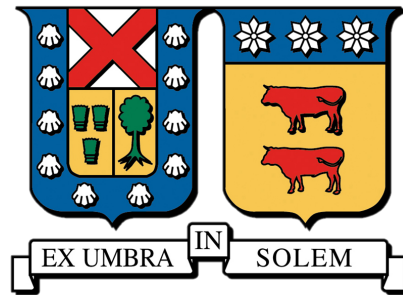


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**“Reconocimiento visual de residuos en entornos urbanos
usando un robot de patrullaje e inteligencia artificial”**

Juan Gamboa Araos

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL
ELECTRÓNICO**

PROFESOR GUÍA : Fernando Auat Cheein
PROFESOR CORREFERENTE : Samir Kouro

Febrero 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Reconocimiento visual de residuos en entornos urbanos usando un robot de patrullaje e inteligencia artificial

Nombre del candidato(a): Juan Anibal Gamboa Araos

Carrera / Grado: Ingeniería Civil Electrónica con mención en Computadores

Campus: Casa Central Valparaíso ; **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Fernando Auat Cheein, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 01/10/2025 ; Firma: 

Estudiante o Candidato(a):

Fecha: 01/10/2025 ; Firma: 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

En primera instancia, deseo expresar mi más profundo agradecimiento a mi familia: Juan Gamboa Vera, Solange Araos Manquilef, Benjamín Gamboa y Diego Gamboa, quienes me han brindado su apoyo incondicional a lo largo de toda mi trayectoria universitaria. Su amor, paciencia y aliento constante han sido pilares fundamentales para alcanzar este logro, y sin su respaldo, este camino habría sido significativamente más difícil.

También quiero extender mi gratitud a mi profesor guía, Fernando Auat, y a su equipo, por su orientación, conocimientos y dedicación a lo largo del desarrollo de este proyecto. Su guía ha sido invaluable para superar los desafíos y consolidar mis aprendizajes en esta etapa.

A mis amigos, quienes me han acompañado en este proceso, brindándome su apoyo, motivación y momentos de distracción que hicieron más llevadero este desafío. Su compañía ha sido esencial para mantener el equilibrio entre el esfuerzo académico y la vida personal.

Finalmente, agradezco a todas aquellas personas que, de una u otra manera, contribuyeron a mi formación y al desarrollo de este proyecto. Cada enseñanza, consejo y gesto de apoyo han sido fundamentales en mi crecimiento personal y profesional.

Resumen

El crecimiento demográfico que han experimentado las ciudades ha generado problemas directos con la recolección de residuos, donde los métodos tradicionales resultan cada vez más ineficientes lo que provoca una acumulación de basura en las calles. Para obtener una mejor gestión en la recolección de residuos, se propone este proyecto donde, usando un robot de patrullaje como lo es el Go1 de Unitree, se logre detectar, clasificar y obtener la geolocalización de los residuos para así crear un informe con los puntos donde se deben recolectar los residuos.

Abstract

The population growth that cities have experienced has led to direct problems with waste collection, where traditional methods are becoming increasingly inefficient, resulting in garbage accumulation on the streets. To achieve better waste collection management, this project proposes using a patrol robot like the Go1 from Unitree, which can detect, classify, and obtain the geolocation of waste to create a report indicating the points where waste should be collected.

Índice de Contenidos

1. Introducción	1
1.1. Problema	2
1.2. Objetivos	2
1.2.1. Objetivo general	2
1.2.2. Objetivos específicos	2
2. Marco teórico	4
2.1. Gestión de Residuos Urbanos y su Problemática	4
2.1.1. Limitaciones de los métodos tradicionales de recolección . . .	5
2.1.2. Integración de tecnología en la gestión de residuos	5
2.1.3. Sistemas de detección automática y geolocalización	6
2.1.4. Ejemplos de proyectos de Smart City existentes	7
2.1.5. Adaptación de la Visión por Computadora y Robótica a la ges- tión de residuos	7
2.2. Visión por computadora y detección de objetos	8
2.2.1. Fundamentos de la visión por computadora	8
2.2.2. Detección de objetos con YOLO (You Only Look Once) . . .	9
2.3. Captura de Video en Tiempo Real a través de Iriun Webcam	16
2.3.1. Cómo Funciona Iriun Webcam + Arquitectura	16
2.3.2. Protocolo y Tecnologías de Iriun Webcam	17
2.3.3. Rendimiento y Latencia de Iriun Webcam	17

2.3.4.	Beneficios de Iriun Webcam en Robótica y Visión por Máquina	18
2.3.5.	Configuración e Integración de Iriun Webcam y OpenCV . . .	18
2.4.	Geolocalización con MATLAB Mobile	21
2.4.1.	Conceptos básicos del posicionamiento GPS	21
2.4.2.	Usando MATLAB Mobile como plataforma para capturar da- tos GPS	22
2.4.3.	Arquitectura de comunicación MATLAB Mobile para Python	23
2.5.	Robótica Cuadrúpeda: Uso del Robot Go1 de Unitree	26
2.5.1.	Introducción a la Robótica Cuadrúpeda	26
2.5.2.	Arquitectura del Go1 de Unitree	26
2.5.3.	Ventajas del Go1 frente a Sistemas Tradicionales	29
2.5.4.	Limitaciones del Go1 en comparación con los sistemas con- vencionales	30
3.	Diseño de la arquitectura del sistema	32
3.1.	Requisitos del sistema	32
3.1.1.	Requisitos Funcionales (RF)	32
3.1.2.	Requisitos No Funcionales (RNF)	33
3.2.	Arquitectura del sistema	34
3.2.1.	Robot Go1 de Unitree	35
3.2.2.	Dispositivos Externos	36
3.2.3.	Notebook del operador (Control y Procesamiento)	36
4.	Desarrollo del sistema	38
4.1.	Entrenamiento IA usando YOLOv11	38
4.1.1.	Dataset	38
4.1.2.	Código para el entrenamiento	40
4.2.	Código para Captura de Imágenes en Tiempo Real	43
4.2.1.	Explicación del Código	44
4.3.	Procesamiento de Imágenes con YOLOv11	45

4.3.1.	Carga del Modelo YOLOv11m	45
4.3.2.	Procesamiento de Imágenes en Tiempo Real	45
4.3.3.	Explicación del Código	47
4.4.	Integración con MATLAB Mobile para Obtención de Coordenadas GPS	49
4.4.1.	Configuración del Servidor en MATLAB	49
4.4.2.	Solicitud de Coordenadas GPS desde Python	50
4.5.	Almacenamiento de Datos en Excel	54
4.5.1.	Objetivo del Módulo de Almacenamiento	54
4.5.2.	Eliminación Automática de Registros Duplicados	57
4.6.	Módulo de Visualización	57
4.7.	Implementación de todos los módulos	63
4.7.1.	Montaje	63
4.7.2.	Iniciar aplicaciones	65
4.7.3.	Iniciar códigos	65
4.7.4.	Manejo del robot	72
4.7.5.	Patrullaje	74
5.	Resultados	76
5.1.	Entrenamiento IA	76
5.1.1.	Evaluación de la Pérdida Durante el Entrenamiento	76
5.1.2.	Matriz de Confusión Normalizada	77
5.1.3.	Curvas de Precisión-Recall (PR)	79
5.1.4.	Curvas F1-Score	80
5.1.5.	Modelo Entrenado	82
5.2.	Prueba del sistema	82
5.3.	Evaluación de Requisitos Funcionales	85
5.4.	Evaluación de Requisitos No Funcionales	87
6.	Conclusiones	89

