB-D. Fuente: [31].

Parámetro	Especificación
Tipo de sensor	Cámara RGB-D Astra Pro Plus
Resolución máxima (hardware)	1920 × 1080 (1080p)
Resolución operativa de captura	640 × 480 (VGA), configurada en el script de inferencia para reducir latencia en la Jetson Nano
Campo de visión (FOV)	60° H / 49° V / 73° D
Función en el sistema	Adquisición de imágenes para la red neuronal convolucional (CNN)

2.3.5. Sistema de locomoción: Encoders, servo y orugas

La movilidad diferencial del chasis se logra mediante actuadores de corriente continua acoplados al mecanismo de tracción. Internamente cuentan con *encoders* magnéticos para el registro odométrico. Adicionalmente, se emplea un servomotor de alto torque para controlar el giro (Pan-Tilt) del soporte de la cámara.

Para justificar la selección de este tipo de tracción frente a otras alternativas cinemáticas, en la Fig. 11 se presentan las configuraciones de ruedas convencionales. A diferencia de las configuraciones con ruedas orientables u omnidireccionales, el sistema de orugas maximiza el área de contacto, previniendo el hundimiento o deslizamiento en terrenos agrícolas con tierra suelta, tal como se puede observar en la Fig. 12 y en la Tabla V.



Fig. 11. Clasificación de tipos de ruedas y sistemas de locomoción en robótica móvil. Fuente: [22].

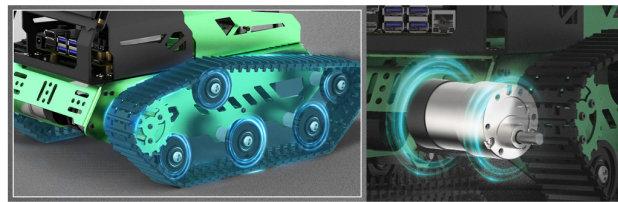


Fig. 12. Detalle del motor 520, orugas y servomotor PTZ. Fuente: [31].

Tabla V. Especificaciones de los actuadores y tracción. Fuente: [31].

Parámetro	Especificación
Motores de tracción	2 × Motores de engranajes DC (tipo 520)
Sensores de giro	<i>Encoders</i> magnéticos de alta precisión (efecto Hall)
Servomotor (cámara PTZ)	Bus serie inteligente HTS-20H (rango de rotación de 240°)
Sistema de ruedas	Orugas continuas de nylon con rodamientos metálicos

2.3.6. Concentrador (hub) USB

El procesamiento robótico complejo requiere manejar un alto volumen de información simultánea. El robot integra un hub USB para expandir la capacidad de conexión de la Jetson Nano, permitiendo enlazar múltiples periféricos de alta tasa de datos sin saturar el bus de comunicaciones. La configuración del concentrador y sus parámetros se presentan en la Fig. 13 y en la Tabla VI.

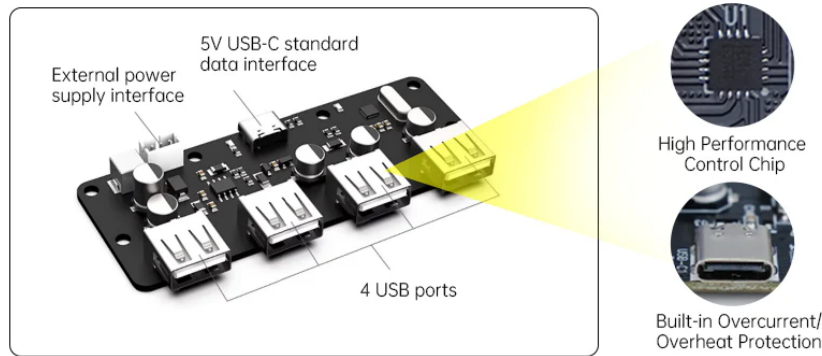


Fig. 13. Concentrador (hub) USB integrado en la plataforma. Fuente: [31].

Tabla VI. Especificaciones del concentrador de expansión de datos. Fuente: [31].

Parámetro	Especificación
Protocolo	USB 3.0 / 2.0 de alta velocidad
Periféricos conectados	Sensor LiDAR YDLIDAR G4, cámara RGB-D, tarjeta de expansión
Propósito	Garantizar transmisión fluida de video y nube de puntos al procesador

2.3.7. Tarjeta de expansión ROS (Expansion Board)

La placa de expansión actúa como la médula espinal que conecta la computadora lógica (Jetson Nano) con el hardware analógico y de potencia. Gestiona la señal de los sensores y decodifica las instrucciones de movimiento en energía real para los actuadores. La disposición física de la tarjeta de expansión y sus especificaciones operativas se presentan en la Fig. 14 y en la Tabla VII respectivamente.

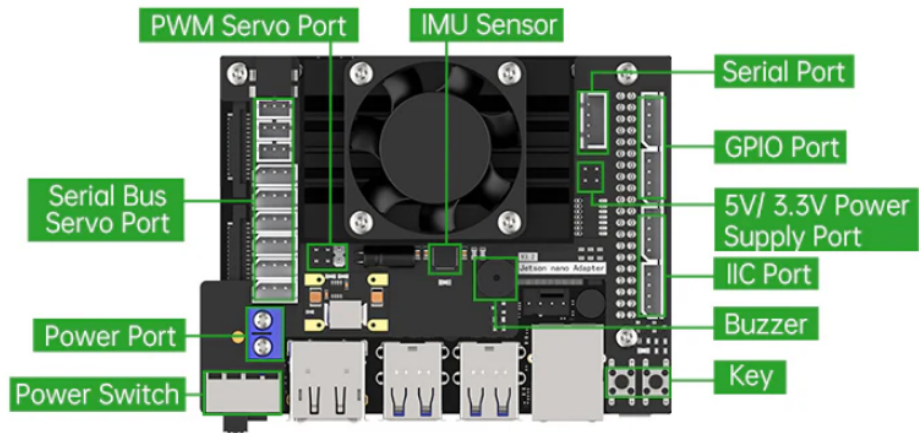


Fig. 14. Tarjeta de expansión ROS y gestión de potencia. Fuente: [31].

Tabla VII. Especificaciones de la tarjeta de expansión. Fuente: [31].

Parámetro	Especificación
Funciones de control	Gestión de motores DC, control de servomotores serie
Sensores integrados	Unidad de medición inercial (IMU) para detectar postura espacial
Comunicación	Puerto serial UART / I2C para interacción con Jetson Nano
Protección eléctrica	Regulación de voltaje de la batería para micro-controladores

2.3.8. Batería de polímero de litio (LiPo)

Para garantizar un desempeño ininterrumpido sin cables (*tethering*), la plataforma depende de un suministro energético centralizado de alta tasa de descarga, capaz de alimentar la Jetson Nano a pleno rendimiento y los motores en terrenos rugosos simultáneamente. Las características electroquímicas y la disposición física de la batería se detallan en la Fig. 15 y en la Tabla VIII.



Fig. 15. Paquete de batería LiPo de 6000 mAh y 11.1 V. Fuente: [31].

Tabla VIII. Especificaciones del sistema energético. Fuente: [31].

Parámetro	Especificación
Tecnología química	Polímero de litio (LiPo)
Voltaje nominal	11.1 V
Capacidad de carga	6000 mAh
Autonomía estimada	Aproximadamente 60 minutos de trabajo continuo

2.4. Componentes lógicos

El sistema de software gobierna la autonomía operacional del robot mediante una estructura modular, organizada en capas funcionales para facilitar la comunicación entre los componentes:

- **Entorno del sistema operativo:** La plataforma opera sobre una distribución de Linux Ubuntu como entorno base. Las actividades de programación, depuración y transferencia de archivos se apoyaron en el Subsistema de Windows para Linux (WSL) y en conexiones SSH, mientras que la ejecución final de los nodos se realizó localmente en la NVIDIA Jetson Nano.
- **Capa de comunicación (ROS):** La comunicación entre los componentes del sistema se gestiona mediante *Robot Operating System* (ROS), empleando su mecanismo de publicación y suscripción de tópicos. Los algoritmos de visión se implementaron en Python 3; por ello, se evitó el uso

de `cv_bridge` cuando se presentaron incompatibilidades con el entorno de visión, y se optó por capturar las imágenes directamente mediante OpenCV/V4L2. De esta forma, las tramas de cámara se convierten en matrices de NumPy dentro del mismo entorno de inferencia, mientras ROS se mantiene como middleware para los sensores, actuadores y comandos de movimiento.

- **Navegación, cartografía y percepción:** En la fase exploratoria se emplearon algoritmos de SLAM, como *Gmapping*, para construir mapas de ocupación (*Occupancy Grid*) a partir de la odometría y los escaneos LiDAR. En la operación autónoma final, la navegación se basa en un control reactivo que utiliza las mediciones del LiDAR para mantener el robot centrado en el surco. La IMU se emplea para supervisar la actitud del robot y detectar trepadas, mientras que la odometría se utiliza como referencia de apoyo durante las maniobras de cambio de hilera. Por su parte, la percepción visual se encarga de clasificar y contar plantas, aplicando una restricción espacial en la mitad inferior de la imagen y un periodo de inactivación (*cooldown*) entre detecciones. Esta separación funcional evita que la clasificación visual intervenga directamente en la generación de comandos de trayectoria.
- **Flujo de procesamiento e inferencia en visión:** La evaluación fitosanitaria se basa en un modelo de detección de objetos entrenado en PyTorch y optimizado para ejecución embebida mediante ONNX y TensorRT. Antes de la inferencia, cada imagen capturada en formato BGR se redimensiona, se convierte a RGB y se normaliza mediante arreglos de NumPy. Posteriormente, el motor TensorRT procesa la imagen y entrega las detecciones candidatas, las cuales son filtradas por puntaje de confianza, posición espacial y temporización para determinar la presencia de plantas sanas, con plaga o con estrés.

2.5. Diseño experimental

El diseño experimental de este proyecto comprende la estructuración, configuración y ensamblaje de los subsistemas físicos y lógicos que conforman la plataforma de inspección agrícola. El proceso busca garantizar que la integración entre el hardware del Hiwonder JetTank y el software basado en ROS opere de manera coordinada para cumplir con los objetivos de navegación autónoma y visión artificial, estableciendo protocolos claros para favorecer la reproducibilidad de las pruebas en campo.

2.5.1. Diseño y caracterización cinemática de la plataforma

La parametrización del sistema dinámico comprendió la determinación de las restricciones físicas del chasis de orugas, la calibración de la trocha efectiva entre bandas de rodadura y el acotamiento de las velocidades lineal y angular, con el propósito de mitigar perturbaciones mecánicas sobre el sistema radicular o el dosel foliar durante el desplazamiento interhilera.

Fase preliminar de adquisición de datos Previo a los ensayos de navegación autónoma, se implementó un protocolo de recolección de datos articulado en dos ejes experimentales. En primer lugar, se ejecutaron pruebas de conducción teleoperada orientadas a caracterizar la respuesta del sensor LiDAR y validar la generación de mapas de rejilla de ocupación (*Occupancy Grid Maps*). En segundo lugar, se procedió a la adquisición del registro visual de cultivos de ciclo corto. Ante la pérdida de los especímenes biológicos durante el período de preparación, el conjunto de datos de evaluación se construyó mediante réplicas sintéticas tridimensionales, diseñadas para reproducir tres condiciones fitosanitarias discretas: tejido sano, infestación por plagas y sintomatología de estrés foliar. Las imágenes obtenidas fueron anotadas, normalizadas y estructuradas para el entrenamiento de la arquitectura de aprendizaje profundo.

2.5.2. Diseño eléctrico

El diseño eléctrico está orientado a garantizar un suministro de energía continuo y seguro a todos los componentes a bordo. Se configuró un sistema de alimentación centralizado basado en un paquete de baterías de polímero de litio (LiPo) de alta tasa de descarga. Mediante la tarjeta de expansión y distribución de energía, se implementaron reguladores de voltaje (*buck converters*) para separar lógicamente los circuitos: se estableció una línea de 5 V / 4 A dedicada exclusivamente a la microcomputadora NVIDIA Jetson Nano para evitar caídas de tensión por picos de procesamiento de la GPU, y una línea independiente para suministrar potencia a los controladores de los motores de corriente continua, aislando así el ruido eléctrico generado por los actuadores.

2.5.3. Diseño del sistema de control reactivo y supervisión de seguridad

El sistema de control fue diseñado con una separación funcional de responsabilidades para favorecer la estabilidad en entornos agrícolas no estructurados.

Reconociendo que la variabilidad lumínica y la geometría irregular de la biomasa foliar pueden afectar la fiabilidad de la cámara para tareas de navegación, el control cinemático lateral se delegó al sensor LiDAR, mientras que la visión artificial se destinó a tareas de percepción, conteo y clasificación.

Modelo cinemático y ley de control

El comportamiento cinemático del vehículo y el bucle de corrección de trayectoria se fundamentan en el modelo diferencial y en la formulación del controlador PD que se detalla a continuación.

El acoplamiento entre la velocidad lineal (v) y angular (ω) del centro de masa del robot respecto a las velocidades de rotación de las orugas derecha e izquierda ($\dot{\varphi}_R, \dot{\varphi}_L$) se expresa como:

$$v = \frac{R(\dot{\varphi}_R + \dot{\varphi}_L)}{2}, \quad \omega = \frac{R(\dot{\varphi}_R - \dot{\varphi}_L)}{L}, \quad (2)$$

donde R es el radio efectivo de las ruedas motrices y L es la distancia entre los ejes de las orugas.

Para implementar la *ley de repulsión reactiva*, se calcula el error de centrado $e(t)$. Se adopta esta denominación porque el comportamiento resultante es de naturaleza repulsiva: cuando el robot se aproxima a una de las paredes del surco, la señal de corrección lo empuja hacia el lado opuesto, alejándolo de la superficie más cercana. A diferencia de las leyes de repulsión basadas en campos de potencial artificial, donde la fuerza crece de forma no lineal con la inversa de la distancia al obstáculo, la repulsión en este trabajo se materializa mediante un controlador PD que actúa sobre la asimetría de las distancias laterales, lo que garantiza una respuesta proporcional y amortiguada sin singularidades cuando el robot se encuentra muy próximo a una pared. El error de centrado se obtiene a partir de las distancias medidas por el LiDAR hacia la pared izquierda (d_{izq}) y derecha (d_{der}) del surco:

$$e(t) = \frac{d_{izq}(t) - d_{der}(t)}{2}. \quad (3)$$

Para reducir el impacto de lecturas aisladas del LiDAR sobre la señal de error, se aplica un filtro exponencial de suavizado antes de la acción de control:

$$e_f(t) = \alpha \cdot e_f(t-1) + (1 - \alpha) \cdot e(t), \quad (4)$$

donde $\alpha = 0,7$ actúa como factor de memoria sobre el error previo.

La corrección del rumbo se realiza mediante un control proporcional-derivativo (PD) en su forma discreta. La velocidad angular de comando (ω_{cmd}) se calcula con la siguiente expresión:

$$\omega_{cmd}[k] = K_p \cdot e_f[k] + K_d \cdot \Delta e_f[k], \quad (5)$$

donde $\Delta e_f[k] = e_f[k] - e_f[k - 1]$ representa la diferencia finita del error filtrado entre dos iteraciones consecutivas del bucle de control. Se omite la normalización por el período de muestreo dado que la frecuencia de operación del lazo es aproximadamente constante; en consecuencia, el efecto de la tasa de variación temporal queda absorbido directamente en la sintonización empírica de la ganancia derivativa.

En esta ley de control, la ganancia K_p define la rapidez con la que el robot retorna al centro del surco, mientras que la ganancia K_d actúa como un término de amortiguamiento para reducir oscilaciones (*chattering*) cuando el robot se acerca a las paredes. Las ganancias se sintonizaron experimentalmente durante las pruebas de campo; en la configuración final se emplearon $K_p = 3$ y $K_d = 1$.

Límites de velocidad angular por contexto operativo

Los comandos angulares se acotan según la fase de la maniobra para proteger los actuadores y mantener la estabilidad:

- **Corrección lateral durante la navegación en hilera:** la salida del controlador PD se satura en el rango $\omega_{cmd} \in [-0,4, 0,4]$ rad/s.
- **Fuga hacia la apertura detectada:** al confirmar el fin de hilera y antes de iniciar el giro en U, el robot se desplaza hacia el lado abierto con una velocidad angular de 0,40 rad/s y una velocidad lineal reducida a 0,078 m/s (correspondiente al 65 % de la velocidad de crucero), facilitando la salida del surco sin impacto contra las paredes.
- **Giro en U:** la maniobra de pivote se organiza en tres fases diferenciadas según el ángulo girado y la proximidad a la pared interna del giro:
 1. *Fase principal de pivote* (primeros 160°): la velocidad angular nominal es de 0,35 rad/s con una velocidad lineal residual de 0,06 m/s para favorecer la tracción de las orugas. Durante esta fase, la distancia a la pared interna del giro determina el comportamiento:
 - Si la pared interna se encuentra a menos de 0,25 m, se aplica un retroceso lineal de 0,05 m/s y un refuerzo angular de +0,1

rad/s, alcanzando un máximo de 0,45 rad/s, con el fin de alejar el chasis de la superficie.

- Si la pared interna se encuentra entre 0,25 m y 0,40 m, la velocidad angular se reduce al 80 % del valor nominal (0,28 rad/s), manteniendo la velocidad lineal residual de 0,06 m/s, para ejecutar un giro más conservador.
 - Si la pared interna supera los 0,40 m, se mantiene la velocidad angular nominal de 0,35 rad/s.
2. *Aproximación final* (últimos 15°): una vez superados los 160° de giro, la velocidad angular se calcula de forma proporcional al ángulo restante, acotada entre 0,15 y 0,35 rad/s, con una ganancia de 1,5, mientras se mantiene la velocidad lineal residual de 0,06 m/s.
 3. *Finalización*: el giro se considera completado cuando el ángulo restante es inferior a una tolerancia de 0,15 rad (aproximadamente 8,6°).

Durante toda la maniobra, la IMU supervisa la actitud del robot; si se superan los umbrales de inclinación de seguridad, se interrumpe el pivote y se aplica un retroceso de 0,05 m/s con una corrección angular fija de 0,2 rad/s en la dirección del giro.