Índice de Tablas

2.1. Protocolos y tecnologías utilizadas en Iriun Webcam	17
2.2. Comparación de métodos de captura de video en robótica.	18
2.3. Métodos de Posicionamiento GPS y su precisión	22
2.4. Especificaciones Técnicas del Go1 de Unitree	27
3.1. Requisitos Funcionales del Sistema	33
3.2. Requisitos No Funcionales del Sistema	34

Índice de Figuras

2.1. Versiones de YOLO	10
2.2. Arquitectura YOLOv11	11
2.3. Diagrama de comunicación del sistema	24
2.4. Arquitectura Go1	28
2.5. Partes Robot	28
3.1. Diagrama de Alto Nivel	35
4.1. Soporte NEEWER SP-10	63
4.2. Pieza impresa 3d	64
4.3. Montaje Final	64
4.4. Posición Go1 en reposo	73
4.5. Batería inteligente	73
4.6. Mando Go1	74
4.7. Interfaz gráfica	75
5.1. Resultados del entrenamiento	77
5.2. Resultados del entrenamiento	77
5.3. Matriz de Confusión Normalizada	78
5.4. Box Precision-Recall	79
5.5. Mask Precision-Recall	80
5.6. Box F1 Precision-Recall	81
5.7. Mask F1 Precision-Recall	81

5.8. Reconocimiento Cardboard	82
5.9. Reconocimiento Papel	83
5.10. Reconocimiento Metal	83
5.11. Reconocimiento Plastic	84
5.12. Datos guardados en Excel	85

Capítulo 1

Introducción

En los últimos años, el crecimiento demográfico de las ciudades ha aumentado enormemente [1]. Este crecimiento acelerado ha creado una gran cantidad de desafíos en la gestión de los servicios humanos, siendo la recolección de residuos uno de los más críticos [2]. A medida que las ciudades se expanden, los métodos tradicionales de recolección de basura se vuelven cada vez más ineficientes, lo que conlleva una acumulación de desechos en las calles, lo que tiene una repercusión en la salud pública como en el medio ambiente [3] [4].

Este documento presenta un sistema semiautónomo de patrullaje y detección de residuos, basado en un robot cuadrúpedo Go1 de Unitree, el cual es controlado a distancia mediante un mando inalámbrico. El sistema incorpora dos dispositivos móviles, cada uno con una función específica:

1. **Celular 1:** Captura video en tiempo real usando Iriun Webcam, los frames del video son procesados usando un modelo de visión artificial basado en YOLOv11 que detecta y clasifica residuos en categorías reciclables.
2. **Celular 2:** Obtiene coordenadas GPS en tiempo real usando MATLAB Mobile, permitiendo geolocalizar los residuos detectados.

Una vez se detecta un residuo, se obtiene su geolocalización y se almacena en una base de datos. Esto permite optimizar la planificación y gestión de la recolección de residuos.

1.1. Problema

La creciente demanda sobre los sistemas de recolección de residuos, derivada del acelerado crecimiento demográfico en las grandes ciudades, ha evidenciado sus limitaciones para gestionar eficazmente los desechos. Como consecuencia, se observa una acumulación progresiva de residuos en las vías públicas. En este contexto, resulta imperativo el desarrollo e implementación de tecnologías que optimicen la gestión y recolección de residuos, permitiendo una planificación más eficiente y sostenible.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un sistema semiautónomo basado en un robot de patrullaje capaz de detectar residuos, clasificarlos, registrar su ubicación geográfica y almacenar la información en una base de datos para su posterior análisis y gestión.

1.2.2. Objetivos específicos

1. Se propone la implementación de un sistema de captura de video en tiempo real mediante Iriun Webcam, permitiendo la transmisión de imágenes desde un dispositivo móvil montado en el robot Go1 hacia una computadora para su procesamiento y análisis.
2. Entrenar un modelo de inteligencia artificial capaz de reconocer cartón-papel, metal y plástico.

3. Se propone la implementación de un sistema de geolocalización para el robot Go1 mediante un segundo dispositivo móvil montado en su estructura. Este dispositivo utilizará la aplicación MATLAB Mobile, que permitirá obtener la longitud y latitud del robot cuando sea requerido.
4. Desarrollo de un entrono visual para mostrar en tiempo real las detecciones superponiendo los resultados del modelo sobre la imagen capturada por la cámara.
5. Almacenar automáticamente los datos de detección en un archivo Excel
6. Estudiar y practicar el uso del mando del robot para garantizar su correcta operación y control.
7. Evaluar el rendimiento del sistema, midiendo la precisión del modelo de IA, la confiabilidad del GPS y la autonomía operativa del robot Go1 de Unitree.

Capítulo 2

Marco teórico

2.1. Gestión de Residuos Urbanos y su Problemática

La gestión adecuada de los residuos sólidos urbanos es vital para la salud pública y para la sostenibilidad. La generación de residuos se ha convertido en una cifra impresionante en Chile en los últimos años. En 2019, se produjeron alrededor de 7,86 millones de toneladas de residuos municipales (Ministerio del Medio Ambiente), lo que corresponde a un promedio de 1,13 kg de residuos producidos por persona por día [7]. Este número es representativo del crecimiento de la generación de residuos impulsado por el crecimiento poblacional y el desarrollo económico.

Un manejo deficiente de estos residuos puede crear una serie de problemas:

- **Contaminación:** Esto lleva a la contaminación del suelo y del agua, aumentando las amenazas a la biodiversidad y los recursos naturales.
- **Peligros para la salud:** Los residuos dispuestos de manera inadecuada pueden actuar como un caldo de cultivo para infecciones y enfermedades, representando un riesgo para la salud de la población.
- **Impacto económico:** El costo de limpiar las áreas contaminadas y tratar las enfermedades asociadas puede acumular gastos para los municipios y el sistema de

salud.

2.1.1. Limitaciones de los métodos tradicionales de recolección

Históricamente, la recolección de residuos en las ciudades depende de rutas y horarios predefinidos, independientemente de los residuos producidos en cualquier punto de contacto. Hay varias limitaciones para este enfoque:

- **Ineficiencia operativa:** Los camiones de recolección pueden pasar por regiones con contenedores vacíos, mientras que en otras áreas los residuos pueden acumularse sin recolección oportuna.
- **Altos costos:** Las rutas de recolección no optimizadas llevan a un mayor consumo de combustible y un mayor desgaste de los vehículos, generando mayores costos operativos.
- **Bajo índice de reciclaje:** La clasificación de residuos en contenedores dificulta la separación y recuperación de materiales reciclables y obstaculiza la valorización de los residuos.

En Chile, la cantidad promedio de residuos sólidos domiciliarios y similares generados por una persona al día es de 1,22 kg [8]. Este volumen no siempre coincide con la infraestructura y los métodos de recolección actuales diseñados para recopilar esta cantidad de residuos, especialmente en las áreas urbanas más densamente pobladas.

2.1.2. Integración de tecnología en la gestión de residuos

Por no mencionar que, donde hay tecnología, definitivamente se utilizaría para el reciclaje y la recolección inteligente. La integración de tecnología en la gestión de residuos ha sido un enfoque clave para abordar los inconvenientes de los métodos convencionales. Algunas de las innovaciones más notables son:

- **El uso de Inteligencia Artificial (IA) y Visión por Computadora:** Estas herramientas permiten la identificación y clasificación automática de los residuos, mejorando su tratamiento en plantas. Por ejemplo, los sistemas de visión por computadora pueden identificar diferentes tipos de materiales, lo que ayuda a separarlos y reciclarlos .

Los algoritmos de IA pueden analizar datos en tiempo real, como los niveles de llenado de contenedores y las condiciones del tráfico, y crear rutas de recolección óptimas para los camiones. Así se ahorra en combustible y emisiones de gases contaminantes [9]

- **Sensores IoT (Internet de las Cosas):** La instalación de sensores en los contenedores permite el monitoreo de capacidad en tiempo real, lo cual alerta cuando necesitan ser vaciados. También permite evitar desbordamientos y optimizar la frecuencia de recolección.[10]

2.1.3. Sistemas de detección automática y geolocalización

Las ciudades que buscan mejorar sus sistemas de gestión ambiental han desarrollado sistemas automáticos que combinan la detección de residuos con la geolocalización. Estos sistemas tienen varias ventajas:

- **Monitoreo en tiempo real:** Facilitan la identificación más rápida de regiones con aumento de formación de residuos, promoviendo acciones correctivas prontas.
- **Distribución óptima de recursos:** Estimaciones precisas de las coordenadas y la cantidad de residuos se utilizan para la planificación óptima de rutas de recolección de residuos y distribución de recursos.
- **Incremento de tasas de reciclaje:** Clasificar y separar adecuadamente los residuos en origen significa que se pueden reciclar una mayor cantidad de materiales, minimizando la cantidad de residuos que van a vertederos.

2.1.4. Ejemplos de proyectos de Smart City existentes

Muchas ciudades en todo el mundo están utilizando tecnologías avanzadas para incrementar la eficiencia de la disposición de residuos:

- **Estados Unidos (San Francisco):** SF posee sistemas automáticos de clasificación en sus plantas de reciclaje (con IA) para crear eficiencia y precisión en la separación de materiales reciclables.[11]
- **Alcorcón (España):** La Policía Municipal ha introducido drones para detectar depósitos ilegales de escombros en áreas industriales, mejorando el control y la vigilancia ambiental.[12]
- **Barcelona, España:** La ciudad ha desplegado sensores en contenedores, capturando niveles de llenado y permitiendo ajustar las rutas de recolección basadas en la demanda real. Esta iniciativa ha reducido efectivamente los costos operativos y las emisiones asociadas con el transporte de residuos.[13]

2.1.5. Adaptación de la Visión por Computadora y Robótica a la gestión de residuos

La visión por computadora y la robótica combinadas con la gestión de residuos tienen sus propias ventajas:

- **Eficiencia y precisión:** Los sistemas automatizados pueden trabajar las 24 horas del día y clasificar residuos con una precisión de hasta el 94,74 %. [14]
- **Minimización de riesgos laborales:** La automatización de tareas peligrosas o monótonas ayuda a reducir la exposición de los empleados a entornos desafiantes.[15]
- **Eficiencia de recursos disponibles:** La automatización hace que el procesamiento de volúmenes de residuos sea más eficiente en tiempo, así como reduciendo el costo de las operaciones, ayudando a que las plantas de tratamiento sean más rentables.[16]

En general, esto indica que la adopción de tecnologías avanzadas en la gestión de residuos urbanos es una alternativa significativa ante las limitaciones de los enfoques convencionales, para un proceso más sostenible y eficiente, estrechamente relacionado con los objetivos de desarrollo sostenible y protección ambiental.

2.2. Visión por computadora y detección de objetos

2.2.1. Fundamentos de la visión por computadora

La visión por computadora es una parte de la inteligencia artificial que permite a las computadoras interpretar y procesar datos visuales del mundo que las rodea, imitando la capacidad humana de ver. Implica el uso de algoritmos y modelos matemáticos que permiten a las computadoras extraer, analizar e interpretar características de imágenes y videos. Sus usos son muy diversos, desde la automatización industrial hasta la seguridad, la medicina y el transporte.[17]

¿Qué es la visión por computadora y cómo utilizarla?

La visión por computadora es la práctica de adquirir, procesar y analizar imágenes digitales para generar información significativa. El proceso general incluye:

- **Adquisición de imágenes:** El proceso de tomar imágenes con la ayuda de cámaras o sensores especializados.
- **Preprocesamiento:** Mejora de la calidad de las imágenes, reducción de ruido, ajuste de contraste.
- **Procesamiento y análisis de datos:** Uso de algoritmos para identificar características específicas como bordes, formas o colores.
- **Toma de decisiones:** Mapeo cardinal de los datos procesados hacia acciones o información.

Aplicaciones en la vida diaria y la industria

- **Industria:** Inspección de calidad y detección de defectos en líneas de producción.
- **Seguridad:** Sistemas de cámaras que detectan actividades inusuales o brechas.
- **Atención médica:** Su aplicación en el análisis de imágenes médicas para diagnóstico y monitoreo de enfermedades.
- **Automotriz:** Autos autónomos que ven señales de tráfico y obstáculos en tiempo real. Entrenados en detección y clasificación de objetos en entornos no controlados.
- **Reconocimiento de objetos:** Reconocer y clasificar objetos dentro de escenas complejas.
- **Seguimiento de movimiento:** Seguimiento de la trayectoria de objetos o humanos en movimiento.[18]

2.2.2. Detección de objetos con YOLO (You Only Look Once)

Uno de los algoritmos de detección de objetos más avanzados es YOLO debido a su capacidad para procesar en tiempo real manteniendo un buen grado de precisión. En comparación, otros modelos de detección de objetos más tradicionales, como Faster R-CNN (e incluso SSD: Single Shot Detector), dividen la imagen en varias regiones diferentes y ejecutan la inferencia en todas ellas, lo que los hace significativamente más lentos pero también altamente precisos para liberar imágenes.[19]

Arquitectura de YOLOv11

YOLO se basa en una red neuronal convolucional (CNN) profunda para lograr detección de objetos en tiempo real y con precisión. Su arquitectura consiste en múltiples capas:[20]

1. **Backbone (Espina dorsal):** Extracción de características de la imagen.
2. **Neck:** Una pirámide de características de múltiples etapas que mejora y fusiona características para una mejor detección de objetos de varios tamaños.
3. **Head:** Produce las predicciones finales de detección.

Todos estos componentes son los pilares clave del procesamiento de imágenes y también de la detección de objetos bien organizada.