- **Navegación reactiva con LiDAR:** El chasis del robot bloquea parcialmente el campo de visión trasero del LiDAR, por lo que la navegación emplea principalmente la región frontal y lateral disponible. El escaneo se sectoriza alrededor del índice central del sensor, dividiéndose en sectores frontal, izquierdo y derecho. Para cada sector se calcula una distancia robusta mediante percentiles bajos (el percentil se fijó en 10 para el sector frontal, con el fin de priorizar la detección temprana de obstáculos, y en 15 para los sectores laterales, donde la geometría del surco es más regular) y un filtro de mediana sobre una ventana de tres lecturas, con el fin de reducir el impacto de ruido y lecturas aisladas. Cuando ambas paredes del surco son visibles, el error de centrado se calcula como la diferencia entre las distancias izquierda y derecha. Si solo una pared es visible, se utiliza una referencia lateral de 0,35 m para mantener el robot alineado. Este lazo de control no depende de la odometría para el centrado lateral, lo que reduce el impacto del deslizamiento de las orugas; la odometría se emplea únicamente como referencia de apoyo durante las maniobras de cambio de hilera.
- **Detección del fin de hilera:** El fin de hilera se detecta mediante una estrategia híbrida basada en la apertura lateral de las paredes del

surco. Para la primera hilera, se considera apertura cuando la pared derecha se aleja respecto a la izquierda; para la segunda hilera, se evalúa el caso contrario. La condición de apertura se acepta cuando la distancia exterior supera a la interior en al menos 0,20 m, o cuando la pared exterior alcanza una distancia considerada como ausencia de referencia, fijada en 1,80 m, siempre que además supere a la pared opuesta en al menos 0,10 m. Esta condición se confirma temporalmente durante un mínimo de 0,35 s y, cuando corresponde, se verifica que el robot comience a orientarse hacia el lado abierto mediante el ángulo de *yaw* estimado por la odometría, con un umbral de activación de 15° mantenido durante al menos 0,15 s. Adicionalmente, se exige un tiempo mínimo de permanencia en la hilera de 3 s antes de aceptar cualquier condición de fin de fila, evitando activaciones prematuras durante la entrada al surco. Una vez confirmado el fin de hilera, el robot ejecuta la maniobra de giro en U.

El objetivo de giro se fijó en 175° respecto al *yaw* de referencia para compensar los efectos de deslizamiento lateral observados en el terreno agrícola. Durante la maniobra, la IMU supervisa la actitud del robot; si se superan los umbrales de inclinación de seguridad, se aplica un retroceso controlado para evitar trepadas o vuelcos.

En la Fig. 16 se esquematiza el flujo de señales de la arquitectura de control con separación funcional entre navegación y percepción visual.

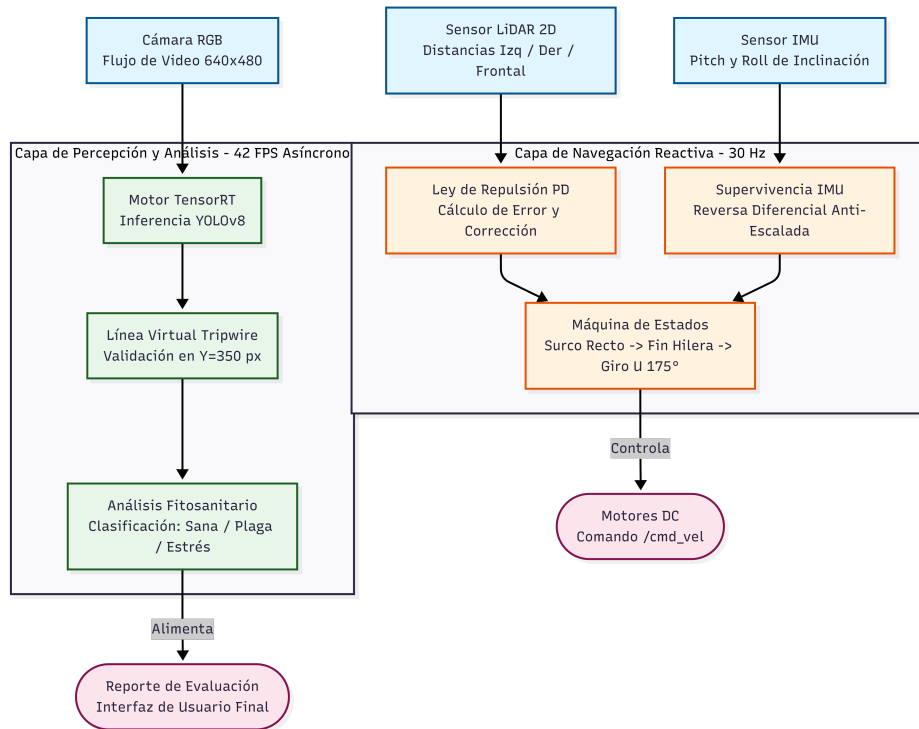


Fig. 16. Diagrama de bloques de la arquitectura de control con separación funcional (Navegación LiDAR + Percepción Visual). Fuente: Elaboración propia.

2.5.4. Diseño del sistema de comunicación

Para garantizar la interoperabilidad de los componentes, el diseño de comunicación se estructuró en dos niveles. A nivel de hardware interno (comunicación embebida), se configuró el bus I²C para la transmisión de datos entre la placa base y los sensores de bajo nivel, debido a su eficiencia en distancias cortas. Para el control de los motores y servos, se habilitaron puertos de comunicación serial UART. A nivel de sistema (comunicación externa), se empleó el protocolo inalámbrico Wi-Fi bajo el estándar IEEE 802.11 durante las etapas de depuración y monitorización remota. Esta red local permitió la comunicación entre los nodos de ROS coordinados por el ROS Master, facilitando la inspección de tópicos y el ajuste de parámetros desde una estación de trabajo. En el despliegue final, el sistema se operó de manera local en la NVIDIA Jetson Nano, evitando la dependencia de conexiones externas.

2.5.5. Diseño de la interfaz gráfica de monitoreo

El diseño de la interfaz gráfica de usuario (GUI) se enfocó en la usabilidad embebida y el bajo consumo computacional, omitiendo ajustes complejos en tiempo real. La interfaz fue estructurada para desplegarse localmente en la pantalla táctil y mostrar al operador la información esencial del sistema. Para ello, se definieron los siguientes elementos visuales: un panel de estado tipo semáforo que indica la conducta del robot (navegando, girando o alerta de escalada), un contador fitosanitario central que incrementa con cada detección validada, contadores diferenciados por clase (sana, plaga y estrés), y una etiqueta con la posición odométrica estimada del robot. La interfaz se ejecuta en la propia Jetson Nano y se actualiza durante el ciclo de operación del sistema, permitiendo la supervisión local sin depender de estaciones remotas.

2.5.6. Entrenamiento del modelo de visión artificial y optimización *edge computing*

El subsistema de visión artificial embebido requiere un equilibrio crítico entre la precisión de detección de la biomasa foliar y la tasa de procesamiento computacional (cuadros por segundo) ejecutable localmente en la NVIDIA Jetson Nano. Para cumplir con esta restricción, el desarrollo analítico se dividió en tres etapas: adquisición del dataset, entrenamiento por transferencia (*fine-tuning*) y pipeline de optimización mediante TensorRT.

Adquisición, biomasa sintética y preprocesamiento del dataset Se estructuró un banco de imágenes propio destinado al entrenamiento del modelo de visión artificial, enfocado en cultivos de lechuga de ciclo corto. Ante la pérdida imprevista de los especímenes vegetales durante la fase preparatoria—ocasionada por la acción de insectos defoliadores (*Atta* spp.) en la parcela experimental—se implementó como estrategia de mitigación el uso de réplicas sintéticas tridimensionales a escala real. Con el propósito de supervisar al modelo en la discriminación de tres estados fitosanitarios discretos, se produjeron tres tipologías de especímenes artificiales:

- **Clase 0 (*sana*):** Réplicas monocromáticas en verde estándar, representativas de tejido foliar íntegro y vigoroso.
- **Clase 1 (*plaga*):** Réplicas con pigmentaciones oscuras puntuales que emulan colonias de insectos o focos de necrosis localizada.
- **Clase 2 (*estrés*):** Réplicas fabricadas en tonalidad amarilla para reproducir la clorosis asociada a estrés hídrico o déficit nutricional.

Se capturaron 255 registros fotográficos sobre el terreno de ensayo bajo diversas condiciones de iluminación natural, contrastando escenarios de radiación solar directa y zonas de sombra. Concluida la captura, las imágenes se importaron en la plataforma Roboflow para efectuar la anotación manual de las cajas delimitadoras y formalizar el flujo de preprocesamiento. En la Fig. 17 se presenta la organización del repositorio personalizado en dicha plataforma, evidenciando la distribución de las muestras y las etiquetas asignadas. Dentro de esta etapa preparatoria, las imágenes se reorientaron automáticamente (*auto-orient*) y se normalizaron a una resolución espacial de 512×512 píxeles, satisfaciendo el formato dimensional de entrada requerido por la red YOLOv8.

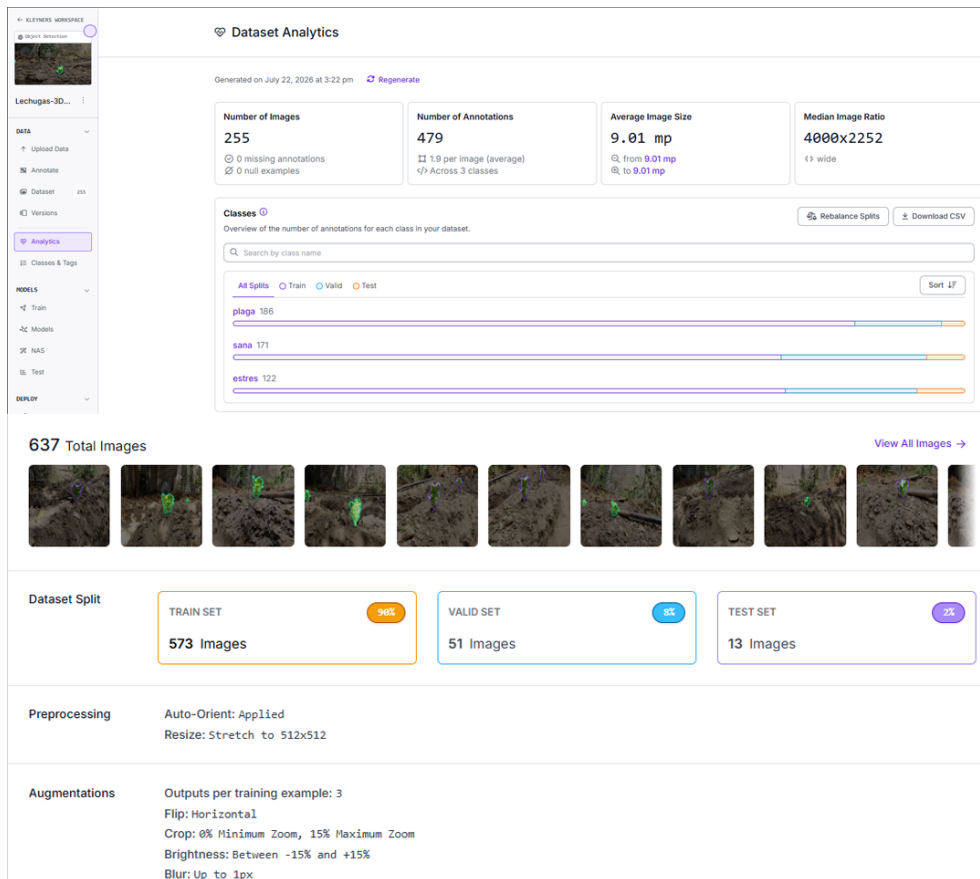


Fig. 17. Estructura del dataset personalizado en la plataforma Roboflow. Fuente: Autoría propia.

Para robustecer el desempeño del modelo frente a la variabilidad lumínica y geométrica en el entorno de pruebas, la muestra inicial, compuesta por 255 imágenes base y 479 anotaciones distribuidas en las clases *sana*, *plaga*

y *estrés*, se procesó mediante una etapa de aumentación de datos (*data augmentation*) en Roboflow. Las transformaciones estocásticas configuradas incluyeron volteo horizontal (*horizontal flip*), recorte con zoom aleatorio (*crop* entre 0 % y 15 %), ajustes de brillo en un rango de -15% a $+15\%$ y desenfoque óptico (*blur*) de hasta 1 píxel. Como resultado de esta etapa, el conjunto final alcanzó un total de 637 imágenes, correspondientes a 255 imágenes originales más 382 versiones aumentadas. La partición del dataset se estructuró de forma estratificada conservando la siguiente proporción: un 90 % para el conjunto de entrenamiento (*train set*), correspondiente a 573 imágenes; un 8 % para el conjunto de validación (*valid set*), equivalente a 51 imágenes; y un 2 % reservado para el conjunto de prueba (*test set*), integrado por 13 imágenes. Esta distribución permite una evaluación independiente del error de generalización y respalda el monitoreo objetivo de las pérdidas y la precisión durante el entrenamiento para prevenir el sobreajuste (*overfitting*).

Arquitectura de red y *transfer learning* Como base algorítmica se seleccionó el modelo de detección de objetos en una sola etapa YOLOv8 Nano (*You Only Look Once*), dada su estructura ligera basada en un *backbone* CSPDarknet y un cuello PANet, idónea para ejecuciones en dispositivos embebidos (*edge computing*). El ajuste por transferencia de conocimiento (*transfer learning*) utilizó como punto de partida los pesos preentrenados del conjunto de datos COCO, ejecutando la fase de cómputo sobre una GPU Tesla T4 en la plataforma Google Colab.

Para el entrenamiento, se configuró el optimizador AdamW con una tasa de aprendizaje inicial (η) de 0,01. Se empleó un tamaño de lote (*batch size*) de 16 y se entrenó durante 100 épocas. Durante esta etapa, el flujo de entrenamiento (*pipeline*) de YOLOv8 aplica técnicas automáticas de aumento de datos (*data augmentation*) sobre las imágenes de entrada—tales como mosaicos espaciales y variaciones de escala—con el fin de incrementar la capacidad de generalización del detector y mitigar el sobreajuste (*overfitting*). En la Fig. 18 se ilustra la composición generada mediante dicha técnica de mosaico, junto con las cajas delimitadoras (*bounding boxes*) empleadas en la supervisión del modelo para la detección y clasificación fitosanitaria.



Fig. 18. Mosaico de lotes de entrenamiento (*train batch*) generado automáticamente por YOLOv8, donde se evidencia el etiquetado de las plantas y la aplicación de aumentación de datos por mosaico. Fuente: Autoría propia.

A medida que avanzan las épocas de entrenamiento, los coeficientes internos de la red se ajustan, reduciendo el margen de error y elevando la precisión de reconocimiento. La red alcanzó una convergencia estable sin indicios de sobreajuste, con la pérdida de clasificación estabilizándose en 0,3093. La evolución detallada de este aprendizaje se expone en las curvas de la Fig. 19.

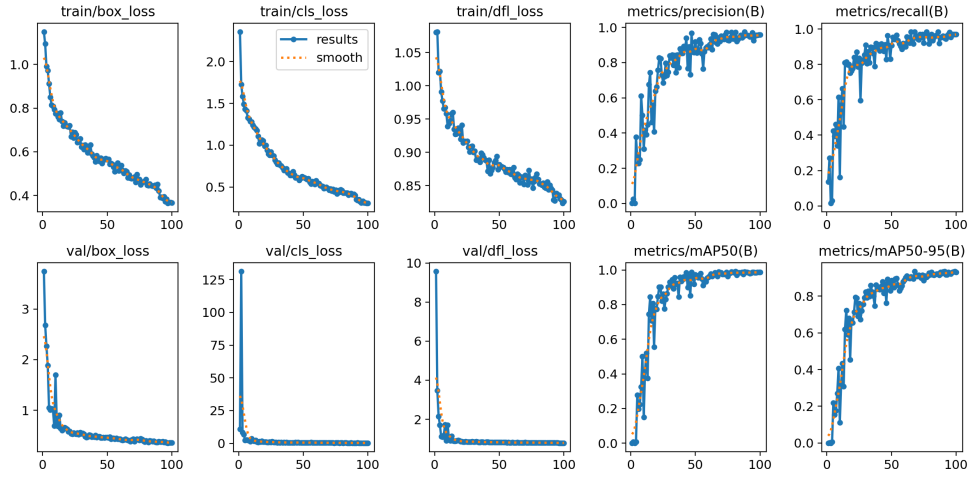


Fig. 19. Curvas de pérdida y precisión mAP@50 durante las épocas de entrenamiento en el servidor base. Fuente: Autoría propia a través de la interfaz de TensorBoard.

Flujo de optimización a TensorRT (best.engine) La ejecución directa de la red empleando los pesos originales en formato PyTorch (.pt) sobre la unidad NVIDIA Jetson Nano introduce una latencia considerable debido al cómputo en coma flotante de 32 bits (FP32). A fin de mitigar esta sobrecarga computacional y mejorar la velocidad de respuesta del sistema embebido, se estructuró una secuencia de conversión y optimización de capas descrita en la Fig. 20.

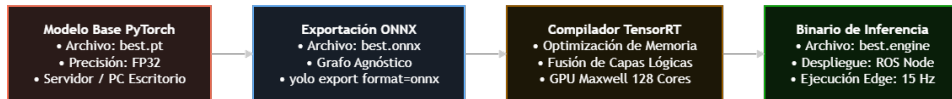


Fig. 20. Diagrama del flujo de optimización de PyTorch a TensorRT (.pt → .onnx → .engine) para despliegue embebido. Fuente: Autoría propia.

1. **Exportación a representación intermedia ONNX:** Los pesos entrenados se convirtieron al formato estándar ONNX (Open Neural Network Exchange), desacoplando la arquitectura del entorno original para simplificar los nodos del grafo computacional.
2. **Compilación en la Jetson Nano:** Se usó el motor TensorRT para compilar el archivo ONNX directamente en la plataforma Jetson Nano. El objetivo de este paso fue adaptar el grafo computacional a la GPU

Maxwell de 128 núcleos y optimizar la ejecución sobre el hardware embebido.

3. **Fusión de capas y generación del binario:** Durante la compilación, `TensorRT` une automáticamente capas de convolución con funciones de activación cuando es posible. Esta reorganización minimiza la sobrecarga de acceso a memoria y disminuye la latencia de inferencia, generando el motor binario optimizado (`best.engine`) para su ejecución en tiempo real sobre la GPU embebida.

La caracterización de rendimiento ejecutada mediante `trtexec` sobre el SoC NVIDIA Tegra X1 arrojó una huella de almacenamiento de 25 MB para el binario final, tras un tiempo de compilación de 284,2 s y una latencia de inferencia de 38,95 ms por cuadro, lo que equivale a una tasa de procesamiento de 25,67 FPS. En la implementación final, el bucle principal de operación se configuró con `rospy.Rate(30)`, estableciendo un techo máximo de 30 Hz (33,3 ms por ciclo). No obstante, dado que cada iteración del bucle integra la captura de imagen, la inferencia del modelo (≈ 39 ms), la actualización del control reactivo y el refresco de la interfaz, el tiempo de ciclo efectivo supera el límite impuesto por el `Rate`, por lo que la frecuencia real de operación queda determinada por el componente más lento: la inferencia visual. En consecuencia, el sistema opera de forma efectiva a aproximadamente 25,7 Hz, valor que resulta adecuado para la velocidad de crucero de 0,12 m/s empleada en los recorridos autónomos.

2.6. Desarrollo experimental

El desarrollo experimental abarca la fase de construcción, ensamblaje e integración de los subsistemas diseñados previamente. En esta etapa se ejecutaron los procedimientos físicos y lógicos para poner en marcha el vehículo autónomo, buscando que cada componente operara según las especificaciones requeridas para la navegación y la percepción del entorno en el escenario agrícola.

2.6.1. Implementación del chasis robótico

El acondicionamiento físico del móvil comprendió la integración mecánica del chasis de orugas del Hiwonder JetTank junto con el montaje de sus elementos sensoriales y de procesamiento. En el nivel inferior de la estructura se montaron los motores DC con *encoders* de efecto Hall, y se realizó un calibrado manual sobre la tensión de las orugas; este ajuste mecánico resultó

determinante para reducir pérdidas de tracción y deslizamiento (*slippage*) frente a las irregularidades del terreno agrícola.

Sobre la plataforma intermedia de acrílico y perfilería metálica se fijó la unidad central NVIDIA Jetson Nano. Para la adquisición de datos espaciales, el sensor LiDAR 2D se ubicó en la zona superior central del vehículo, buscando una línea de visión libre de interferencias estructurales. Por su parte, la cámara RGB-D se montó en el frontal mediante un soporte articulado PTZ de dos grados de libertad. Durante las validaciones físicas se optó por fijar la elevación del lente de forma manual mediante la fricción del mecanismo, manteniendo el eje *tilt* en 0° por software y destinando la actuación electrónica exclusivamente al eje *pan* (barrido horizontal) para los cambios de carril; esta configuración permitió examinar las hileras laterales sin modificar el eje de navegación principal del robot.

2.6.2. Configuración del entorno físico de pruebas

Con el propósito de evaluar el comportamiento mecánico de la estructura y la respuesta cinemática durante el recorrido, se implementó un espacio experimental a escala (patio de ensayos) provisto de pasillos con variaciones de iluminación y obstáculos sólidos. Esta configuración reproduce la geometría de un cultivo de ciclo corto mediante surcos de tierra realzados, exigiendo al vehículo el uso continuo del sistema de tracción por orugas.

La vista superior ilustrada en la Fig. 21 evidencia el desplazamiento autónomo del móvil dentro de la hilera y la distancia de margen respecto a los bordes laterales. Por otra parte, la toma a ras de suelo en la Fig. 22 deja al descubierto la estrechez del canal y la mínima holgura operativa del chasis. Dicho escenario impone restricciones cinemáticas severas al algoritmo de control reactivo, el cual requiere mantener un centrado riguroso para impedir colisiones con la biomasa vegetal o la trepada inadvertida sobre los taludes del surco.



Fig. 21. Vista superior del robot navegando autónomamente dentro de la hilera de cultivo. Fuente: Autoría propia.



Fig. 22. Vista a nivel del suelo evidenciando la estrechez del surco y el reto de navegación autónoma. Fuente: Autoría propia.

2.6.3. Implementación del sistema de alimentación

La etapa de potencia abarcó el conexionado eléctrico y la canalización del cableado para la distribución de energía hacia las cargas lógicas y de potencia, minimizando el acoplamiento de ruido e interferencias.

En el compartimento central e inferior del chasis se alojó la batería de polímero de litio (LiPo). Su ubicación en la parte inferior reduce el centro de gravedad del robot, lo cual es clave para evitar vuelcos u oscilaciones pronunciadas cuando el vehículo pasa por baches y surcos de tierra del terreno agrícola. La batería se interconectó con la placa de expansión central para la distribución de energía hacia las distintas etapas del robot. Antes de

energizar el módulo principal, se efectuaron lecturas de tensión mediante un multímetro para verificar la estabilidad de los reguladores, validando una salida constante de 5 V a 4 A para la alimentación de la NVIDIA Jetson Nano. Este flujo eléctrico contribuye a prevenir caídas de tensión bruscas (*brownouts*) y reinicios del sistema frente a los picos de consumo demandados por la GPU durante el proceso de inferencia.

De forma aislada, se cablearon los buses de potencia dirigidos a los controladores de los motores de corriente continua. El acondicionamiento eléctrico finalizó con pruebas de continuidad e inspección de polaridad en la bornera de entrada para mitigar el riesgo de cortocircuitos, contribuyendo así a la autonomía energética de la plataforma durante los ensayos de campo.

2.6.4. Implementación del sistema de control y mitigación de interferencia lumínica

La integración del control lógico interconectó la unidad principal NVIDIA Jetson Nano con la placa controladora de motores mediante comunicaciones por bus I²C. En la arquitectura de ROS, la placa de bajo nivel recibe las velocidades comandadas en el tópico `cmd_vel` y las convierte en señales de modulación por ancho de pulsos (PWM) para la etapa de potencia.

En los ensayos iniciales sobre el cultivo, la navegación guiada por retroalimentación visual (*visual servoing*) mostró divergencia debido a la morfología irregular del follaje. Para contrarrestar esta limitación, se adoptó una separación funcional de responsabilidades: La guía del vehículo se asignó al sensor LiDAR operando bajo una ley de repulsión, mientras que la cámara se destinó a tareas de percepción, conteo e inspección.

La unidad de medición inercial (IMU) se incorporó para supervisar la actitud espacial del vehículo y detectar el escalamiento involuntario del chasis sobre los taludes del surco. Para tal fin, la rutina de control extrae la orientación tridimensional a partir del cuaternión unitario $q = (q_w, q_x, q_y, q_z)$ reportado por el sensor. La deducción de los ángulos de actitud para el cabeceo (*pitch*, θ) y el balanceo (*roll*, ϕ), bajo la convención de ángulos de Euler (secuencia Tait-Bryan *Z-Y-X*), se expresa formalmente mediante:

$$\theta = \arcsin(2(q_w q_y - q_z q_x)), \quad (6)$$

$$\phi = \text{atan2}(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)). \quad (7)$$

El algoritmo de seguridad monitorea los límites de actitud, activando la maniobra de retroceso cuando se satisface la condición $\theta > 15^\circ$ o $\phi > 10^\circ$, evento asociado al escalamiento indeseado de los taludes. Una vez activada

la alerta, el robot aplica un retroceso controlado a $-0,06$ m/s. Durante este retroceso, si el ángulo de balanceo supera los 5° , se genera un comando angular correctivo proporcional al *roll* medido, con una ganancia de $5,0$, con el fin de enderezar el chasis y reducir el riesgo de vuelco:

$$\omega_{\text{corr}} = \begin{cases} 5,0 \cdot \phi, & \text{si } |\phi| > 5^\circ \\ 0, & \text{en caso contrario.} \end{cases} \quad (8)$$

Si el balanceo no supera los 5° , el retroceso se ejecuta en línea recta sin corrección angular. Esta estrategia permite recuperar el tránsito dentro del surco sin intervenir en el lazo de navegación principal.

Adicionalmente, en las pruebas a campo abierto se constató la saturación del sensor IR de la cámara Astra Pro Plus por efecto de la radiación solar directa (*washout effect*). Al operar bajo una separación funcional entre navegación y percepción, esta falla no afectó la guía del robot. El canal de profundidad fue descartado para la inferencia final, mientras que la percepción fitosanitaria se realizó sobre el canal RGB mediante el modelo YOLOv8 entrenado con imágenes bajo distintas condiciones de iluminación.

2.6.5. Implementación del sistema de comunicación

La topología de red del robot se estructuró dividiendo el tráfico en dos dominios operacionales: interconexión local e interfaz remota. En el ámbito local, la integración física del LiDAR se resolvió mediante un canal serial USB directo con la Jetson Nano. La cámara frontal RGB-D se conectó a través del hub USB de la plataforma y fue accedida mediante el backend V4L2 de OpenCV, evitando depender de un nodo de cámara independiente. Frente a los problemas de compatibilidad entre la librería nativa `cv_bridge` de ROS y el entorno de ejecución en Python 3, se optó por omitir dicho puente y capturar las imágenes directamente desde el dispositivo de video. Cada fotograma recibido en formato BGR se convierte en una matriz de NumPy y se somete a las etapas de redimensionamiento, conversión a RGB y normalización antes de la inferencia.

El enlace externo para monitorización y depuración se estableció sobre la interfaz Wi-Fi de la tarjeta emisora bajo un esquema de direccionamiento IP estático (`192.168.100.95`). Desde la estación de trabajo central impulsada por Windows 11 con WSL, la orquestación se alineó configurando las variables del máster de ROS (`ROS_MASTER_URI`) sobre el `localhost`. Con la finalidad de evadir el estrechamiento de banda del reenvío X11 durante la transmisión de video, el desarrollo se centralizó conectando el entorno IDE Cursor mediante una sesión segura por SSH; este enfoque habilitó la inspección dinámica del

código y el ajuste de la arquitectura de control con separación funcional sin introducir cuellos de botella por la renderización remota de interfaces gráficas.

2.6.6. Implementación de la interfaz gráfica embebida

Con el fin de proporcionar un mecanismo de supervisión directo en campo y suprimir la dependencia de enlaces inalámbricos o estaciones de control externas, se desarrolló una interfaz gráfica de usuario (GUI, por sus siglas en inglés) desplegada en la pantalla táctil integrada del chasis. La aplicación se programó mediante la biblioteca nativa **Tkinter** de Python, seleccionada por su reducida sobrecarga computacional y su compatibilidad directa con el entorno Linux Ubuntu sobre la NVIDIA Jetson Nano.

La GUI se ejecuta en la unidad de procesamiento central y sincroniza sus variables dentro del lazo principal de ROS. Para minimizar el consumo de recursos de la GPU y la memoria compartida, el diseño prescindió de la renderización del flujo de video continuo en pantalla, empleando elementos textuales estáticos y una distribución visual compacta estructurada en tres módulos funcionales:

- **Módulo de estado operativo:** Indicador visual tricolor que codifica el modo de control del robot: verde durante la navegación longitudinal en hilera, naranja durante la maniobra de viraje en cabecera y rojo ante la detección inercial de una inclinación crítica o pérdida de estabilidad mediante la IMU.
- **Contador e identificador fitosanitario:** Indicador numérico que actualiza sus valores cuando el modelo de inferencia (YOLOv8n) confirma una detección válida. Una muestra se contabiliza si supera el umbral de confianza establecido ($\text{conf} \geq 0,50$), se ubica por debajo de la línea virtual de validación ($y_{\text{bottom}} > 240 \text{ px}$) y sobrepasa el periodo de enfriamiento temporal (T_{cd}) respecto al registro precedente. Las detecciones se acumulan y desglosan en las clases *sana*, *plaga* o *estrés*.
- **Panel de telemetría odométrica:** Registro de coordenadas locales que presenta la estimación posicional (x, y) del robot respecto al marco de referencia inicial, permitiendo verificar la trayectoria espacial recorrida a lo largo de los surcos.

Esta arquitectura consolidó la autonomía operativa del sistema, permitiendo la supervisión, el registro y la cuantificación fitosanitaria directamente en campo, sin requerir infraestructura de telecomunicaciones auxiliar ni estaciones de cómputo remotas.

2.6.7. Configuración de parámetros cinemáticos

La configuración de la plataforma se realizó ajustando los parámetros del script principal dentro del espacio de trabajo de ROS. Se definieron los límites físicos del chasis para el controlador de movimiento diferencial: velocidad lineal constante de cruce de 0,12 m/s y velocidad angular máxima de 0,4 rad/s para las correcciones laterales del controlador PD.

Durante las maniobras de cambio de hilera se configuraron los siguientes parámetros cinemáticos: la velocidad angular de fuga hacia la apertura detectada se fijó en 0,40 rad/s, con una velocidad lineal reducida a 0,078 m/s (correspondiente al 65 % de la velocidad de cruce). La maniobra de giro en U se estructuró en tres fases. En la fase principal de pivote (primeros 160°), la velocidad angular nominal se estableció en 0,35 rad/s con una velocidad lineal residual de 0,06 m/s; si la distancia a la pared interna del giro descendía por debajo de 0,25 m, se aplicaba un refuerzo de +0,1 rad/s (alcanzando un máximo de 0,45 rad/s) junto con un retroceso lineal de 0,05 m/s; si la pared interna se situaba entre 0,25 m y 0,40 m, la velocidad angular se reducía al 80 % del valor nominal (0,28 rad/s), manteniendo la velocidad lineal residual de 0,06 m/s para ejecutar un giro más conservador. En la fase de aproximación final (últimos 15°), la velocidad angular se calculaba de forma proporcional al ángulo restante, acotada entre 0,15 rad/s y 0,35 rad/s, con una ganancia de 1,5. El giro se consideraba completado cuando el ángulo restante era inferior a una tolerancia de 0,15 rad. Todos los comandos se mantuvieron dentro de un rango seguro para los actuadores.

Se implementó un control proporcional-derivativo (PD) para la corrección de rumbo basada en las diferencias de distancia del LiDAR. Los índices del arreglo del LiDAR se mapearon a partir del índice central (377) para sectorizar el escaneo en regiones frontal, izquierda y derecha. Para reducir el impacto de lecturas aisladas, cada sector se procesó mediante distancias robustas calculadas con percentiles bajos (percentil 10 para el sector frontal y percentil 15 para los sectores laterales) y un filtro de mediana sobre una ventana de tres lecturas. Adicionalmente, el error de centrado se suavizó mediante un filtro exponencial con factor $\alpha = 0,7$ antes de la acción de control. En la configuración final del controlador se emplearon las ganancias $K_p = 3$ y $K_d = 1$.

En cuanto a los umbrales de seguridad, se estableció una distancia frontal de bloqueo de 0,45 m, confirmada durante dos segundos con distancias laterales inferiores a 0,85 m, con el fin de detener el robot ante obstrucciones reales y evitar detenciones falsas por ruido del sensor. Para la detección de fin de hilera, se consideró ausencia de pared lateral cuando la distancia supera 1,80 m, siempre que además supere a la pared opuesta en al menos 0,10 m, y se

definió una apertura mínima relativa de 0,20 m respecto a la pared opuesta. Esta condición se confirmaba temporalmente durante un mínimo de 0,35 s antes de ser aceptada. Para la última hilera, la parada final se confirmó con una distancia frontal menor a 0,70 m y distancias laterales menores a 1,50 m, mantenidas durante dos segundos.

Adicionalmente, el objetivo de giro en U se fijó en 175° respecto al *yaw* estimado por la odometría, en lugar de los 180° teóricos, con el fin de compensar el deslizamiento lateral de las orugas y lograr una reincorporación alineada al carril adyacente sin sobre-rotaciones.

Para consolidar la configuración completa del sistema y facilitar la reproducibilidad de los ensayos, la Tabla IX resume la totalidad de los parámetros empleados en el script principal, organizados por subsistema funcional.

Tabla IX. Resumen consolidado de parámetros de configuración del sistema. Fuente: Elaboración propia.

Parámetro	Valor	Descripción
Cinemática general		
Velocidad lineal de cruce	0,12 m/s	Velocidad de avance durante la navegación en hilera.
Velocidad angular máxima	0,4 rad/s	Saturación de la salida del controlador PD lateral.
Índice central del LiDAR	377	Índice de referencia del arreglo de lecturas para la sectorización del escaneo.
Distancia mínima válida	0,10 m	Umbral inferior para descartar lecturas del LiDAR por cercanía al chasis.
Distancia máxima considerada	2,0 m	Umbral superior para descartar lecturas del LiDAR fuera del rango operativo.
Percentil frontal / lateral	10 / 15	Percentil bajo utilizado en el cálculo de distancia robusta por sector.
Ventana de filtro mediana	3 lecturas	Número de muestras sobre las que se aplica la mediana para suavizar cada sector.
Número de hileras	3	Cantidad total de surcos a recorrer en la misión.
Control PD lateral		

Continúa en la siguiente página

Tabla IX (continuación)

Parámetro	Valor	Descripción
Ganancia proporcional K_p	3	Rapidez de retorno al centro del surco.
Ganancia derivativa K_d	1	Amortiguamiento para reducir oscilaciones.
Factor de filtro exponencial α	0,7	Peso del error previo en el suavizado de la señal de centrado.
Referencia lateral (1 pared)	0,35 m	Distancia deseada a la pared visible cuando solo se detecta una pared.
Visibilidad de pared lateral	1,50 m	Distancia máxima para considerar que una pared lateral es visible.
Bloqueo de seguridad		
Distancia frontal de bloqueo	0,45 m	Umbral frontal para detectar obstrucciones reales.
Distancia lateral de bloqueo	0,85 m	Umbral lateral para confirmar bloqueo real.
Tiempo de confirmación	2,0 s	Duración mínima del bloqueo antes de detener el robot.
Detección de fin de hilera		
Ausencia de pared lateral	1,80 m	Distancia a partir de la cual se considera que una pared está ausente.
Apertura mínima relativa	0,20 m	Diferencia mínima entre la pared abierta y la pared opuesta.
Margen adicional de ausencia	0,10 m	Diferencia extra exigida sobre la pared opuesta cuando la distancia supera 1,80 m.
Tiempo mínimo en hilera	3,0 s	Permanencia mínima antes de aceptar cualquier condición de fin de fila.
Confirmación temporal de apertura	0,35 s	Tiempo mínimo que debe mantenerse la apertura antes de ser aceptada.
Ángulo de activación de giro	15°	<i>Yaw</i> mínimo para confirmar que el robot comenzó a orientarse hacia el lado abierto.
Confirmación temporal de <i>yaw</i>	0,15 s	Tiempo mínimo que el <i>yaw</i> debe mantenerse en la dirección correcta.