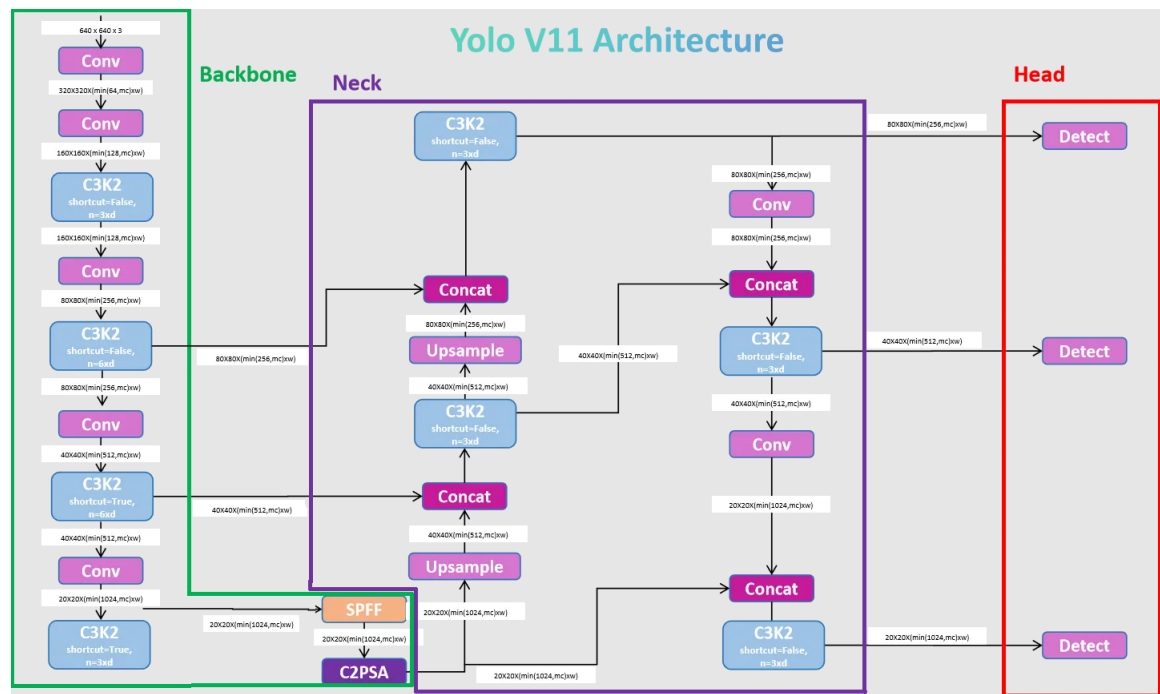


Figura 2.2: Arquitectura YOLOv11

Backbone

Ayuda a extraer las características relevantes de la imagen. Esto implica de nuevo la aplicación de una secuencia de capas convolucionales y mecanismos de atención para mejorar la detección de objetos en diferentes escalas.

1. Procesamiento Inicial

El tamaño de la imagen de entrada es 640x640x3 (RGB). Esto indica que se adoptan capas convolucionales con diferentes tamaños de kernel para adquirir no solo los detalles de grano fino, sino también la información estructural y gruesa. A medida que avanzamos, la resolución disminuye, pero el número de canales aumenta para capturar características más abstractas.

2. Uso de Bloques C3K2

C3K2 (shortcut=False, n=3cd, 5cd): Bloques convolucionales que fortalecen la extracción de características y minimizan la duplicación de datos. La arquitectura combina capas convolucionales y residuales, lo que permite un procesamiento más rápido.

3. Mecanismos de Atención para Mejor Detección

- a) **SPPF (Pooling de Pirámide Espacial - Rápido):** Esta capa captura la información contextual de múltiples escalas y permite detectar los objetos de varios tamaños.
- b) **C2PSA (Bloque convolucional con Atención Espacial Paralela):** Dos de las 3 capas convolucionales implementan un mecanismo de atención espacial, enfocándose en las regiones más relevantes de la imagen de entrada.

Finalmente, el resultado del Backbone es un mapa de características, que tiene información de alto nivel sobre qué objetos están presentes en la imagen.

Neck

El Neck refina la representación de las características extraídas por el backbone para facilitar una mejor fusión de detección multiescalar, permitiendo una mayor detección en diferentes escalas.

1. Reducción de Dimensionalidad y Combinación de Características

- a) Uso de Upsampling:* La resolución de algunas capas se aumenta para detectar mejor objetos pequeños.
- b) Concatenación de características (Concat):* Combina las características de diferentes capas para preservar tanto la información de bajo nivel como de alto nivel.

2. Módulos C3K2 Capas Convolucionales

C3K2 (shortcut=False, n=3cd, 5cd): Bloques que ayudan a combinar características relevantes sin descartar ningún detalle. Para afinar características para la detección final, se utilizan capas convolucionales adicionales.

Lo que se obtiene en las salidas del neck es una representación multiescalar más eficiente, lo que permite una localización más precisa de objetos de diferentes tamaños.

Head

La Cabeza es el último componente en la arquitectura YOLOv11, y convierte características del Backbone, refinadas por el Neck, en predicciones concretas para la detección de objetos. Esta parte genera cuadros delimitadores, confianza en la detección y clasificación de objetos.

Predicción de Detección La cabeza contiene las capas convolucionales y las operaciones finales de detección. La versión del modelo en YOLOv11 está diseñada para detección a múltiples escalas, lo que permite realizar predicciones a múltiples escalas y ayuda al modelo a generalizar mejor para objetos pequeños y grandes.

1. Predicciones a través de escalas

La Cabeza produce predicciones en 2 resoluciones diferentes:

- a)* **Salida de 80×80 píxeles:** Para la localización de objetos pequeños en la imagen.
- b)* **Salida de 40×40 píxeles:** Para la detección de objetos medianos y grandes.

¿Por qué resoluciones tan diferentes?

Esperamos enseñar al modelo nuevas técnicas para detectar objetos de varios tamaños. Generalmente, los objetos pequeños se detectarán en una resolución más alta que los objetos grandes, que pueden ser detectados con menos detalle.

2. Predicción del Cuadro Delimitador

Cada celda de la cuadrícula predice múltiples cuadros delimitadores, los cuales contienen la información sobre la posible ubicación del objeto en la imagen. Cada predicción de detección se compone de:

- a)* **(x, y)** Coordenadas de la esquina superior izquierda del cuadro delimitador del objeto.
- b)* **(w, h)** El ancho y la altura del cuadro delimitador en tamaño de imagen normalizado

3. Corrección de Predicción

Las coordenadas para los cuadros delimitadores se refinan basándose en un ajuste sigmooidal para abordar desafíos como la superposición entre cuadros delimitadores y también ayudan a dirigirse correctamente a los objetos.

Cálculo de la Confianza para la Detección El modelo también predice un valor de confianza para cada cuadro delimitador, que expresa cuán seguro está el modelo de que el cuadro rodea un objeto real. Este valor se calcula como:

$$Confianza = P(objeto) \times I \cap U \quad (2.1)$$

Donde:

- $P(objeto) \rightarrow$ Es la probabilidad de que el cuadro contenga un objeto.
- $I \cap U \rightarrow$ Es la intersección entre la predicción y el cuadro de verdad del terreno del objeto.

Filtrado de Predicciones con Non-Maximum Suppression (NMS) Después de calcular la confianza, el modelo elimina predicciones superpuestas a través de Non-Maximum Suppression. Este proceso:

- Remueve esos cuadros superpuestos y retiene solo el más confiable.
- Reduce el número de falsos positivos, llevando a una mejor precisión final

Clasificación de Detección de Objetos Finalmente, la Cabeza clasifica cada objeto detectado en la imagen. Para cada objeto, el clasificador produce una probabilidad para cada clase y selecciona la clase con la mayor confianza.

El modelo produce una probabilidad para cada clase y elige la que tiene el valor más alto.

$$Clase = \arg \max_i P(clase_i)$$

Donde:

1. $P(clase_i)$ es la probabilidad de que el objeto esté en la clase i
2. Se elige la clase con el valor de probabilidad más alto.