Continúa en la siguiente página

Tabla IX (continuación)

Parámetro	Valor	Descripción
Confirmación de fin de fila	2,0 s	Tiempo de confirmación para parada final o activación del giro.
Distancia frontal fin de misión	0,70 m	Umbral frontal para la parada final en la última hilera.
Distancia lateral fin de misión	1,50 m	Umbral lateral para la parada final en la última hilera.
Giro en U		
Objetivo de giro	175°	Ángulo total de pivote respecto al <i>yaw</i> de referencia.
Velocidad angular nominal	0,35 rad/s	Velocidad angular durante la fase principal de pivote.
Refuerzo por proximidad	+0,1 rad/s	Incremento angular cuando la pared interna está a menos de 0,25 m.
Velocidad angular máxima (giro)	0,45 rad/s	Velocidad angular resultante con refuerzo activo.
Reducción conservadora	80 % (0,28 rad/s)	Velocidad angular cuando la pared interna está entre 0,25 m y 0,40 m.
Velocidad lineal residual	0,06 m/s	Velocidad lineal durante el pivote para favorecer la tracción.
Retroceso por proximidad	−0,05 m/s	Retroceso lineal cuando la pared interna está a menos de 0,25 m.
Ganancia proporcional final	1,5	Factor proporcional al ángulo restante en la fase de aproximación final.
Acotación angular final	0,15 – 0,35 rad/s	Rango de velocidad angular en la fase de aproximación final.
Tolerancia de finalización	0,15 rad	Ángulo restante máximo para considerar el giro completado.
Velocidad angular de fuga	0,40 rad/s	Velocidad angular hacia la apertura detectada.
Velocidad lineal de fuga	0,078 m/s	Velocidad lineal reducida al 65 % del crucero durante la fuga.
Tiempo de entrada a hilera	2,0 s	Duración del avance recto tras completar el giro antes de retomar el control PD.

Continúa en la siguiente página

Tabla IX (continuación)

Parámetro	Valor	Descripción
Supervisión IMU		
Umbral de cabeceo (<i>pitch</i>)	15°	Límite para detectar condición de trepada.
Umbral de balanceo (<i>roll</i>)	10°	Límite para detectar condición de trepada.
Umbral de corrección de balanceo	5°	Ángulo a partir del cual se aplica corrección angular durante el retroceso.
Ganancia de corrección angular	5,0	Factor proporcional al <i>roll</i> medido durante el retroceso.
Retroceso por escalada	−0,06 m/s	Velocidad de retroceso al superar los umbrales de inclinación.
Retroceso por escalada en giro	−0,05 m/s	Velocidad de retroceso si la escalada se detecta durante el giro en U.
Corrección angular fija en giro	0,2 rad/s	Corrección angular aplicada durante el retroceso por escalada en el giro.
Desviación máxima de <i>yaw</i>	30°	Límite de desviación de <i>yaw</i> considerado por el sistema.
Visión artificial		
Resolución de captura	640 × 480 px	Resolución de adquisición de la cámara (VGA).
Resolución de inferencia	512 × 512 px	Redimensionamiento de la imagen antes de la inferencia.
Umbral de confianza	0,45	Puntaje mínimo para aceptar una detección como válida.
Restricción espacial	$y_{\text{bottom}} > 240$ px	Validación de detecciones únicamente en la mitad inferior de la imagen.
Periodo de inactivación	4,0 s	Tiempo mínimo entre dos conteos consecutivos (<i>cooldown</i>).
Cámara PTZ		
Ángulo <i>pan</i> derecho	0,785 rad ($\approx 45^\circ$)	Orientación de la cámara hacia la hilera derecha.
Ángulo <i>pan</i> izquierdo	−0,785 rad ($\approx 45^\circ$)	Orientación de la cámara hacia la hilera izquierda.
Ángulo <i>tilt</i>	0,0 rad	Elevación fija por software.
Sistema		

Continúa en la siguiente página

Tabla IX (continuación)

Parámetro	Valor	Descripción
Frecuencia del bucle principal	30 Hz	Techo máximo configurado con <code>rospy.Rate(30)</code> .
Frecuencia efectiva de operación	$\approx 25,7$ Hz	Determinada por la latencia de inferencia (38,95 ms).
Timeout de sensores	15 s	Tiempo máximo de espera para odometría, IMU y LiDAR antes de abortar.
Modo de parada	Habilitado	Detención automática al completar todas las hileras.

2.6.8. Configuración del entorno de depuración y despliegue autónomo

Durante la etapa de desarrollo y pruebas de laboratorio, se utilizó una arquitectura de depuración remota configurando las variables de entorno `ROS_MASTER_URI` y `ROS_HOSTNAME` apuntando al `localhost` de la estación de trabajo bajo el Subsistema de Windows para Linux (WSL). Esta configuración permitió monitorizar los tópicos mediante herramientas de diagnóstico gráfico como RViz.

Sin embargo, para el despliegue definitivo en el patio de ensayos (entorno agrícola real), se eliminó por completo la dependencia de redes inalámbricas locales y la monitorización externa. Se configuró un sistema de autoarranque local en la NVIDIA Jetson Nano utilizando lanzadores de escritorio nativos (*.desktop files*) en Ubuntu. Estos lanzadores ejecutan *scripts* de Bash que inician de forma transparente el núcleo de ROS (`roscore`), los nodos de sensores y el script principal de control y percepción, permitiendo operar el robot en campo mediante periféricos locales o pantallas embebidas sin requerir conexiones Wi-Fi activas.

Para facilitar la operatividad en campo por parte del usuario, se diseñaron dos accesos directos principales en la pantalla táctil del robot (Fig. 23). El primero, **Iniciar SLAM**, se encarga de levantar el entorno ROS y los nodos del LiDAR, la IMU y la odometría, así como el proceso de cartografía utilizado para registro. El segundo, **Arrancar Tesis**, ejecuta el script principal de control reactivo, carga el motor de inferencia TensorRT en la GPU y despliega la interfaz gráfica de usuario, dando inicio al monitoreo fitosanitario autónomo. El protocolo detallado de puesta en marcha se encuentra documentado en el anexo de este trabajo.

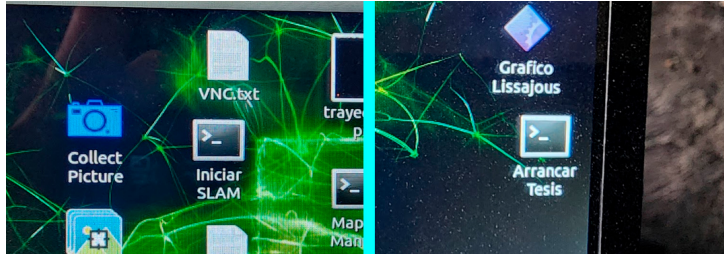


Fig. 23. Lanzadores de escritorio para el despliegue autónomo en la pantalla táctil de la Jetson Nano. Fuente: Autoría propia.

2.6.9. Mitigación de conflictos de red y optimización del enlace inalámbrico

Durante la implementación del entorno de depuración remota, se identificó un conflicto crítico de hardware a nivel de sistema operativo. La tarjeta de red inalámbrica integrada (Realtek RTL8822CE) presentaba inestabilidad en el arranque, manteniéndose en estado *unavailable* e impidiendo la comunicación por SSH mediante la IP estática asignada (192.168.100.95). El diagnóstico a través de los registros del sistema reveló que el demonio `wpa_supplicant` entraba en un bucle de adquisición fallida de la interfaz, bloqueando al gestor de red (*NetworkManager*).

Para garantizar una telemetría estable durante las fases de depuración remota y eliminar la dependencia de cables Ethernet físicos, se desarrolló un servicio del sistema (*systemd service*) con prioridad de arranque. Este servicio ejecuta un *script* de Bash en el encendido que detiene el gestor de red, termina forzosamente el proceso `wpa_supplicant` conflictivo, reinicia la interfaz lógica de la antena (`wlan0`) y restablece el enlace con la red local. Esta solución garantizó la disponibilidad del enlace inalámbrico de forma autónoma en cada sesión experimental, aislando el nodo maestro de ROS de las fallas físicas subyacentes del *driver*.

2.6.10. Configuración de la sensibilidad visual

El ajuste del umbral de confianza en la detección se determinó de forma experimental para mitigar las fluctuaciones de iluminación natural en exteriores y la respuesta reflectiva de la biomasa vegetal. Se fijó un umbral de confianza (*score*) mínimo de 0,45 en el modelo YOLOv8; este valor permitió mantener la detección de especímenes afectados por sombras densas o sobreexposición solar, evitando la aparición de falsos positivos inducidos por destellos lumínicos, vegetación fuera de interés o patrones de textura en la tierra. Dicho parámetro, integrado con la restricción de validación en la franja

inferior del encuadre y la temporización de inactivación de 4 s, consolidó el registro automatizado de muestras sin requerir la aplicación de técnicas de segmentación por color tradicionales.

3. Resultados

A continuación se presentan los datos y evaluaciones derivadas de las pruebas de integración física, ajuste cinemático y procesamiento de señales. Los ensayos ejecutados permiten constatar el desempeño operacional de la plataforma Hiwonder JetTank frente a restricciones de ancho de banda, irregularidades en la superficie de prueba y restricciones de compatibilidad en el ecosistema de *software*, sustentando la verificación de los requerimientos operativos para la navegación autónoma y la inspección mediante visión artificial.

3.1. Requerimientos y restricciones del sistema

El diseño, integración y validación del prototipo respondieron a un conjunto de condiciones de operación y límites físicos previamente establecidos. La Tabla X sintetiza los criterios de desempeño exigidos y las restricciones del entorno de ensayo:

Tabla X. Requerimientos y restricciones de diseño del sistema

Parámetro de evaluación	Límite / especificación operativa
<i>Condiciones requeridas de funcionamiento</i>	
Tasa de procesamiento visual	Velocidad de cómputo del módulo de visión en la placa embebida ≥ 25 FPS.
Autonomía de trabajo	Tiempo objetivo de operación continua ≥ 45 minutos.
Margen de guiado	Recorrido centrado dentro del surco, sin contacto lateral con las paredes del mismo.
Conteos duplicados	Sin registros duplicados durante las pruebas experimentales (0 %).
<i>Límites físicos del entorno y hardware</i>	
Unidad de procesamiento	Hardware base definido por la NVIDIA Jetson Nano (4 GB RAM, GPU Maxwell de 128 núcleos).
Irregularidad del terreno	Parcela de cultivo semiestructurada con elevaciones de tierra hasta 15 cm.
Entorno lumínico	Operación en campo abierto expuesta a alta radiación solar (zona semiárida).

Los criterios de procesamiento visual y estabilidad del conteo se cuantifican en las secciones siguientes. La autonomía y el margen de guiado se evaluaron de forma funcional durante las pruebas de campo completas, considerando el comportamiento del robot a lo largo de los recorridos autónomos.

3.2. Rendimiento de la telemetría y latencia

Durante las pruebas de comunicación inalámbrica iniciales, se evaluó la transmisión del flujo de video en tiempo real desde la Jetson Nano hacia la estación base utilizando el protocolo X11 sobre SSH. Se determinó que este método saturaba el ancho de banda local, generando caídas críticas en el nodo maestro de ROS. La implementación del sistema de telemetría asíncrona (`tomar_foto.py`) mediante extracción SFTP hacia el entorno de Cursor demostró una mejora significativa en la estabilidad del sistema, tal como se detalla en la Tabla XI.

Tabla XI. Comparativa de rendimiento de los protocolos de telemetría visual

Parámetro evaluado	Transmisión X11 (SSH)	Telemetría asíncrona (SFTP)
Latencia de visualización	> 2000 ms (inestable)	N/A (bajo demanda)
Consumo de ancho de banda	Alto (≈ 15 Mbps)	Bajo (< 2 Mbps durante la transferencia de capturas)
Carga en GPU (Jetson Nano)	Alta (renderizado remoto)	Mínima (escritura en disco)
Estabilidad de nodos ROS	Caídas frecuentes (<i>timeout</i>)	Estable durante las pruebas

Estos resultados justificaron la adopción definitiva del mecanismo de telemetría asíncrona durante las pruebas experimentales.

3.3. Calibración de navegación LiDAR y compensación cinemática

La separación funcional del sistema requirió calibrar el sensor LiDAR para la mitigación de perturbaciones del terreno, mientras que la adquisición visual se destinó exclusivamente al conteo fitosanitario. El sector frontal del escaneo láser se configuró con una distancia de bloqueo de 0,45 m, confirmada durante 2 s con distancias laterales inferiores a 0,85 m, con el fin de detener el chasis ante obstrucciones reales y evitar detenciones falsas por ruido del sensor.

Para la detección de fin de hilera, se consideró ausencia de pared lateral cuando la distancia medida superaba 1,80 m, siempre que además superara a la pared opuesta en al menos 0,10 m, y se definió una apertura mínima relativa de 0,20 m respecto a la pared opuesta. Esta condición se confirmaba temporalmente durante un mínimo de 0,35 s antes de ser aceptada. Para la última hilera, la parada final se confirmó con una distancia frontal menor a 0,70 m y distancias laterales menores a 1,50 m, mantenidas durante 2 s.

Los ensayos sobre suelo agrícola expusieron pérdidas de adherencia (*slip-page*) por la tracción de las orugas en tierra suelta. Para mitigar la deriva angular y los errores de posicionamiento durante el centrado, se sintonizaron las constantes del controlador proporcional-derivativo (PD) en $K_p = 3$ y $K_d = 1$. Asimismo, la lectura inercial de la IMU fijó límites de seguridad de

15° para el ángulo de cabeceo (*pitch*) y 10° para el ángulo de balanceo (*roll*), con el fin de detectar trepadas sobre los taludes del surco.

La maniobra de giro en U se estructuró en tres fases. En la fase principal de pivote (primeros 160°), la velocidad angular nominal se estableció en 0,35 rad/s con una velocidad lineal residual de 0,06 m/s; si la distancia a la pared interna del giro descendía por debajo de 0,25 m, se aplicaba un refuerzo de +0,1 rad/s (alcanzando un máximo de 0,45 rad/s) junto con un retroceso lineal de 0,05 m/s; si la pared interna se situaba entre 0,25 m y 0,40 m, la velocidad angular se reducía al 80 % del valor nominal (0,28 rad/s) para ejecutar un giro más conservador. En la fase de aproximación final (últimos 15°), la velocidad angular se calculaba de forma proporcional al ángulo restante, acotada entre 0,15 rad/s y 0,35 rad/s, con una ganancia de 1,5. El giro se consideraba completado cuando el ángulo restante era inferior a una tolerancia de 0,15 rad. El objetivo de giro se ajustó empíricamente en 175° respecto al *yaw* estimado por la odometría, frente al valor teórico de 180°, logrando una reincorporación alineada al surco contiguo sin sobre-rotaciones.

Los valores finales fueron determinados mediante un proceso iterativo de ajuste empírico basado en pruebas repetidas sobre el terreno agrícola.

3.4. Métricas de evaluación del modelo de inferencia embebido

El desempeño del sistema de visión artificial se midió en condiciones de operación en el patio de pruebas, previa carga del archivo binario `best.engine` en la memoria física de la Jetson Nano. La reestructuración multiclase y la fase de reentrenamiento con muestras sintéticas permitieron evaluar el desempeño del modelo en términos de precisión, sensibilidad, mAP@50 y tiempo de procesamiento en el dispositivo embebido.

La evaluación cuantitativa del clasificador se basó en los parámetros estándar de visión por computador. Considerando TP (verdaderos positivos) como las instancias identificadas de forma correcta, FP (falsos positivos) como las detecciones erróneas y FN (falsos negativos) como los ejemplares omitidos, las expresiones para la precisión (P) y la sensibilidad (R) responden a:

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}. \quad (9)$$

El promedio de precisión media (mAP@50) se calcula como el área bajo la curva *precision-recall*, evaluada con un umbral de intersección sobre unión (IoU) del 50 %:

$$\text{mAP@50} = \frac{1}{N} \sum_{i=1}^N AP_i, \quad \text{donde } AP_i = \int_0^1 P_i(r) dr. \quad (10)$$

Las métricas globales obtenidas se consolidan en la Tabla XII, mientras que la correspondencia de aciertos por clase se detalla en la Fig. 24.

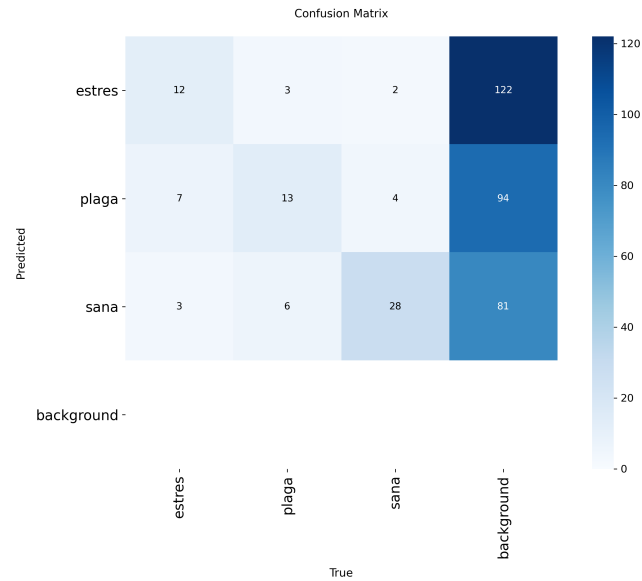


Fig. 24. Matriz de confusión del modelo optimizado multiclase `best.engine` validado en condiciones de campo. Fuente: Autoría propia.

Tabla XII. Métricas de rendimiento de la red neuronal multiclase optimizada en la Jetson Nano

Clase de cultivo	Precisión (P)	Sensibilidad (R)	mAP@50	Latencia inferencia
Clase 0 (sana)	0,889	1,000	0,978	38,95 ms
Clase 1 (plaga)	0,985	0,955	0,993	38,95 ms
Clase 2 (estrés)	0,951	1,000	0,993	38,95 ms
Promedio global	0,942	0,985	0,988	38,95 ms ($\approx 25,7$ FPS)

Los resultados obtenidos evidencian un desempeño consistente entre las tres clases evaluadas dentro del conjunto de prueba.

La red alcanzó una sensibilidad (*recall*) global del 98,5 %, lo que indica una baja tasa de omisión de instancias etiquetadas en el conjunto evaluado. El valor de $mAP@50 = 98,8 \%$ respalda un alto nivel de confiabilidad dentro de las condiciones del dataset sintético y del entorno de pruebas. Asimismo, la restricción espacial en la mitad inferior de la imagen y el periodo de inactivación temporal (*cooldown*) de 4 s redujeron las detecciones de fondo lejanas y evitaron conteos duplicados en el reporte final. El tiempo de procesamiento promedio de 38,95 ms por cuadro, equivalente a 25,7 FPS, resulta compatible con la velocidad de crucero de 0,12 m/s empleada en los recorridos autónomos.

3.5. Fusión de sensores y mitigación de interferencia solar

Durante las pruebas de campo en exteriores, bajo condiciones de alta radiación solar propias del entorno geográfico, se documentó una falla crítica en la percepción 3D: el fenómeno de ceguera infrarroja (*washout effect*). La saturación lumínica natural afectó el patrón de luz estructurada emitido por el sensor de la cámara Astra Pro Plus, devolviendo matrices de profundidad nulas o con alto grado de ruido.

Para asegurar la continuidad operativa del sistema autónomo, se adoptó una estrategia de percepción complementaria basada en sensores disponibles. La navegación se apoyó en el sensor LiDAR YDLIDAR G4, mientras que la percepción fitosanitaria se realizó sobre el canal RGB de la cámara mediante el modelo YOLOv8. El canal de profundidad fue descartado de la inferencia final debido a la saturación solar observada. Al operar bajo el principio de tiempo de vuelo (*time-of-flight*) mediante pulsos láser de banda estrecha, el LiDAR mostró robustez frente a la interferencia solar diurna, manteniendo mediciones estables para el control reactivo del robot.

3.6. Análisis de trayectorias y limitaciones físicas del terreno

Las pruebas experimentales en campo pusieron de manifiesto las restricciones cinemáticas asociadas a la tracción por orugas sobre suelo agrícola no consolidado. Para evaluar dicho comportamiento y validar la efectividad del control propuesto, se compararon tres registros de trayectoria obtenidos bajo diferentes condiciones operativas.

1. Evaluación experimental del algoritmo SLAM durante la fase inicial del desarrollo.

La Fig. 25 ilustra la rejilla de ocupación (*occupancy grid*) estimada por el algoritmo SLAM durante el desplazamiento autónomo a velocidad nominal (0,12 m/s). En lugar de definir con claridad las tres hileras del cultivo, la representación espacial sufrió una deformación lineal incoherente. Esta falla corresponde a un desacople por coincidencia de escaneo (*scan matching failure*): al ejecutar virajes en U y correcciones bruscas mediante el controlador PD, las orugas de la plataforma JetTank experimentan deslizamiento lateral sobre la tierra suelta. Dicho fenómeno introduce un desfase instantáneo entre la odometría provista por los *encoders* y la cinemática real del móvil, provocando que el algoritmo SLAM pierda la referencia geométrica y detenga la actualización coherente del mapa.

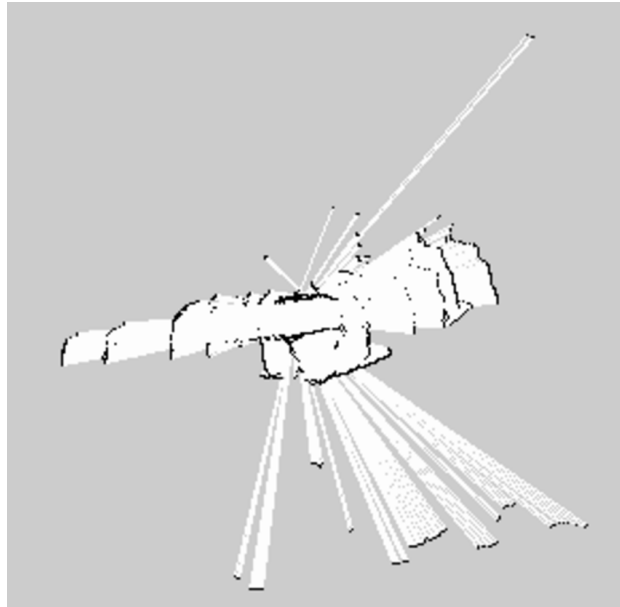


Fig. 25. Resultado obtenido durante la evaluación preliminar del algoritmo SLAM. El deslizamiento de las orugas produjo inconsistencias en la cartografía, motivo por el cual se adoptó posteriormente una estrategia de navegación reactiva basada en LiDAR. Fuente: Elaboración propia.

Los resultados obtenidos evidenciaron que, aunque el algoritmo SLAM funcionó adecuadamente en condiciones controladas, las características mecánicas del sistema de tracción por orugas y las irregularidades del terreno agrícola provocaron pérdidas de consistencia en la cartografía durante la navegación autónoma. Por esta razón, se descartó su utilización como estrategia principal

de navegación y se implementó un algoritmo reactivo basado en mediciones del sensor LiDAR, el cual demostró un comportamiento más robusto y estable en las pruebas de campo.

2. Validación de sensores (mapeo manual).

Para comprobar que la falla no correspondía a un defecto del hardware (LiDAR y *encoders*), se realizó un recorrido teleoperado a velocidad reducida. En la Fig. 26 se observa la trayectoria registrada por la odometría del robot. Al eliminar las maniobras bruscas, el deslizamiento mecánico fue mínimo, obteniendo una trayectoria en “S” consistente. Esto validó que el hardware funciona correctamente y que la distorsión evidenciada en modo autónomo corresponde principalmente a un efecto físico del terreno sobre la dinámica del chasis.

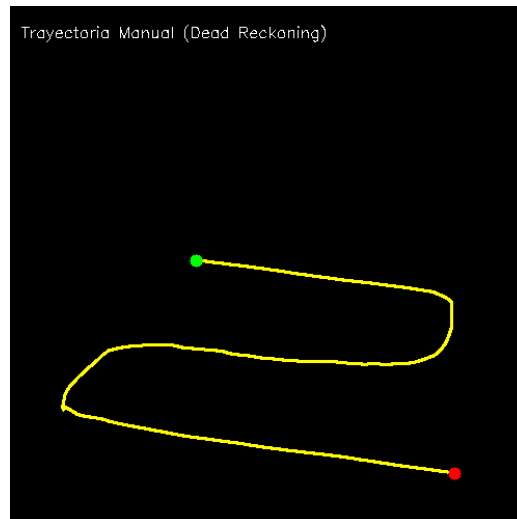


Fig. 26. Trayectoria en “S” generada mediante teleoperación lenta. La ausencia de derrape permite obtener una geometría consistente. Fuente: Elaboración propia.

3. Trazado de ruta autónoma y atenuación de deriva odométrica.

En la Fig. 27 se observa la trayectoria registrada por el robot durante la prueba autónoma. Se distingue claramente la forma de “S” al recorrer los tres surcos y realizar los dos giros. Sin embargo, en las curvas se aprecia una elongación de la trayectoria durante las maniobras de giro.

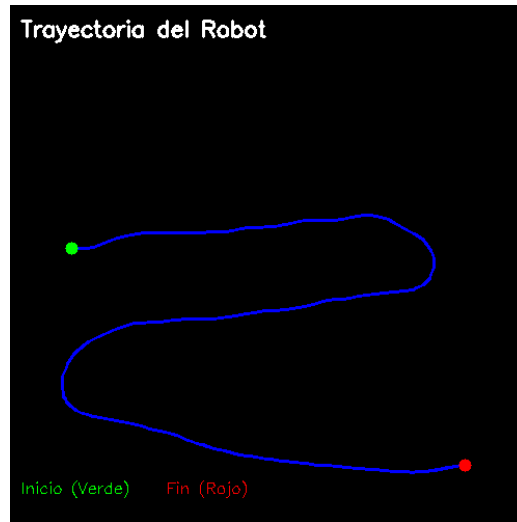


Fig. 27. Trayectoria autónoma del robot. La elongación en los giros refleja el deslizamiento físico de las orugas. Fuente: Elaboración propia.

Este comportamiento ocurre por el deslizamiento de las orugas en la tierra (*slippage*). Cuando el robot gira, el chasis se desplaza ligeramente de forma lateral, lo que introduce error en el cálculo de la posición. Dado que la odometría puede verse afectada por este fenómeno, no se empleó para el centrado lateral dentro del surco; se reservó como referencia para las maniobras de giro y para el registro de la trayectoria. De esta forma, la *ley de repulsión reactiva* del LiDAR corrigió los desvíos de forma continua durante el recorrido, permitiendo que el robot completara la misión a pesar de las condiciones del terreno.

3.7. Despliegue de la interfaz embebida

Finalmente, se validó la operatividad autónoma del sistema mediante la GUI embebida en la pantalla táctil (Fig. 28), permitiendo al operador visualizar el estado del robot, el contador total de plantas, los contadores por clase fitosanitaria y la posición odométrica estimada, sin dependencia de redes externas.

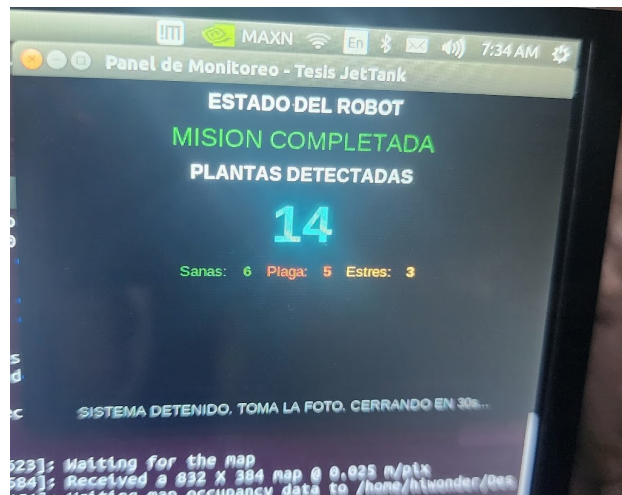


Fig. 28. Interfaz gráfica de monitoreo embebida en la pantalla táctil de la Jetson Nano. Fuente: Autoría propia.

4. Conclusiones y recomendaciones

4.1. Conclusiones

A partir del desarrollo, la integración y las evaluaciones en campo de la plataforma robótica móvil, se formulan las siguientes conclusiones:

- **Evaluación del modelo de visión en arquitectura embebida:** Se confirmó la factibilidad de ejecutar un modelo de aprendizaje profundo multiclase para la inspección fitosanitaria sobre hardware de capacidad limitada. La adaptación del modelo YOLOv8 Nano con datos de biomasa sintética impresa en 3D permitió superar la falta temporal de especímenes reales en campo. El flujo de optimización desplegado en TensorRT (`best.engine`) alcanzó una velocidad de inferencia de 25,7 FPS, obteniendo métricas cuantitativas de 94,2 % en precisión global, 98,5 % en sensibilidad y un mAP@50 de 98,8 % durante la evaluación del conjunto de prueba. Estos resultados respaldan la viabilidad del pipeline de detección y conteo sobre el entorno sintético validado, mientras que la generalización a tejido vegetal real constituye una línea de trabajo futuro.
- **Ajuste de parámetros para la navegación y la seguridad:** Los ensayos de campo determinaron que un umbral frontal de bloqueo de 0,45 m, confirmado durante 2 s con distancias laterales inferiores a 0,85 m, permitió detener oportunamente la plataforma ante obstrucciones reales. Para la supervisión de estabilidad, la IMU empleó límites de 15° para el ángulo de cabeceo (*pitch*) y 10° para el ángulo de balanceo (*roll*), con el fin de detectar condiciones de trepada sobre los taludes del surco. Al sintonizar el control PD en $K_p = 3$ y $K_d = 1$, se redujeron las oscilaciones del robot dentro del surco. La maniobra de giro en U se estructuró en tres fases con protección de la pared interna: una fase principal de pivote con velocidad nominal de 0,35 rad/s y refuerzo de hasta 0,45 rad/s cuando la pared interna descendía por debajo de 0,25 m, una condición intermedia que reducía la velocidad al 80 % del nominal cuando la pared se situaba entre 0,25 m y 0,40 m, y una fase de aproximación final con control proporcional al ángulo restante. El objetivo de giro se calibró empíricamente en 175° respecto al *yaw* estimado por la odometría, en lugar de los 180° teóricos; este ajuste compensó parcialmente el deslizamiento de las orugas sobre tierra suelta y permitió una reincorporación más adecuada a la siguiente hilera de cultivo.

- **Desempeño de la separación funcional entre navegación y percepción:** La separación funcional de responsabilidades permitió asignar roles específicos a cada sensor. El LiDAR se empleó para el seguimiento de la hilera mediante control reactivo, la IMU para la supervisión de la actitud del robot y el canal RGB de la cámara frontal para la clasificación y el conteo de plantas. El canal de profundidad fue descartado para la inferencia final debido a la saturación provocada por la radiación solar directa. Esta organización redujo la dependencia entre los módulos de navegación y percepción, favoreciendo un comportamiento estable durante las pruebas de campo. El control reactivo basado en LiDAR permitió mantener un centrado estable dentro del surco, reduciendo la dependencia de la iluminación visible. Por su parte, la IMU permitió detectar condiciones de trepada, mientras que la restricción espacial en la mitad inferior de la imagen y el periodo de inactivación temporal (*cooldown*) de 4 s evitaron conteos duplicados. El sistema mantuvo la velocidad de cruce de 0,12 m/s durante los recorridos autónomos.
- **Gestión de ancho de banda en la transmisión de datos:** Se observó que la transmisión de video en vivo mediante SSH usando el protocolo X11 consumía aproximadamente 15 Mbps de ancho de banda, lo que provocaba caídas del sistema por expiración de tiempo (*timeout*) en ROS. Al reemplazar este mecanismo por la extracción de capturas fotográficas bajo demanda mediante SFTP, se redujo el consumo de red durante la telemetría y se estabilizó la comunicación en las pruebas de depuración remota.
- **Despliegue de la interfaz de usuario local:** La implementación de una interfaz gráfica desarrollada en `Tkinter` sobre la pantalla táctil conectada a la Jetson Nano proporcionó autonomía operativa a la plataforma. Esto permitió al usuario supervisar el estado del robot, el contador acumulado de plantas, la clasificación por categorías fitosanitarias y la posición odométrica estimada directamente en campo, eliminando la dependencia de redes inalámbricas externas en sectores rurales.
- **Selección de la estrategia de navegación:** Las pruebas experimentales evidenciaron que la implementación de técnicas de localización y mapeo simultáneo (SLAM) presentó limitaciones en el entorno agrícola debido al deslizamiento de las orugas y a la pérdida de consistencia en la cartografía generada. En consecuencia, la adopción de una estrategia de navegación reactiva basada en mediciones del sensor LiDAR proporcionó un comportamiento más robusto, estable y reproducible.

para el desplazamiento autónomo entre hileras de cultivo. El SLAM se mantuvo como una herramienta de cartografía y registro, pero no como método principal de navegación.

4.2. Recomendaciones

A partir de los hallazgos y las oportunidades de mejora identificadas en la fase experimental, se plantean las siguientes directrices para trabajos futuros:

- **Evolución del middleware de comunicaciones:** Se aconseja la migración del entorno de software hacia ROS 2 (*Robot Operating System 2*). La implementación de sus políticas de calidad de servicio (QoS) y la arquitectura basada en DDS permitirán mejorar el determinismo y la confiabilidad del intercambio de mensajes entre nodos en redes inalámbricas.
- **Optimización mediante cuantización numérica:** Conviene probar la cuantización a precisión de entero de 8 bits (INT8) en la compilación con TensorRT. Dicha configuración permitirá acortar la latencia de respuesta y liberar memoria en el sistema, abriendo espacio para la ejecución de arquitecturas de percepción más complejas o para aumentar la tasa efectiva de procesamiento visual.
- **Filtro solar para la cámara:** Aunque el sistema de visión funciona correctamente durante gran parte del día, se observó que al mediodía la radiación solar directa puede saturar el sensor de la cámara. Se recomienda incorporar un sistema de protección física para la cámara frontal mediante una visera o parasol, con el propósito de reducir la incidencia directa de la radiación solar sobre el sensor. Esta modificación disminuiría la saturación de la imagen durante las horas de mayor iluminación y reduciría la necesidad de aplicar correcciones adicionales mediante procesamiento digital.
- **Mejora del sistema de percepción espacial:** En terrenos con baches o pendientes muy marcadas, el chasis puede inclinarse y el LiDAR 2D puede perder parcialmente la referencia de las paredes del surco. Para futuras versiones, se recomienda incorporar un sensor LiDAR 3D que permita una representación más completa del entorno. Si se desea mantener el sensor actual, otra alternativa es utilizar los ángulos de inclinación reportados por la IMU para compensar en software la interpretación de las distancias laterales y mejorar la estabilidad del control reactivo frente a irregularidades del terreno.

- **Ajuste del clasificador en cultivos reales:** Una vez comprobada la eficacia del clasificador con réplicas sintéticas impresas en 3D, se recomienda realizar un reajuste fino (*fine-tuning*) incorporando un conjunto representativo de fotografías de vegetación orgánica real. Este proceso ajustará los pesos de las capas convolucionales a la morfología real de las plantas sin modificar el flujo de inferencia configurado en la Jetson Nano.
- **Separación modular asíncrona del software:** Se recomienda evolucionar la implementación actual hacia una arquitectura de nodos o procesos independientes para navegación, visión e interfaz gráfica. Esta separación permitiría que la inferencia visual no condicione la frecuencia del lazo de control, mejorando la escalabilidad del sistema y facilitando la incorporación de nuevos módulos de percepción o planificación.
- **Registro georreferenciado de detecciones:** Para fortalecer el valor agronómico del sistema, se recomienda incorporar un registro persistente de las detecciones, incluyendo clase, puntaje de confianza, posición odométrica y marca de tiempo. Esta información permitiría generar reportes fitosanitarios por ubicación, facilitar la trazabilidad de las zonas afectadas y apoyar la toma de decisiones en campo.
- **Validación cuantitativa adicional y fusión sensorial:** Se recomienda cuantificar de forma explícita el error lateral de centrado, la autonomía energética efectiva y la consistencia angular durante las maniobras de giro. Asimismo, la fusión de la IMU con la odometría podría mejorar la estimación de orientación en los giros en U, reduciendo el impacto del deslizamiento de las orugas sobre la referencia de rumbo.