2.3. Captura de Video en Tiempo Real a través de Iriun Webcam

En sistemas de visión por máquina en robótica, la transferencia de video en tiempo real es una condición vital para el conocimiento preciso del entorno y para tomar decisiones basadas en imágenes. Iriun Webcam: Una aplicación que convierte un dispositivo móvil en una cámara web inalámbrica, ayudando a la integración con herramientas de procesamiento de imágenes (OpenCV) en Python.[21]

2.3.1. Cómo Funciona Iriun Webcam + Arquitectura

Iriun Webcam funciona con una arquitectura cliente-servidor, lo que significa que:

1. **Dispositivo móvil (Cliente):** Es donde se captura y envía el video.
2. **Computadora (Servidor):** Recibe la transmisión de video e interpreta como un dispositivo de captura estándar (DirectShow en Windows, V4L2 en Linux, AVFoundation en macOS).
3. **Flujo de Datos de Iriun Webcam en la red:**
 - a) **Captura de video en el móvil:** Usa la API de cámara nativa para obtener imágenes en vivo.
 - b) **Compresión y Transmisión de Video:** El video es comprimido con formatos de baja latencia como MJPEG o H.264, reduciendo el uso innecesario de ancho de banda.
 - c) **Recepción en PC:** La aplicación del servidor decodifica la señal y la pone a disposición como un dispositivo de video virtual.
 - d) **Integración con OpenCV:** Transmisión capturada desde `cv2.VideoCapture()` en Python, para procesamiento en tiempo real.

2.3.2. Protocolo y Tecnologías de Iriun Webcam

Iriun Webcam usa una combinación de protocolos y estándares multimedia para habilitar la transmisión eficiente de video:

Tecnología	Función en Iriun Webcam
MJPEG (Motion JPEG)	Codificación de video con compresión intra-frame (cada frame es una imagen independiente).
H.264	Compresión de video más eficiente que reduce el uso de ancho de banda, optimizando la transmisión.
UDP/TCP	Transmisión de datos de video con opciones para baja latencia o corrección de errores.
DirectShow (Windows)	Interfaz de captura de video compatible con OpenCV.
V4L2 (Linux)	API para manejar dispositivos de video en sistemas Unix.
AVFoundation (macOS)	Plataforma de multimedia utilizada en dispositivos Apple.

Tabla 2.1: Protocolos y tecnologías utilizadas en Iriun Webcam

Iriun Webcam se puede conectar tanto por USB como de forma inalámbrica (Wi-Fi), convirtiéndola en una opción versátil para sistemas de visión por máquina con diferentes requisitos de transmisión de datos.

2.3.3. Rendimiento y Latencia de Iriun Webcam

La latencia del sistema de captura de video es una de las métricas más importantes en aplicaciones de visión por máquina en robótica. Gracias a la optimización de código al enviar video comprimido, Iriun Webcam logra bajas latencias con anchuras de respuesta de:

- **Conexión USB:** < 20 ms (dependiendo del procesador y puerto USB).
- **Wi-Fi (5 GHz):** ~ 50-80 ms (adecuado para uso en vivo).
- **Wi-Fi (2.4 GHz):** > 100 ms (pobre para alta velocidad).

2.3.4. Beneficios de Iriun Webcam en Robótica y Visión por Máquina

Método	Ventajas	Desventajas
Iriun Webcam (Wi-Fi)	Conexión inalámbrica, alta calidad, fácil integración con OpenCV.	Puede presentar retardo si la red es inestable.
Cámara integrada en el robot	Baja latencia, alto rendimiento en hardware dedicado.	Costo elevado, difícil de reemplazar.
Cámara IP (RTSP)	Calidad estable, fácil acceso remoto.	Latencia alta en algunas configuraciones.
Webcam USB	Baja latencia, conexión directa.	Requiere cableado, menos flexible en entornos móviles.

Tabla 2.2: Comparación de métodos de captura de video en robótica.

2.3.5. Configuración e Integración de Iriun Webcam y OpenCV

Para utilizar Iriun Webcam en Python, es necesario configurar correctamente la conexión con OpenCV y asegurarse de que el software reconozca el índice de la cámara.

Instalación de Iriun Webcam

Iriun Webcam puede desplegarse de esta manera:

1. Obtén la aplicación Iriun Webcam en tu teléfono celular.
2. Descargar e instalar la aplicación en tu computadora
3. Asegúrate de que el teléfono celular y la PC estén conectados a la misma red Wi-Fi.

4. Ejecuta la aplicación en el teléfono celular y verifica que la cámara se ingrese en la PC.

Identificación del Índice de la Cámara en OpenCV

La transmisión de la cámara a un sistema en OpenCV se asigna con un índice para que se pueda acceder a la transmisión continua de video de la cámara.

Listing 2.1: Código para verificar el índice de la cámara en OpenCV

```
# Prueba y error para encontrar el indice correcto
for i in range(5):
    cap = cv2.VideoCapture(i)
    if cap.isOpened():
        print(f"Camara encontrada en el indice {i}")
        cap.release()
```

OpenCV - Captura de Video en Tiempo Real

OpenCV se utiliza junto con el índice detectado para recibir y procesar video desde Iriun Webcam:

Listing 2.2: Captura de Video en Tiempo Real

```
import cv2

# Captura de video desde Iriun Webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        print(" No se pudo capturar el video.")
        break

    # Mostrar el video en tiempo real
    cv2.imshow("Iriun Webcam - Video en Vivo", frame)

    # Salir con la tecla 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

- **cv2.VideoCapture(0)**→ Abre la transmisión de video desde Iriun Webcam.
- **cap.read()**→ Lee cada cuadro del video en tiempo real.

- `cv2.imshow()` → Renderiza el video en una ventana
- `cv2.waitKey(1) & 0xFF == ord('q')` → Permite cerrar la ventana con la tecla q.

2.4. Geolocalización con MATLAB Mobile

La geolocalización es el conjunto de datos que obtiene la ubicación geográfica de un cierto objeto o dispositivo en un sistema de coordenadas mundial. La integración de la geolocalización es un paso que aumenta la efectividad de los procesos en sistemas de visión artificial y robótica, considerando no solo la detección del objeto móvil sino también su posición en el espacio.

La geolocalización es un enfoque esencial de este proyecto ya que utiliza la ubicación GPS y asocia dicha ubicación con cada residuo detectado, haciendo posible la planificación de la recolección y el análisis posterior para determinar los patrones de acumulación de residuos. Con este fin, se emplea MATLAB Mobile, que proporciona acceso a los sensores del teléfono móvil, como el módulo GPS, y envía estos datos a través de la conexión de red (TCP/IP) a la computadora.[24]

2.4.1. Conceptos básicos del posicionamiento GPS

El Sistema de Posicionamiento Global (GPS) es un sistema de navegación basado en satélites que proporciona al público mundial, incluyendo a los que están en la superficie, en el aire y en el mar. Los principios del proceso de posicionamiento GPS se pueden resumir de la siguiente manera:

Métodos de Posicionamiento GPS

Método	Precisión Aproximada	Descripción
GPS Autónomo (Standalone GPS)	5 - 10 m	Utiliza señales satelitales sin correcciones adicionales, afectado por condiciones atmosféricas y calidad del receptor.
A-GPS (Assisted GPS)	2 - 5 m	Complementa el GPS estándar con datos de servidores externos para reducir el tiempo de adquisición de señal y mejorar la precisión.
DGNSS (Differential GPS)	10 cm - 1 m	Aplica correcciones basadas en estaciones terrestres de referencia, compensando errores del GPS autónomo y mejorando la precisión.
RTK (Real-Time Kinematics)	1 - 2 cm	Técnica de alta precisión que usa medición de fase de la señal de los satélites y corrección en tiempo real a través de estaciones de referencia.

Tabla 2.3: Métodos de Posicionamiento GPS y su precisión

Para este proyecto, MATLAB Mobile emplea los GPS de teléfonos inteligentes funcionando en modo GPS Autónomo o A-GPS, con precisión de aproximadamente 3-5 metros.

2.4.2. Usando MATLAB Mobile como plataforma para capturar datos GPS

MATLAB Mobile es una aplicación que proporciona acceso a los sensores del dispositivo móvil y envía datos en tiempo real a MATLAB que se ejecuta en la PC.

Este proyecto obtiene coordenadas GPS en tiempo real a través de MATLAB Mobile y las envía a un servidor en Python.[25]

Características de geolocalización de MATLAB Mobile:

- Utiliza el sensor GPS del teléfono para capturar los datos de latitud, longitud y altitud.
- No impide guardar datos en MATLAB en el PC o enviarlos a un servidor separado.
- Se conecta a Wi-Fi o datos para transmitir información en tiempo real.
- Funciona en los sistemas operativos Android y iOS.

2.4.3. Arquitectura de comunicación MATLAB Mobile para Python

Esto se hace configurando un servidor TCP en MATLAB que transmitirá sin problemas los datos GPS de MATLAB Mobile a Python. Este servidor permite que Python solicite datos en tiempo real, manteniéndolo sincronizado con el sistema de detección de residuos.

La comunicación que ocurre entre MATLAB Mobile y Python:

1. Las coordenadas GPS son obtenidas del teléfono por MATLAB Mobile.
2. EL computador ejecuta un servidor TCP con MATLAB que espera solicitudes de los datos.
3. El código Main de Python envía una solicitud de GPS.
4. MATLAB entrega la latitud y longitud.

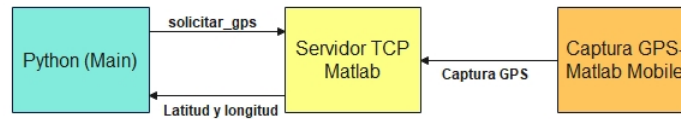


Figura 2.3: Diagrama de comunicación del sistema

Estructura del Diagrama de Flujo

El siguiente flujo de eventos se realiza: MATLAB Mobile ¿MATLAB ¿Python. A través de esto, se captura la información de geolocalización con MATLAB Mobile y se procesa.

1. Inicio:

El código Main debe estar activo. De hecho, el sistema de geolocalización está actualmente inactivo hasta que el sistema avise que se detectarán residuos.

2. Captura de GPS de MATLAB Mobile:

MATLAB Mobile funciona como un cliente de captura de datos. Utiliza el módulo GPS del dispositivo móvil para obtener la latitud y longitud de la ubicación del robot. Cabe mencionar que MATLAB Mobile no almacena datos directamente; se capturan datos en tiempo real y se retransmiten cuando se solicitan.

3. Servidor MATLAB (TCP):

MATLAB en PC: actúa como servidor TCP y espera solicitudes externas de datos GPS. Este servidor:

- Establece la conexión con MATLAB Mobile y permite la captura de GPS.
- Verifica si hay solicitudes activas de Python presentes.
- Continúa hasta que registra las coordenadas GPS cuando sea necesario.

Ahora, técnicamente, esto actúa como un puente de comunicación entre la forma en que el GPS del dispositivo y el Código Main de python.

4. **Solicitud de GPS en Python:**

Python solicita coordenadas GPS solo cuando la IA detecta residuos.

- Activa conexión TCP a MATLAB.
- Envía un mensaje de solicitud (`request_gps`).
- Espera una respuesta del servidor de MATLAB con latitud y longitud.

5. **Recepción de Coordenadas:**

La solicitud llega a MATLAB, que verifica la disponibilidad de los datos GPS y envía la respuesta a Python.

- Si los datos son legítimos (latitud y longitud válidas), se envían inmediatamente
- En caso de datos inválidos (sin GPS disponible), se envía un valor predeterminado (0.000000, 0.000000).
- Python recibe la respuesta y filtra los valores incorrectos antes de procesarlos.