Referencias

- [1] Y. Gai and H. Wang, “Plant disease: A growing threat to global food security,” *Agronomy*, vol. 14, no. 8, p. 1615, 2024.
- [2] A.-K. Mahlein, “Plant disease detection by imaging sensors – parallels and specific demands for precision agriculture and plant phenotyping,” *Plant Disease*, vol. 100, no. 2, pp. 241–251, 2016.
- [3] P. González-de Santos, Á. Ribeiro, C. Fernández-Quintanilla, M. Brandstøtter, S. Tomic, and S. Pedrazzi, “Fleets of robots for environmentally-safe pest control in agriculture,” *Precision Agriculture*, vol. 18, no. 4, pp. 574–614, 2017.
- [4] R. Xu and C. Li, “A modular agricultural robotic system (MARS) for precision farming: Concept and implementation,” *Journal of Field Robotics*, vol. 39, no. 4, pp. 387–409, 2022.
- [5] P. Singh, B. Zhao, and Y. Shi, “Computer vision for site-specific weed management in precision agriculture: A review,” *Agriculture*, vol. 15, no. 21, p. 2296, 2025.
- [6] K. G. Fue, W. M. Porter, E. M. Barnes, and G. C. Rains, “An extensive review of mobile agricultural robotics for field operations: Focus on cotton harvesting,” *AgriEngineering*, vol. 2, no. 1, pp. 150–174, 2020.
- [7] Y. Bai, B. Zhang, N. Xu, J. Zhou, J. Shi, and Z. Diao, “Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review,” *Computers and Electronics in Agriculture*, vol. 205, p. 107584, 2023.
- [8] W. Hua, Z. Zhang, W. Zhang, X. Liu, C. Hu, Y. He *et al.*, “Key technologies in apple harvesting robot for standardized orchards: A comprehensive review of innovations, challenges, and future directions,” *Computers and Electronics in Agriculture*, vol. 235, p. 110343, 2025.
- [9] R. Guebsi, S. Mami, and K. Chokmani, “Drones in precision agriculture: A comprehensive review of applications, technologies, and challenges,” *Drones*, vol. 8, no. 11, p. 686, 2024.
- [10] D. Tiozzo Fasiolo, L. Scalera, E. Maset, and A. Gasparetto, “Towards autonomous mapping in agriculture: A review of supportive technologies for ground robotics,” *Robotics and Autonomous Systems*, vol. 169, p. 104514, 2023.

- [11] A. Boyle, R. Cui, W. Dougherty, and P. Rudawski, “Accessible framework for automation in precision agriculture leveraging Boston Dynamics’ SPOT,” in *2023 IEEE MIT Undergraduate Research Technology Conference (URTC)*, Cambridge, MA, USA, 2023, pp. 1–5.
- [12] M. Ouaarous, K. El Fakhouri, N. Taarji, A. Baouchi, M. Amri, C. Ramdani, M. Sobeh, A. Mesfioui, and M. El Bouhssini, “Impact of field insect pests on seed and nutritional quality of some important crops: A comprehensive review,” *ACS Omega*, vol. 10, no. 9, pp. 8779–8792, Feb. 2025.
- [13] Koppert Biological Systems, “Pulgones: Plagas en plantas,” 2024, [En línea]. Disponible: <https://www.koppert.ec/plagas-en-plantas/pulgones/>. [Acceso: 29 de abril de 2024].
- [14] —, “Mosca blanca: Plagas en plantas,” 2024, [En línea]. Disponible: <https://www.koppert.ec/plagas-en-plantas/moscas-blancas/mosca-blanca/>. [Acceso: 29 de abril de 2024].
- [15] —, “Trips: Plagas en plantas,” 2024, [En línea]. Disponible: <https://www.koppert.ec/plagas-en-plantas/trips/>. [Acceso: 29 de abril de 2024].
- [16] —, “Minadores de hojas: Plagas en plantas,” 2024, [En línea]. Disponible: <https://www.koppert.ec/plagas-en-plantas/minadores-de-hojas/>. [Acceso: 29 de abril de 2024].
- [17] M. Rocha, C. L. Junior, R. S. Oliveira, and C. L. N. Junior, “Internet of Things meets computer vision to make an intelligent pest monitoring network,” *Applied Sciences*, vol. 12, no. 18, p. 9397, 2022.
- [18] C. Xie, R. Wang, J. Zhang, P. Chen, W. Dong, R. Li, T. Chen, and H. Chen, “Multi-level learning features for automatic classification of field crop pests,” *Computers and Electronics in Agriculture*, vol. 152, pp. 233–241, 2018.
- [19] A. Kamilaris and F. X. Prenafeta-Boldú, “Deep learning in agriculture: A survey,” *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.
- [20] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [21] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

- [22] J. R. Figueroa Olmedo, W. M. Montalvo López, and M. M. Bayas Sampedro, *Cinemática y Dinámica de Robots Móviles con Ruedas*, 1st ed. Sangolquí, Ecuador: Universidad de las Fuerzas Armadas ESPE, 2023.
- [23] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.
- [24] R. Pertierra Lazo and J. Quispe Gonzabay, “Análisis económico de lechugas hidropónicas bajo sistema raíz flotante en clima semiárido,” *La Granja: Revista de Ciencias de la Vida*, vol. 31, no. 1, pp. 118–130, 2020.
- [25] J. Vera, “Análisis de los factores causantes de la morosidad en los créditos agrícolas en la provincia de santa elena,” Master’s thesis, Escuela Superior Politécnica del Litoral (ESPOL), Guayaquil, Ecuador, 2015.
- [26] S. Savary, L. Willocquet, S. J. Pethybridge, P. Esker, N. McRoberts, and A. Nelson, “The global burden of pathogens and pests on major food crops,” *Nature Ecology & Evolution*, vol. 3, no. 3, pp. 430–439, 2019.
- [27] N. C. Eli-Chukwu, “Applications of artificial intelligence in agriculture: A review,” *Engineering, Technology & Applied Science Research*, vol. 9, no. 4, pp. 4377–4383, 2019.
- [28] S. Fountas, N. Mylonas, I. Malounas, E. Rodias, C. Hellmann Santos, and E. Pekkeriet, “Agricultural robotics for field operations,” *Sensors*, vol. 20, no. 9, p. 2672, 2020.
- [29] A. Walter, R. Finger, R. Huber, and N. Buchmann, “Smart farming is key to developing sustainable agriculture,” *Proceedings of the National Academy of Sciences*, vol. 114, no. 24, pp. 6148–6150, 2017.
- [30] D. C. Rose and J. Chilvers, “Agriculture 4.0: Broadening responsible innovation in an era of smart farming,” *Frontiers in Sustainable Food Systems*, vol. 2, p. 87, 2018.
- [31] Hiwonder, “JetTank: AI vision robot tank powered by NVIDIA Jetson Nano,” 2024, [En línea]. Disponible: <https://www.hiwonder.com/products/jettank>. [Acceso: 13 de mayo de 2024].
- [32] NVIDIA Corporation, “Jetson Nano Developer Kit: Product development,” 2024, [En línea]. Disponible: <https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/>. [Acceso: 13 de mayo de 2024].

Anexos

Anexo A: Código de navegación autónoma y arquitectura desacoplada

Script principal en Python 3 ejecutado en la NVIDIA Jetson Nano. Integra la lectura del LiDAR, la IMU, el control PD, la máquina de estados de giro en U, el conteo visual mediante Tripwire y la GUI en Tkinter.

```
1  #!/usr/bin/env python3
2  import os
3  import sys
4  import time
5  import math
6  import cv2
7  import numpy as np
8  import tensorrt as trt
9  import pycuda.driver as cuda
10 import pycuda.autoinit
11 import tkinter as tk
12
13 from collections import deque
14
15 # Librerías nativas de ROS 1
16 import rospy
17 from geometry_msgs.msg import Twist
18 from std_msgs.msg import Float64
19 from nav_msgs.msg import Odometry
20 from sensor_msgs.msg import LaserScan, Imu
21
22
23 # =====
24 # CONFIGURACION DE NAVEGACION, VISION Y SEGURIDAD
25 # =====
26
27 ENGINE_PATH = "/home/hiwonder/jettank_ws/src/src/
    jettank_control/jettank_control/best.engine"
28 CAMERA_INDEX = 0
29
30 VEL_LINEAL_BUSQUEDA = 0.12
31 VEL_ANGULAR_MAX = 0.4
32 VEL_GIRO_U = 0.35
33 VEL_PIVOTE = 0.06
34
35 # Índice central del LiDAR
36 IDX_CENTRO = 377
37
38 # Control PD lateral
```

```

39 KP_PARED = 3.0
40 KD_PARED = 1.0
41
42 # Distancia minima valida del LiDAR.
43 UMBRAL_CHASIS = 0.10
44
45 # Distancia maxima considerada para paredes/obstaculos.
46 DIST_MAXIMA_LIDAR = 2.0
47
48 # Distancia frontal para detectar fin de fila en ultima
    hilera.
49 # Se deja por compatibilidad, pero la parada final usa los
    umbrales de mision.
50 DIST_FIN_FILA_FRONTAL = 0.70
51
52 # =====
53 # UMBRALES DE DETENCION POR BLOQUEO
54 # =====
55
56 # Distancia frontal para considerar bloqueo real durante
    navegacion.
57 DIST_FRENTE_BLOQUEO = 0.45
58
59 # Distancia lateral para considerar bloqueo real durante
    navegacion.
60 DIST_LATERAL_BLOQUEO = 0.85
61
62 # Distancia frontal para detenerse al final de la mision.
63 DIST_FRENTE_FIN_MISION = 0.70
64
65 # Distancia lateral para detenerse al final de la mision.
66 DIST_LATERAL_FIN_MISION = 1.50
67
68 # Distancia maxima para considerar que existe una pared
    lateral visible.
69 DIST_PARED_VISIBLE = 1.50
70
71 # Distancia deseada a una pared cuando solo se ve una pared.
72 DIST_OBJETIVO_LATERAL = 0.35
73
74 # =====
75 # DETECCION HIBRIDA DE FIN DE FILA
76 # =====
77
78 # Distancia a partir de la cual consideramos que una pared
    lateral esta ausente.
79 DIST_PARED_AUSENTE = 1.80
80

```

```

81 # Diferencia minima entre la pared que se abre y la pared
    presente.
82 DIFERENCIA_APERTURA = 0.20
83
84 # Angulo de yaw necesario para confirmar que el robot empezo
    a girar.
85 ANGULO_ACTIVACION_GIRO = math.radians(15)
86
87 # Tiempo minimo de apertura antes de aceptar yaw como valido.
88 TIEMPO_APERTURA_MINIMA = 0.35
89
90 # Tiempo minimo de yaw en direccion correcta.
91 TIEMPO_YAW_MINIMO = 0.15
92
93 # Tiempo de confirmacion para parada final / fin de fila.
94 TIEMPO_CONFIRMACION_FIN_FILA = 2.0
95
96 # Velocidad angular cuando se detecta apertura y el robot
    empieza a fugarse.
97 VEL_APERTURA = 0.40
98
99 # Velocidad lineal mientras empieza a fugarse hacia el lado
    abierto.
100 VEL_APERTURA_LINEAL = VEL_LINEAL_BUSQUEDA * 0.65
101
102 # IMU
103 UMBRAL_INCLINACION = math.radians(15)
104 MAX_DESVIACION_YAW = math.radians(30)
105
106 # Compensacion de inercia en giro U
107 ANGULO_OBJETIVO = math.radians(175)
108
109 # Vision
110 SCORE_MINIMO = 0.45
111 TIEMPO_ENFRIAMIENTO = 4.0
112 Y_LIMIT_INF = 240
113
114 # Angulos camara PTZ
115 CAM_ANG_DER = 0.785
116 CAM_ANG_IZQ = -0.785
117
118 # =====
119 # TIEMPOS DE SEGURIDAD
120 # =====
121
122 # Tiempo que debe mantenerse bloqueo frontal + laterales para
    detenerse.
123 TIEMPO_BLOQUEO = 2.0
124

```

```

125 # Tiempo minimo que debe haber estado en una fila antes de
    aceptar fin de fila.
126 TIEMPO_MINIMO_EN_FILA = 3.0
127
128 # Si se quiere que la mision pare al terminar todas las
    hileras, dejar True.
129 # Si se quiere modo estricto: unica parada por bloqueo
    frontal/lateral,
130 # poner False. En ese caso el robot seguira hasta bloqueo,
    cerrar tactilmente la terminal o Ctrl+C.
131 PARADA_POR_FIN_MISION = True
132
133
134 # =====
135 # 1. INICIALIZAR LA CAMARA PRIMERO CON V4L2
136 # =====
137 print("Iniciando camara con backend V4L2...")
138 cap = cv2.VideoCapture(CAMERA_INDEX + cv2.CAP_V4L2)
139 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
140 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
141 cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
142
143 if not cap.isOpened():
144     print("ERROR: No se pudo abrir la camara. Abortando.")
145     sys.exit(1)
146
147 print("Camara lista.")
148
149
150 # =====
151 # 2. INICIALIZACION DE MOTOR TENSORRT
152 # =====
153 print("Cargando TensorRT para conteo y clasificacion...")
154 TRT_LOGGER = trt.Logger(trt.Logger.WARNING)
155
156 with open(ENGINE_PATH, "rb") as f, trt.Runtime(TRT_LOGGER) as
    runtime:
157     engine = runtime.deserialize_cuda_engine(f.read())
158
159 context = engine.create_execution_context()
160
161 inputs, outputs, allocations = [], [], []
162
163 for binding in engine:
164     shape = engine.get_binding_shape(binding)
165     size = trt.volume(shape)
166     dtype = trt.nptype(engine.get_binding_dtype(binding))
167     allocation = cuda.mem_alloc(size * np.dtype(dtype).
        itemsize)

```

```

168     allocations.append(int(allocation))
169
170     binding_dict = {
171         'shape': shape,
172         'dtype': dtype,
173         'allocation': allocation
174     }
175
176     if engine.binding_is_input(binding):
177         inputs.append(binding_dict)
178     else:
179         outputs.append(binding_dict)
180
181     print("Esperando 5s para que el bus USB y el LiDAR se
182           recuperen...")
183     time.sleep(5.0)
184
185     # =====
186     # CONFIGURACION DEL NODO ROS Y SENSORES
187     # =====
188     rospy.init_node('navegacion_lidar_cultivos', anonymous=True)
189
190     pub_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
191     pub_pan = rospy.Publisher('/joint1_controller/command',
192                               Float64, queue_size=1)
193     pub_tilt = rospy.Publisher('/joint2_controller/command',
194                                Float64, queue_size=1)
195
196     vel_cmd = Twist()
197
198     # Odometria
199     current_yaw = 0.0
200     odom_received = False
201     trayectoria_puntos = []
202     pos_x_calc = 0.0
203     pos_y_calc = 0.0
204
205     def calcular_yaw(q):
206         siny_cosp = 2.0 * (q.w * q.z + q.x * q.y)
207         cosy_cosp = 1.0 - 2.0 * (q.y * q.y + q.z * q.z)
208         return math.atan2(siny_cosp, cosy_cosp)
209
210     def odom_callback(msg):
211         global current_yaw, odom_received, pos_x_calc, pos_y_calc
212         current_yaw = calcular_yaw(msg.pose.pose.orientation)
213         odom_received = True

```

```

214     pos_x_calc = msg.pose.pose.position.x
215     pos_y_calc = msg.pose.pose.position.y
216
217
218 # IMU
219 current_pitch = 0.0
220 current_roll = 0.0
221 imu_received = False
222
223
224 def imu_callback(msg):
225     global current_pitch, current_roll, imu_received
226
227     q = msg.orientation
228
229     sinp = 2.0 * (q.w * q.y - q.z * q.x)
230     if abs(sinp) >= 1:
231         current_pitch = math.copysign(math.pi / 2, sinp)
232     else:
233         current_pitch = math.asin(sinp)
234
235     sinr_cosp = 2.0 * (q.w * q.x + q.y * q.z)
236     cosr_cosp = 1.0 - 2.0 * (q.x * q.x + q.y * q.y)
237     current_roll = math.atan2(sinr_cosp, cosr_cosp)
238
239     imu_received = True
240
241
242 # LiDAR
243 dist_izq = 10.0
244 dist_der = 10.0
245 dist_frontal = 10.0
246 scan_received = False
247
248 # Historiales para suavizar lecturas LiDAR.
249 hist_frontal = deque(maxlen=3)
250 hist_izq = deque(maxlen=3)
251 hist_der = deque(maxlen=3)
252
253
254 def distancia_robusta(vals, percentil=15):
255     """
256     Devuelve una distancia robusta frente a ruido.
257     """
258     limpio = []
259
260     for r in vals:
261         if math.isfinite(r) and UMBRAL_CHASIS < r <
DIST_MAXIMA_LIDAR:

```

```

262         limpio.append(r)
263
264     if len(limpio) == 0:
265         return 10.0
266
267     limpio.sort()
268
269     if len(limpio) <= 2:
270         return limpio[0]
271
272     idx = int(round((percentil / 100.0) * (len(limpio) - 1)))
273     return limpio[idx]
274
275
276 def scan_callback(msg):
277     global dist_izq, dist_der, dist_frontal, scan_received
278
279     puntos = len(msg.ranges)
280
281     def sector(start_idx, end_idx):
282         return [msg.ranges[i % puntos] for i in range(
283             start_idx, end_idx)]
284
285     f_raw = distancia_robusta(sector(IDX_CENTRO - 25,
286                                     IDX_CENTRO + 25), 10)
287     l_raw = distancia_robusta(sector(IDX_CENTRO + 40,
288                                     IDX_CENTRO + 100), 15)
289     r_raw = distancia_robusta(sector(IDX_CENTRO - 100,
290                                     IDX_CENTRO - 40), 15)
291
292     hist_frontal.append(f_raw)
293     hist_izq.append(l_raw)
294     hist_der.append(r_raw)
295
296     dist_frontal = float(np.median(hist_frontal))
297     dist_izq = float(np.median(hist_izq))
298     dist_der = float(np.median(hist_der))
299
300     scan_received = True
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

307 print("Mecanismo PTZ calibrado. Camara apuntando a la derecha
      (Carril 1).")
308
309
310 # =====
311 # 3. INTERFAZ GRAFICA (GUI)
312 # =====
313 root = tk.Tk()
314 root.title("Panel de Monitoreo - Tesis JetTank")
315 root.geometry("480x320")
316 root.configure(bg="black")
317
318 estado_txt = tk.StringVar(value="Iniciando...")
319 total_txt = tk.StringVar(value="0")
320 sanas_txt = tk.StringVar(value="0")
321 plagas_txt = tk.StringVar(value="0")
322 estres_txt = tk.StringVar(value="0")
323 pos_txt = tk.StringVar(value="Esperando odometria...")
324
325 tk.Label(
326     root,
327     text="ESTADO DEL ROBOT",
328     font=("Helvetica", 12, "bold"),
329     bg="black",
330     fg="white"
331 ).pack(pady=5)
332
333 lbl_estado = tk.Label(
334     root,
335     textvariable=estado_txt,
336     font=("Helvetica", 16),
337     bg="black",
338     fg="blue"
339 )
340 lbl_estado.pack()
341
342 tk.Label(
343     root,
344     text="PLANTAS DETECTADAS",
345     font=("Helvetica", 12, "bold"),
346     bg="black",
347     fg="white"
348 ).pack(pady=5)
349
350 tk.Label(
351     root,
352     textvariable=total_txt,
353     font=("Helvetica", 36, "bold"),
354     bg="black",

```

```

355     fg="cyan"
356 ).pack()
357
358 f_counts = tk.Frame(root, bg="black")
359 f_counts.pack(pady=5)
360
361 tk.Label(f_counts, text="Sanas:", font=("Helvetica", 10), bg=
    "black", fg="green").grid(row=0, column=0, padx=5)
362 tk.Label(f_counts, textvariable=sanas_txt, font=("Helvetica",
    10, "bold"), bg="black", fg="green").grid(row=0, column
    =1, padx=5)
363
364 tk.Label(f_counts, text="Plaga:", font=("Helvetica", 10), bg=
    "black", fg="red").grid(row=0, column=2, padx=5)
365 tk.Label(f_counts, textvariable=plagas_txt, font=("Helvetica"
    , 10, "bold"), bg="black", fg="red").grid(row=0, column=3,
    padx=5)
366
367 tk.Label(f_counts, text="Estres:", font=("Helvetica", 10), bg
    ="black", fg="orange").grid(row=0, column=4, padx=5)
368 tk.Label(f_counts, textvariable=estres_txt, font=("Helvetica"
    , 10, "bold"), bg="black", fg="orange").grid(row=0, column
    =5, padx=5)
369
370 tk.Label(root, textvariable=pos_txt, font=("Helvetica", 10),
    bg="black", fg="gray").pack(side=tk.BOTTOM, pady=5)
371
372 root.update()
373
374
375 # =====
376 # FUNCIONES DE EMERGENCIA, LIMPIEZA Y GRAFICADO
377 # =====
378 def apagar_sistema_seguro():
379     print("Mision finalizada. Deteniendo motores...")
380
381     vel_cmd.linear.x = 0.0
382     vel_cmd.angular.z = 0.0
383     pub_vel.publish(vel_cmd)
384
385     pub_pan.publish(Float64(0.0))
386     print("Camara centrada.")
387
388     if len(trayectoria_puntos) > 2:
389         print("Dibujando y guardando trayectoria...")
390
391         img = np.zeros((500, 500, 3), dtype=np.uint8)
392
393         xs = [p[0] for p in trayectoria_puntos]

```

```

394         ys = [p[1] for p in trayectoria_puntos]
395
396         min_x, max_x = min(xs), max(xs)
397         min_y, max_y = min(ys), max(ys)
398
399         scale = 400 / max(max_x - min_x + 0.1, max_y - min_y
400 + 0.1)
401
402         pts = []
403         for x, y in trayectoria_puntos:
404             px = int(50 + (x - min_x) * scale)
405             py = int(450 - (y - min_y) * scale)
406             pts.append([px, py])
407
408         pts_np = np.array(pts, np.int32)
409
410         cv2.polylines(img, [pts_np], False, (255, 0, 0), 2)
411         cv2.circle(img, pts[0], 6, (0, 255, 0), -1)
412         cv2.circle(img, pts[-1], 6, (0, 0, 255), -1)
413         cv2.putText(img, "Trayectoria del Robot", (10, 30),
414 cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
415
416         cv2.imwrite('/home/hiwonder/Desktop/trayectoria.png',
417 img)
418         print("Trayectoria guardada.")
419
420         print("Guardando mapa SLAM...")
421         os.system("timeout 15 rosrn map_server map_saver -f /
422 home/hiwonder/Desktop/mapa_cultivo map:=/robot_1/map")
423
424         print("Cerrando SLAM/LIDAR...")
425         os.system("pkill -f slam")
426         os.system("pkill -f ydlidar")
427         os.system("pkill -f gmapping")
428         os.system("pkill -f hector")
429         os.system("pkill -f map_saver")
430
431         estado_txt.set("MISION COMPLETADA")
432         lbl_estado.config(fg="green")
433         pos_txt.set("SISTEMA DETENIDO. Registro disponible
434 durante 30 segundos...")
435         root.update()
436
437         print("Registro disponible durante 30 segundos...")
438         time.sleep(30.0)
439
440         # =====
441         # TIMEOUT DE SENSORES

```

```

438 # =====
439 print("Esperando Odometria, IMU y LiDAR (Timeout 15s)...")
440
441 timeout_start = time.time()
442
443 while not rospy.is_shutdown() and (not odom_received or not
    scan_received or not imu_received) and (time.time() -
    timeout_start) < 15.0:
444     time.sleep(0.5)
445
446 if not odom_received or not scan_received or not imu_received
    :
447     print("ERROR: Faltan sensores. Abortando script.")
448     apagar_sistema_seguro()
449     sys.exit(1)
450 else:
451     print("Sensores listos. LiDAR, IMU y Odometria operativos
    !")
452
453
454 # =====
455 # MAQUINA DE ESTADOS
456 # =====
457 estado_actual = "NAVEGANDO_FILA"
458 hila_actual = 1
459 total_hileras = 3
460
461 yaw_inicio_fila = 0.0
462 objetivo_yaw_giro = 0.0
463 direccion_giro = 1
464 tolerancia_yaw = 0.15
465
466 error_previo = 0.0
467
468 contador_plantas = 0
469 sanas = 0
470 plagas = 0
471 estres = 0
472
473 tiempo_ultimo_conteo = 0.0
474 deteccion_activada = True
475
476 # Timers de seguridad
477 t_bloqueo = 0.0
478 t_apertura = 0.0
479 t_yaw_giro = 0.0
480 t_inicio_fila = time.time()
481 t_entrada_fila = 0.0
482

```

```

483 # Bandera para modo estricto si PARADA_POR_FIN_MISION = False
484 mision_finalizada = False
485
486
487 def normalizar_angulo(angulo):
488     while angulo > math.pi:
489         angulo -= 2 * math.pi
490     while angulo < -math.pi:
491         angulo += 2 * math.pi
492     return angulo
493
494
495 yaw_inicio_fila = current_yaw
496 t_inicio_fila = time.time()
497
498 print("JetTank listo! Iniciando recorrido...")
499
500 rate = rospy.Rate(30)
501
502
503 # =====
504 # BUCLE PRINCIPAL
505 # =====
506 try:
507     while not rospy.is_shutdown() and cap.isOpened():
508         ret, frame = cap.read()
509
510         if not ret:
511             break
512
513         h_orig, w_orig, _ = frame.shape
514
515         # =====
516         # INFERENCIA VISUAL
517         # =====
518         if deteccion_activada:
519             input_img = cv2.resize(frame, (512, 512))
520             input_img = cv2.cvtColor(input_img, cv2.
COLOR_BGR2RGB).astype(np.float32) / 255.0
521             input_img = input_img.transpose(2, 0, 1)
522             input_img = np.ascontiguousarray(np.expand_dims(
input_img, axis=0))
523
524             cuda.memcpy_htod(inputs[0]['allocation'],
input_img)
525             context.execute_v2(allocations)
526
527             output_data = np.zeros(outputs[0]['shape'], dtype
=output_data['dtype'])

```

```

528         cuda.memcpy_dtoh(output_data, outputs[0]['
allocation'])
529
530         output = np.squeeze(output_data).T
531
532         deteccion_actual_valida = False
533         clase_actual = -1
534         max_score_actual = 0.0
535
536         for row in output:
537             scores = row[4:]
538
539             if len(scores) == 0:
540                 continue
541
542             max_score = np.max(scores)
543
544             if max_score > SCORE_MINIMO and max_score >
max_score_actual:
545                 cls = int(np.argmax(scores))
546
547                 xc, w = row[0], row[2]
548                 yc, h = row[1], row[3]
549
550                 y1 = int((yc - h / 2) * (h_orig / 512.0))
551                 target_h = int(h * (h_orig / 512.0))
552                 y_bottom = y1 + target_h
553
554                 if y_bottom > Y_LIMIT_INF:
555                     max_score_actual = max_score
556                     deteccion_actual_valida = True
557                     clase_actual = cls
558
559                 if deteccion_actual_valida:
560                     if time.time() - tiempo_ultimo_conteo >
TIEMPO_ENFRIAMIENTO:
561                         contador_plantas += 1
562                         tiempo_ultimo_conteo = time.time()
563
564                     if clase_actual == 0:
565                         sanas += 1
566                     elif clase_actual == 1:
567                         plagas += 1
568                     elif clase_actual == 2:
569                         estres += 1
570
571                     print(f"Planta {contador_plantas}
detectada! Clase: {clase_actual} (Score: {max_score_actual
:.2f})")

```

```

572
573     # =====
574     # SEGURIDAD Y CONDICIONES BASICAS
575     # =====
576     esta_escalando = abs(current_pitch) >
577     UMBRAL_INCLINACION or abs(current_roll) > math.radians(10)
578     t_now = time.time()
579
580     yaw_relativo = normalizar_angulo(current_yaw -
581     yaw_inicio_fila)
582
583     # Visibilidad de paredes
584     izq_visible = dist_izq < DIST_PARED_VISIBLE
585     der_visible = dist_der < DIST_PARED_VISIBLE
586     paredes_visibles = izq_visible or der_visible
587
588     # =====
589     # BLOQUEO REAL: FRONTAL + LATERALES
590     # =====
591     frontal_cerca = dist_frontal < DIST_FRENTE_BLOQUEO
592     izq_cerca = dist_izq < DIST_LATERAL_BLOQUEO
593     der_cerca = dist_der < DIST_LATERAL_BLOQUEO
594
595     bloqueado = frontal_cerca and izq_cerca and der_cerca
596
597     if bloqueado:
598         if t_bloqueo == 0.0:
599             t_bloqueo = t_now
600
601         if (t_now - t_bloqueo) >= TIEMPO_BLOQUEO:
602             print("BLOQUEO CONFIRMADO: pared frontal y
603             laterales por mas de 2 segundos.")
604             print("Deteniendo el robot de forma segura.")
605
606             vel_cmd.linear.x = 0.0
607             vel_cmd.angular.z = 0.0
608             pub_vel.publish(vel_cmd)
609
610             break
611         else:
612             t_bloqueo = 0.0
613
614     # =====
615     # DETECCION HIBRIDA DE FIN DE FILA
616     # =====
617     apertura_instantanea = False
618     apertura_confirmada = False
619     yaw_en_direccion = False
620     yaw_giro_confirmado = False

```

```

618         giro_esperado = 0
619         fin_fila_confirmada = False
620
621         if estado_actual == "NAVEGANDO_FILA" and not
622         mision_finalizada:
623
624             if hila_actual == 1:
625                 # Hilera 1: giro a la derecha.
626                 # La pared derecha debe abrirse respecto a la
627                 izquierda.
628                 giro_esperado = -1
629
630                 apertura_instantanea = (
631                     izq_visible and
632                     (
633                         dist_der > dist_izq +
634                         DIFERENCIA_APERTURA or
635                         (dist_der > DIST_PARED_AUSENTE and
636                         dist_der > dist_izq + 0.10)
637                     )
638                 )
639
640                 yaw_en_direccion = yaw_relativo < -
641                 ANGULO_ACTIVACION_GIRO
642
643             elif hila_actual == 2:
644                 # Hilera 2: giro a la izquierda.
645                 # La pared izquierda debe abrirse respecto a
646                 la derecha.
647                 giro_esperado = 1
648
649                 apertura_instantanea = (
650                     der_visible and
651                     (
652                         dist_izq > dist_der +
653                         DIFERENCIA_APERTURA or
654                         (dist_izq > DIST_PARED_AUSENTE and
655                         dist_izq > dist_der + 0.10)
656                     )
657                 )
658
659                 yaw_en_direccion = yaw_relativo >
660                 ANGULO_ACTIVACION_GIRO
661
662             else:
663                 # Ultima hilera: no debe hacer otro giro en U
664                 .
665                 # Debe detenerse si tiene las 3 paredes
666                 durante 2 segundos.

```

```

656         giro_esperado = 0
657
658         pared_fin_mision = (
659             dist_frontal < DIST_FRENTE_FIN_MISION and
660             dist_izq < DIST_LATERAL_FIN_MISION and
661             dist_der < DIST_LATERAL_FIN_MISION
662         )
663
664         apertura_instantanea = pared_fin_mision
665         yaw_en_direccion = False
666
667         if apertura_instantanea and not bloqueado:
668             if t_apertura == 0.0:
669                 t_apertura = t_now
670
671                 apertura_confirmada = (t_now - t_apertura) >=
TIEMPO_APERTURA_MINIMA
672
673             else:
674                 t_apertura = 0.0
675                 apertura_confirmada = False
676
677             if apertura_confirmada and yaw_en_direccion:
678                 if t_yaw_giro == 0.0:
679                     t_yaw_giro = t_now
680
681                 yaw_giro_confirmado = (t_now - t_yaw_giro) >=
TIEMPO_YAW_MINIMO
682
683             else:
684                 t_yaw_giro = 0.0
685                 yaw_giro_confirmado = False
686
687             if apertura_confirmada:
688                 tiempo_apertura = t_now - t_apertura
689
690                 if hila_actual >= total_hileras:
691                     fin_fila_confirmada = tiempo_apertura >=
TIEMPO_CONFIRMACION_FIN_FILA
692                 else:
693                     # La condicion principal es yaw +
apertura.
694                     # Si el yaw no llega, igualmente se
confirma por apertura prolongada.
695                     fin_fila_confirmada = (
696                         yaw_giro_confirmado or
697                         tiempo_apertura >=
TIEMPO_CONFIRMACION_FIN_FILA
698                     )

```

```

699         else:
700             fin_fila_confirmada = False
701
702             # Descomentar para depurar:
703             # print(f"[CHECK] h={hila_actual} YAW={math.
degrees(yaw_relativo):.1f} I={dist_izq:.2f} D={dist_der:.2
f} F={dist_frontal:.2f} AP={apertura_instantanea} FIN={
fin_fila_confirmada}")
704
705         else:
706             t_apertura = 0.0
707             t_yaw_giro = 0.0
708             apertura_confirmada = False
709             yaw_giro_confirmado = False
710             fin_fila_confirmada = False
711
712             # =====
713             # MAQUINA DE ESTADOS
714             # =====
715             if estado_actual == "NAVEGANDO_FILA":
716                 deteccion_activada = True
717
718                 if esta_escalando:
719                     t_apertura = 0.0
720                     t_yaw_giro = 0.0
721
722                     vel_cmd.linear.x = -0.06
723
724                     if abs(current_roll) > math.radians(5):
725                         vel_cmd.angular.z = current_roll * 5.0
726                     else:
727                         vel_cmd.angular.z = 0.0
728
729                 elif fin_fila_confirmada and (t_now -
t_inicio_fila) >= TIEMPO_MINIMO_EN_FILA and not
mision_finalizada:
730
731                     if hila_actual >= total_hileras:
732                         if PARADA_POR_FIN_MISION:
733                             print("Mision Completada. Todas las
hileras recorridas.")
734                             print("Condicion de parada final por
3 paredes confirmada.")
735
736                         vel_cmd.linear.x = 0.0
737                         vel_cmd.angular.z = 0.0
738                         pub_vel.publish(vel_cmd)
739
740                         break

```

```

741         else:
742             print("Mision finalizada, pero
PARADA_POR_FIN_MISION=False.")
743             print("El robot no se detiene por fin
de mision. Solo parara por bloqueo o Ctrl+C.")
744             mision_finalizada = True
745
746             t_apertura = 0.0
747             t_yaw_giro = 0.0
748
749             vel_cmd.linear.x =
VEL_LINEAL_BUSQUEDA
750             vel_cmd.angular.z = 0.0
751
752         else:
753             estado_actual = "PREPARANDO_GIRO"
754
755             t_apertura = 0.0
756             t_yaw_giro = 0.0
757
758         else:
759             # Si hay apertura en la hilera correcta, se
deja que el robot
760             # empiece a girar hacia el lado abierto.
761             if apertura_instantanea and giro_esperado !=
0 and not bloqueado:
762                 vel_cmd.linear.x = VEL_APERTURA_LINEAL
763                 vel_cmd.angular.z = giro_esperado *
VEL_APERTURA
764                 error_previo = 0.0
765
766             elif paredes_visibles:
767                 if izq_visible and der_visible:
768                     # Ambas paredes visibles: centrado
clasico.
769                     error_actual = (dist_izq - dist_der)
770                     / 2.0
771
772                 elif izq_visible:
773                     # Solo pared izquierda visible.
774                     error_actual = dist_izq -
DIST_OBJETIVO_LATERAL
775
776                 else:
777                     # Solo pared derecha visible.
778                     error_actual = DIST_OBJETIVO_LATERAL
- dist_der

```

```

779             # Filtro simple del error para reducir
ruido.
780             error_actual = 0.7 * error_previo + 0.3 *
error_actual
781
782             d_error = error_actual - error_previo
783             error_previo = error_actual
784
785             vel_cmd.linear.x = VEL_LINEAL_BUSQUEDA
786             vel_cmd.angular.z = (KP_PARED *
error_actual) + (KD_PARED * d_error)
787
788             if vel_cmd.angular.z > VEL_ANGULAR_MAX:
789                 vel_cmd.angular.z = VEL_ANGULAR_MAX
790
791             if vel_cmd.angular.z < -VEL_ANGULAR_MAX:
792                 vel_cmd.angular.z = -VEL_ANGULAR_MAX
793
794             else:
795                 # Sin paredes: seguir recto, no detenerse
.
796                 vel_cmd.linear.x = VEL_LINEAL_BUSQUEDA
797                 vel_cmd.angular.z = 0.0
798                 error_previo = 0.0
799
800             elif estado_actual == "PREPARANDO_GIRO":
801                 deteccion_activada = False
802
803                 # No se coloca velocidad cero aqui para evitar
pausas falsas.
804
805                 if hila_actual == 1:
806                     direccion_giro = -1
807                     objetivo_yaw_giro = normalizar_angulo(
yaw_inicio_fila - ANGULO_OBJETIVO)
808                     pub_pan.publish(Float64(CAM_ANG_IZQ))
809                     print("Camara girada a la izquierda para
Carril 2")
810                     estado_actual = "GIRANDO_U"
811
812                 elif hila_actual == 2:
813                     direccion_giro = 1
814                     objetivo_yaw_giro = normalizar_angulo(
yaw_inicio_fila + ANGULO_OBJETIVO)
815                     pub_pan.publish(Float64(CAM_ANG_DER))
816                     print("Camara girada a la derecha para Carril
3")
817                     estado_actual = "GIRANDO_U"
818