En resumen, se tiene lo siguiente:

1. La IA detecta un residuo → Activa la geolocalización.
2. Python solicita datos GPS al servidor MATLAB.
3. MATLAB Mobile captura la posición y la envía a MATLAB en la PC.
4. MATLAB responde con coordenadas GPS y las transmite a Python.
5. Python almacena la detección y la ubicación GPS en Excel.

2.5. Robótica Cuadrúpeda: Uso del Robot Go1 de Unitree

2.5.1. Introducción a la Robótica Cuadrúpeda

Los robots de cuatro patas siguen los principios de los animales cuadrúpedos; se basan en el andar de cuatro patas para lograr movimiento con más estabilidad y adaptabilidad en terrenos irregulares que los robots mecánicos con ruedas o bipedales. Para lograr un movimiento suave y eficiente.[26]

Características de la Locomoción Cuadrúpeda

- **Mayor estabilidad**→ Reparte el peso a través de cuatro puntos de apoyo, minimizando la probabilidad de caídas.
- **Adaptabilidad en todo tipo de terreno**→ Capaz de navegar en terrenos irregulares, escaleras o entornos urbanos con obstáculos.
- **Eficiencia energética**→ Durante movimientos de larga distancia, reduce el consumo de energía mediante un control optimizado del galope.

2.5.2. Arquitectura del Go1 de Unitree

Unitree Go1: Robot Cuadrúpedo de Exploración, Vigilancia y Automatización. Está optimizado tanto para hardware como software y puede integrarse con modelos de inteligencia artificial, sensores de navegación y comunicación inalámbrica.[27]

Componente	Especificación
Peso	12 kg
Velocidad Máxima	4.7 m/s
Batería	Hasta 2 horas de autonomía (Li-Ion)
Carga Útil	3 - 5 kg
Procesador	NVIDIA Jetson / Raspberry Pi (según configuración)
Sensores	Cámaras de profundidad RGB, Sensor de ultrasonido, IMU
Sistema Operativo	Ubuntu 18.04 / ROS (Robot Operating System)
Protocolos de Comunicación	Ethernet, Wi-Fi, CAN Bus, UART
Modos de Control	Autónomo / Teleoperado
Interfaz de Programación	Python, C++, ROS

Tabla 2.4: Especificaciones Técnicas del Go1 de Unitree

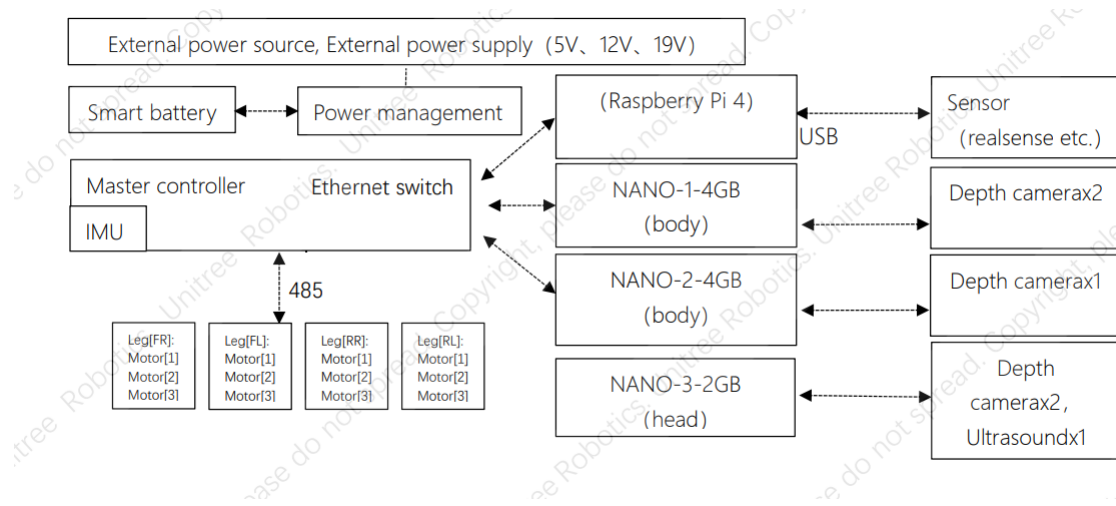


Figura 2.4: Arquitectura Go1

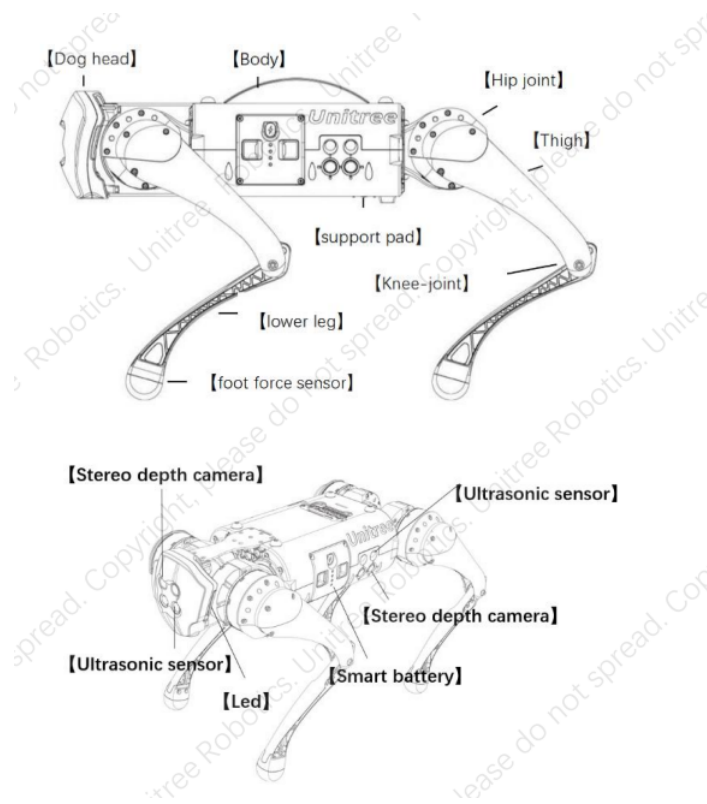


Figura 2.5: Partes Robot

En la Figura 2.4 se puede observar como se encuentran conectadas cada componente del robot y en la Figura 2.5 podemos ver la disposición de cada componente del

robot.

2.5.3. Ventajas del Go1 frente a Sistemas Tradicionales

El robot Unitree Go1 es un avance notable para la robótica móvil aplicada a la vigilancia, monitoreo y exploración en entornos urbanos. Su configuración de cuatro patas y su fusión con sensores avanzados le brindan numerosos beneficios desde la perspectiva de los sistemas de vigilancia tradicionales, ya sean cámaras fijas o drones.

Movilidad en cualquier lugar

El Go1 puede navegar terrenos complejos o incluso superar obstáculos que las cámaras fijas o los sistemas sobre ruedas no pueden.

- **Adaptabilidad en entornos urbanos y rurales**→ Sus patas se encuentran diseñadas para operar en calles, pisos irregulares, estructuras en construcción y ubicaciones industriales.
- **Evasión de obstáculos en tiempo real**→ Con la ayuda de sus sensores IMU y sus sensores ultrasonicos, el Go1 puede evitar colisiones.
- **Accesibilidad en zonas de difícil acceso**→ Puede llegar a espacios estrechos donde drones o vehículos terrestres no pueden ir.

Automatización y controlado a distancia

El Go1 puede operar de manera autónoma o ser teleoperado, brindándole versatilidad operacional.

- **Modo autónomo** → Puede seguir una trayectoria predefinida según algoritmos de navegación.
- **Modo teleoperado**→ Un operador lo controla de manera remota con un controlador inalámbrico o interfaces desde una PC o móvil.

Durabilidad y resistencia

El robot Go1 permite trabajar en condiciones ambientales extremas, desde temperaturas, humedad y vibraciones.

- **Materiales de alta resistencia**→ Su carcasa y estructura de aluminio reforzado lo mantienen seguro en entornos industriales y al aire libre.
- **Resistencia a condiciones climáticas adversas**→ Puede funcionar con lluvia ligera, partículas de polvo o terrenos inclinados medianos.
- **Fiabilidad en seguridad y vigilancia**→ Minimiza la participación humana en actividades de monitoreo en actividades peligrosas.

2.5.4. Limitaciones del Go1 en comparación con los sistemas convencionales

Aunque el robot Go1 muestra grandes ventajas tácticas, los robots cuadrúpedos, siendo el Go1 un modelo ejemplar, tienen limitaciones en comparación con los sistemas de vigilancia y monitoreo convencionales.

Costo y Mantenimiento

El alto costo de adquisición y mantenimiento es uno de los principales factores limitantes.

Mientras un sistema utilizando un robot móvil basado en ruedas tiene un costo aproximado de \$300 USD, el Go1, cuando está configurado con todas las opciones, supera los \$10,000 USD.

El Go1 necesita tener chequeos de rutina para asegurarse de que opere de la mejor manera. Dentro de los chequeos se tiene:

- **Desgaste en motores y actuadores** → Requiere calibración y lubricación regular

- **Baja autonomía de las baterías** → Necesita cargas muy frecuentes o baterías de reemplazo (2h de autonomía).

Cabe mencionar que ante una eventual falla de algún componente, las piezas de repuesto pueden no encontrarse en el mercado local; por ende, el costo de ésta es considerable tanto en lo monetario como en tiempo.

Capítulo 3

Diseño de la arquitectura del sistema

3.1. Requisitos del sistema

Los siguientes requisitos funcionales y no funcionales están definidos para que el sistema de detección y geolocalización de residuos urbanos con el Go1 de Unitree funcione según lo previsto.

3.1.1. Requisitos Funcionales (RF)

Requisitos Funcionales enumeran las funciones específicas requeridas que debe implementar cualquier sistema para alcanzar sus objetivos.

ID	Requisito Funcional
RF-01	El sistema debe capturar imágenes en tiempo real desde la cámara del Dispositivo 1 montado en el robot Go1 utilizando Iriun Webcam.
RF-02	El sistema debería analizar los cuadros de video a través de YOLOv11 para identificar residuos de cartón/papel, metal y plástico.