```

```

819         else:
820             print("Estado de giro no valido. Volviendo a
navegacion.")
821             estado_actual = "NAVEGANDO_FILA"
822
823         elif estado_actual == "GIRANDO_U":
824             deteccion_activada = False
825
826             error_yaw = normalizar_angulo(objetivo_yaw_giro -
current_yaw)
827             angulo_restante = abs(error_yaw)
828             angulo_girado = ANGULO_OBJETIVO - angulo_restante
829
830             if esta_escalando:
831                 vel_cmd.linear.x = -0.05
832                 vel_cmd.angular.z = direccion_giro * 0.2
833
834             elif angulo_girado < math.radians(160):
835                 if direccion_giro == -1:
836                     pared_interna = dist_der
837                 else:
838                     pared_interna = dist_izq
839
840                 if pared_interna < 0.25:
841                     vel_cmd.linear.x = -0.05
842                     vel_cmd.angular.z = direccion_giro * (
VEL_GIRO_U + 0.1)
843
844                 elif pared_interna < 0.40:
845                     vel_cmd.linear.x = VEL_PIVOTE
846                     vel_cmd.angular.z = direccion_giro * (
VEL_GIRO_U * 0.8)
847
848                 else:
849                     vel_cmd.linear.x = VEL_PIVOTE
850                     vel_cmd.angular.z = direccion_giro *
VEL_GIRO_U
851
852             elif angulo_restante > tolerancia_yaw:
853                 vel_cmd.linear.x = VEL_PIVOTE
854                 vel_cmd.angular.z = direccion_giro * max
(0.15, min(VEL_GIRO_U, angulo_restante * 1.5))
855
856             else:
857                 # Giro terminado.
858                 hila_actual += 1
859
860                 yaw_inicio_fila = current_yaw
861                 t_entrada_fila = time.time()

```

```

862         estado_actual = "ENTRANDO_FILA"
863
864         vel_cmd.linear.x = VEL_LINEAL_BUSQUEDA
865         vel_cmd.angular.z = 0.0
866
867         print(f"Giro completado. Entrando a la hilera
868 {hila_actual}")
869
870     elif estado_actual == "ENTRANDO_FILA":
871         deteccion_activada = True
872
873         # Avanzar recto durante unos segundos para entrar
874         a la nueva hilera.
875         vel_cmd.linear.x = VEL_LINEAL_BUSQUEDA
876         vel_cmd.angular.z = 0.0
877
878         if (t_now - t_entrada_fila) >= 2.0:
879             estado_actual = "NAVEGANDO_FILA"
880
881         # Reiniciar referencia de yaw y tiempo de
882         fila despues de entrar.
883         yaw_inicio_fila = current_yaw
884         t_inicio_fila = time.time()
885
886         error_previo = 0.0
887         t_apertura = 0.0
888         t_yaw_giro = 0.0
889
890         print("Entrada a nueva hilera completada.
891 Navegando.")
892
893         # Publicar velocidad
894         pub_vel.publish(vel_cmd)
895
896         # Trayectoria
897         trayectoria_puntos.append((pos_x_calc, pos_y_calc))
898
899         # =====
900         # ACTUALIZAR GUI
901         # =====
902         estado_txt.set(estado_actual)
903         total_txt.set(str(contador_plantas))
904         sanas_txt.set(str(sanas))
905         plagas_txt.set(str(plagas))
906         estres_txt.set(str(estres))
907
908         if estado_actual in ["NAVEGANDO_FILA", "ENTRANDO_FILA
909 "]:

```

```

906         lbl_estado.config(fg="green")
907     elif "GIRO" in estado_actual:
908         lbl_estado.config(fg="orange")
909
910     if esta_escalando:
911         lbl_estado.config(fg="red")
912         estado_txt.set("ALERTA ESCALADA")
913
914     if mision_finalizada and not PARADA_POR_FIN_MISION:
915         estado_txt.set(f"{estado_actual} (MISION
FINALIZADA, SIN PARADA)")
916
917     pos_txt.set(f"Pos Robot: X={pos_x_calc:.1f}m, Y={
pos_y_calc:.1f}m")
918
919     root.update_idletasks()
920     root.update()
921
922     rate.sleep()
923
924 except KeyboardInterrupt:
925     print("\nEjecucion interrumpida por el usuario (Ctrl+C).")
926
927 finally:
928     apagar_sistema_seguro()
929     print(f"Resumen final: Total plantas: {contador_plantas}
(Sanas: {sanas}, Plaga: {plagas}, Estres: {estres})")

```

Anexo B: Código de mapeo manual teleoperado

Script auxiliar para la validación del sensor SLAM mediante control por joystick, limitando la velocidad lineal a un máximo de 0,05 m/s y la velocidad angular a 0,3 rad/s, con el fin de evitar el deslizamiento de orugas y lograr una reconstrucción óptima del mapa.

```
1  #!/usr/bin/env python3
2  import rospy
3  import math
4  import cv2
5  import numpy as np
6  import os
7  import time
8  from sensor_msgs.msg import Joy, Imu, LaserScan
9  from nav_msgs.msg import Odometry
10 from geometry_msgs.msg import Twist
11
12 rospy.init_node('mapeo_manual_lento', anonymous=True)
13 pub_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
14
15 # Matar el control oficial de Hiwonder para tener control
16   total y lento
17 os.system("rosnode kill /hiwonder_teleop 2>/dev/null")
18 os.system("rosnode kill /teleop_twist_joy 2>/dev/null")
19
20 linear_speed = 0.0
21 angular_speed = 0.0
22 stop_requested = False
23
24 # Variables para monitorear odometria real
25 real_x = 0.0
26 real_y = 0.0
27
28 def odom_callback(msg):
29     global real_x, real_y
30     real_x = msg.pose.pose.position.x
31     real_y = msg.pose.pose.position.y
32
33 def joy_callback(msg):
34     global linear_speed, angular_speed, stop_requested
35     ly = msg.axes[1]
36     rx = msg.axes[3]
37
38     if abs(ly) < 0.1: ly = 0.0
39     if abs(rx) < 0.1: rx = 0.0
40
41     # Velocidad SUPER LENTA para que el SLAM dibuje perfecto
42     linear_speed = ly * 0.05 # Mx 5 cm/s
```

```

42     angular_speed = rx * 0.3 # Mx 0.3 rad/s
43
44     for i in range(len(msg.buttons)):
45         if msg.buttons[i] == 1:
46             stop_requested = True
47             print(f"\n Botn {i} presionado. Deteniendo misin
...")
48             break
49
50 def guardar_recursos():
51     print("\n DETENIENDO MISIN Y GUARDANDO DATOS...")
52     print(" Guardando mapa SLAM...")
53     os.system("timeout 15 rosrn map_server map_saver -f /
home/hiwonder/Desktop/mapa_perfecto map:=/robot_1/map")
54     print(" Mapa guardado en mapa_perfecto.pgm")
55     os.system("pkill -f joy_node")
56     os.system("pkill -f slam")
57     os.system("pkill -f ydlidar")
58
59 rospy.Subscriber('/joy', Joy, joy_callback)
60 rospy.Subscriber('/robot_1/odom', Odometry, odom_callback)
61
62 vel_cmd = Twist()
63 rate = rospy.Rate(10) # Bucle a 10 Hz para no saturar la
terminal
64 print(" Control manual listo. Velocidad limitada al 5% (Sper
Lento).")
65 print(" Presiona CUALQUIER BOTN para terminar y guardar el
mapa.")
66
67 try:
68     while not rospy.is_shutdown() and not stop_requested:
69         vel_cmd.linear.x = linear_speed
70         vel_cmd.angular.z = angular_speed
71         pub_vel.publish(vel_cmd)
72
73         # Imprimir la odometra real en una sola linea
74         print(f"\r Odometra Real -> X: {real_x:.2f}m | Y: {
real_y:.2f}m", end="", flush=True)
75
76         rate.sleep()
77
78 except KeyboardInterrupt:
79     pass
80 finally:
81     vel_cmd.linear.x = 0.0
82     vel_cmd.angular.z = 0.0
83     pub_vel.publish(vel_cmd)
84     guardar_recursos()

```

Anexo C: Guía de despliegue y operación en campo

El presente instructivo detalla el protocolo secuencial para la inicialización y puesta en marcha del sistema robótico en modalidad autónoma desde la pantalla táctil integrada, prescindiendo de terminales de cómputo externas o enlaces inalámbricos.

Paso 1: Inicialización del sistema operativo y entorno base Con la plataforma energizada y el sistema operativo cargado en la NVIDIA Jetson Nano, la secuencia de arranque se ejecuta mediante el lanzador automatizado *Iniciar SLAM*, dispuesto en el entorno de escritorio mostrado en la Fig. 29. Este acceso directo ejecuta el *script* de Bash que levanta el nodo maestro de ROS (*roscore*) y los controladores de los sensores exteroceptivos e inerciales.

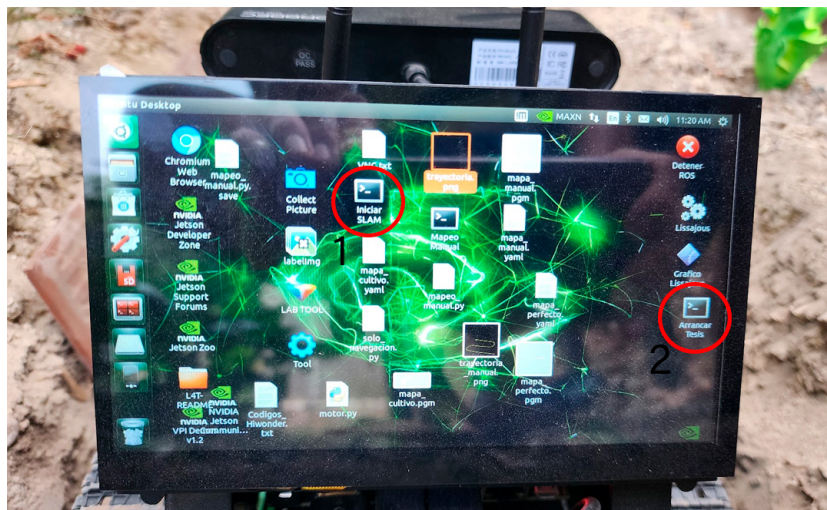


Fig. 29. Entorno de escritorio de la Jetson Nano con los lanzadores de auto-arranque.

Paso 2: Calibración y verificación sensorial Tras la ejecución del primer servicio, la consola de terminal confirma la calibración del sensor inercial (IMU) y el barrido activo del LiDAR 2D, tal como se observa en la Fig. 30. El operador corrobora la ausencia de errores de enlace o excepciones de lectura antes de proceder con la activación del lazo de navegación y percepción.

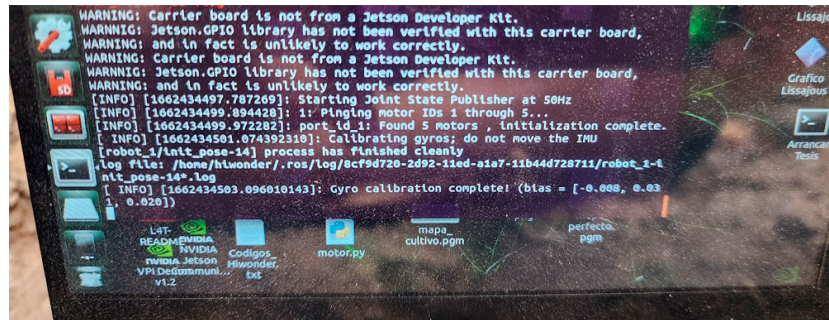


Fig. 30. Terminal del sistema confirmando la calibración de la IMU y el inicio del escaneo LiDAR.

Paso 3: Despliegue de la arquitectura de control y visión artificial

Posteriormente, se acciona el lanzador *Arrancar Tesis*, el cual inicializa el nodo principal de navegación, carga el motor optimizado TensorRT (*best.engine*) en la memoria de la GPU y despliega la interfaz gráfica embebida en la pantalla táctil (Fig. 31). Una vez confirmada la sincronización de tópicos, el vehículo inicia de forma autónoma el recorrido interhilera, transitando al estado operativo *NAVEGANDO_FILA*.

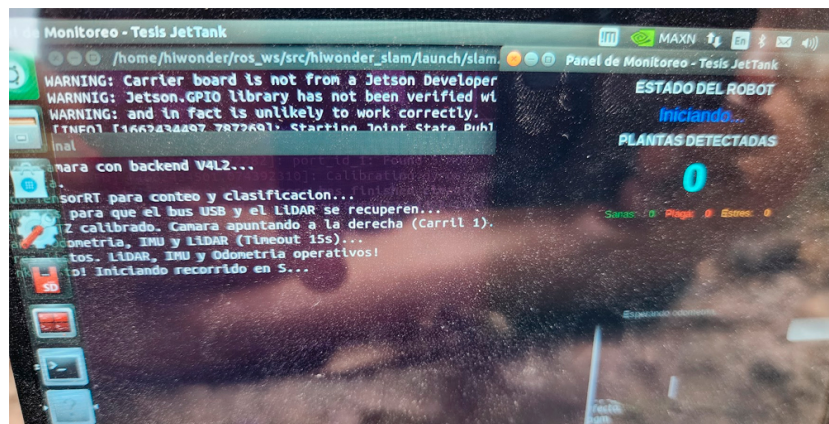


Fig. 31. Carga del motor de inferencia TensorRT y despliegue de la GUI táctil previo al inicio del tránsito.

Paso 4: Culminación de la misión y reporte fitosanitario

Al completar el tránsito por las tres hileras del cultivo, el controlador reactivo comanda la detención total de los motores de tracción. En la pantalla táctil se congela el reporte fitosanitario consolidado con la cuantificación discriminada por clase vegetal (Fig. 32). El sistema mantiene la telemetría visible durante un

intervalo de 30 s para el registro de datos en campo antes de proceder al cierre ordenado de los procesos y la liberación de la memoria física.

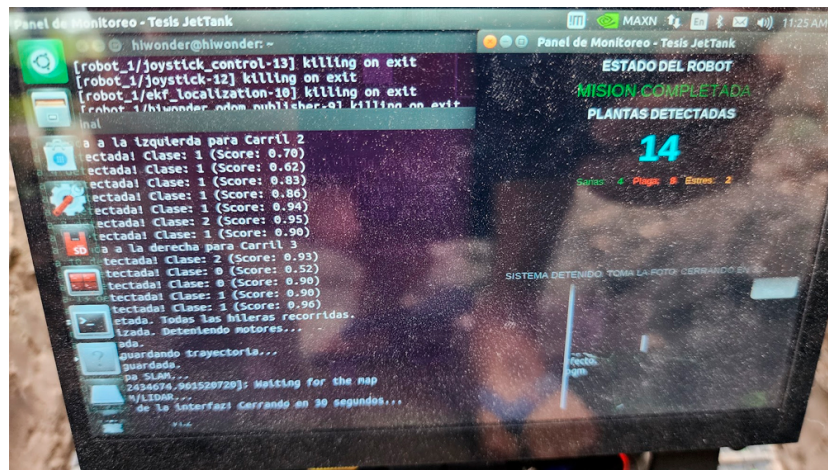


Fig. 32. Finalización de la misión en campo y presentación del balance fitosanitario en la interfaz.

Anexo D: Repositorio de código fuente

Para garantizar la transparencia, reproducibilidad y verificación del sistema implementado, el código fuente completo del proyecto, junto con los scripts auxiliares, modelos de IA y diseños 3D, se encuentran disponibles en un repositorio público de GitHub.



Fig. 33. Código QR - Repositorio de GitHub.

Anexo E: Dataset de entrenamiento (Roboflow)

El banco de imágenes original, junto con las anotaciones manuales y el flujo del preprocesamiento utilizado para el entrenamiento de la red neuronal, está disponible públicamente en Roboflow Universe para su revisión y descarga.



Fig. 34. Código QR - Dataset de imágenes en Roboflow.

Anexo F: Videos demostrativos (Google Drive)

Se ha dispuesto una carpeta de acceso público en Google Drive que contiene los videos demostrativos del robot realizando el recorrido autónomo en el campo de cultivos, evidenciando la navegación reactiva y el conteo fitosanitario en tiempo real.



Fig. 35. Código QR - Carpeta de videos demostrativos en Google Drive.

Anexo G: Reporte de coincidencias

A continuación se adjunta el reporte oficial de análisis de similitud generado por la herramienta Compilatio.



KleynerFabricioFloresCedeño_Tesis
ID : 0b30b48f2aef6a0927303336a11a93b1b09273b3



<1%
Textos sospechosos

Nombre del fichero : KleynerFabricioFloresCedeño_Tesis.txt Tamaño del archivo original : 10,91 MB Número de palabras : 25.562	Depositante : Junior Rafael Figueroa Olmedo Fecha de depósito : 7 de agosto de 2026 Tipo de carga : interface fecha de fin de análisis : 7 de agosto de 2026
---	---

Resumen (sección 1/2)

Localización de los textos sospechosos en el documento :



Incluido en el porcentaje de textos sospechosos :

Similitudes

Pasajes con similitudes a fuentes encontradas en diferentes colecciones.

<1%

Detección de IA

Textos estilísticamente próximos a un texto generado por una IA.
Este índice es un indicador y no una prueba. Comprueba con el autor si domina los conocimientos mencionados en el documento.

15%

Idiomas no reconocidos

Pasajes en los que parte del vocabulario utilizado no forma parte del diccionario de la lengua.
Puede tratarse de un intento del autor de modificar el texto para evitar ser detectado.

20%

No incluido en el porcentaje de textos sospechosos :

Textos entre comillas

Pasajes entre comillas, a menudo indicativos de una cita.

3%