RF-03	Una vez que se detecte el residuo, el sistema debería solicitar la ubicación GPS del Go1 a través de MATLAB Mobile, aplicación instalada en el Dispositivo 2.

ID	Requisito Funcional
RF-04	El sistema debe almacenar cada detección en un archivo Excel con los siguientes datos: tipo de residuo, nivel de confianza, coordenadas GPS y timestamp.
RF-05	Dentro de un intervalo de 3 segundos, el sistema no debería registrar residuos en el mismo lugar para evitar detecciones duplicadas.
RF-06	La interfaz gráfica debe presentar videos en tiempo real con las detecciones, resaltadas y etiquetadas.
RF-07	El operador debe poder controlar manualmente el Go1 mediante un mando inalámbrico, permitiendo ajustes de posición en el entorno.
RF-08	El sistema debe permitir la exportación de datos en formatos CSV y SQL para análisis posterior.
RF-09	Si MATLAB Mobile no está disponible, el sistema debe notificar al usuario y evitar solicitudes innecesarias.
RF-10	El sistema debe eliminar automáticamente registros duplicados de residuos en el archivo Excel, filtrando por latitud y longitud al finalizar la ejecución.
RF-11	En caso de que el archivo Excel esté en uso, el sistema debe crear un backup automático para evitar la pérdida de datos.

Tabla 3.1: Requisitos Funcionales del Sistema

3.1.2. Requisitos No Funcionales (RNF)

Los requisitos no funcionales se definen como las características de calidad, rendimiento y usabilidad del sistema en general.

ID	Requisito No Funcional
RNF-01	La detección de residuos con YOLOv11 debe tener una precisión mínima del 85 %.
RNF-02	La interfaz gráfica debe actualizarse al menos cada 10 ms para mostrar el video en tiempo real sin retrasos perceptibles.
RNF-03	El sistema debe evitar pérdida de datos en caso de fallos mediante un mecanismo de recuperación automática.
RNF-04	El software debe ser compatible con Python 3.8+, OpenCV, YOLOv11 y MATLAB Mobile.
RNF-05	El almacenamiento en Excel debe manejar correctamente la concurrencia, evitando bloqueos en el acceso al archivo.
RNF-06	El sistema debe funcionar en una computadora con al menos 8GB de RAM y una GPU NVIDIA con soporte CUDA.
RNF-07	En caso de falla en la conexión con MATLAB Mobile, el sistema debe esperar 10 segundos antes de reintentar la solicitud.
RNF-08	El sistema debe ser capaz de operar en exteriores con conexión Wi-Fi.

Tabla 3.2: Requisitos No Funcionales del Sistema

3.2. Arquitectura del sistema

Para una mejor comprensión de este proyecto, se presenta el siguiente diagrama de alto nivel.

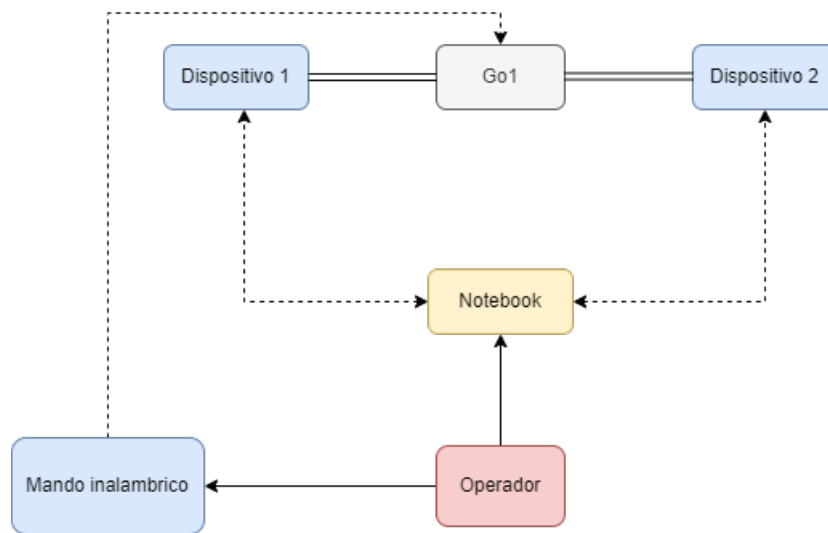


Figura 3.1: Diagrama de Alto Nivel

- **Lineas punteadas:** Representan una comunicación inalámbrica.
- **Lineas continuas:** Indican una interacción directa.
- **Lineas dobles:** Señalan que los objetos están montados.

El diseño de la arquitectura describe cómo los diversos componentes del sistema están organizados e interactúan entre sí. Este sistema consta de tres componentes principales:

1. **Unitree Go1** → Plataforma móvil.
2. **Dispositivos Externos** → Captura de imágenes y GPS.
3. **Notebook del Operador** → Procesa visión artificial y almacena datos.

3.2.1. Robot Go1 de Unitree

- **Función:** Sirve como una plataforma móvil que transporta los dispositivos para la captura de datos.
- **Componentes:**

- a) **Cámara RGB (dispositivo 1)** → Se utiliza para capturar imágenes en tiempo real
- b) **MATLAB Mobile (dispositivo 2)** → Transmite coordenadas GPS.
- c) **Control Remoto Inalámbrico** → Permite controlar manualmente el robot.

3.2.2. Dispositivos Externos

Dispositivo 1 (Cámara montada en el Go1)

- **Función:** Captura y transmite imágenes en tiempo real al Notebook.
- **Tecnología:** Iriun Webcam y OpenCV.
- **Conexión:** Vía Wi-Fi al Notebook.

Dispositivo 2 (Smartphone con MATLAB Mobile)

- **Función:** Captura coordenadas GPS cuando se detecta un residuo.
- **Tecnología:** MATLAB Mobile + Servidor TCP en MATLAB.
- **Conexión:** Comunicación vía TCP/IP con el Notebook.

3.2.3. Notebook del operador (Control y Procesamiento)

- **Funciones:**
 - a) Recibe video en tiempo real desde el Dispositivo 1.
 - b) Procesa imágenes con YOLOv11 para identificar residuos.
 - c) Solicita y recibe datos GPS desde el Dispositivo 2.
 - d) Almacena información en Excel y genera reportes.
- **Tecnologías Utilizadas:**

- a) **Procesamiento de Imágenes:** OpenCV + YOLOv11.
- b) **Geolocalización:** Comunicación con MATLAB Mobile (TCP/IP).
- c) **Almacenamiento de Datos:** Pandas (Excel, SQL).
- d) **Interfaz Gráfica:** Tkinter para visualización en tiempo real.

- **Conectividad del Notebook:**

- a) **Wi-Fi** → Para recibir imágenes desde el Dispositivo 1.
- b) **TCP/IP** → Para obtener datos GPS desde MATLAB Mobile.
- c) **Interfaz de usuario** → Para mostrar el video y control del sistema.

Capítulo 4

Desarrollo del sistema

4.1. Entrenamiento IA usando YOLOv11

La detección de desechos en el sistema se entrenó en Google Colab usando un conjunto de datos personalizado cargado en Google Drive. El modelo utilizado fue un YOLOv11m. El experimento fue configurado para durar 300 épocas, para mejorar la precisión en la detección de desechos de papel/cartón, metal y plástico.

4.1.1. Dataset

Para este proyecto, se utilizó un dataset personalizado compuesto por un conjunto mixto de imágenes obtenidas desde RoboFlow [21] [22] y capturadas de forma manual. Se consideraron únicamente imágenes pertenecientes a las clases cartón/papel, metal y plástico.

El almacenamiento de datos debe seguir una estructura estándar para referenciar datos al implementar YOLO. En resumen, el conjunto de datos debería tener una carpeta raíz que contenga la siguiente estructura: dos subcarpetas train/ con imágenes usadas para entrenamiento y val/ con imágenes usadas para validación. Y también .yaml como un archivo de configuración, en el cual están las rutas de las carpetas train/ y val/, además del número de clases y los nombres de dichas clases.

Dentro de las carpetas `train/` y `val`, debe haber dos subdirectorios más: `images/`, donde se almacenan las imágenes utilizadas para entrenamiento y validación, y `labels/`, donde se guardan los archivos de etiquetas para cada una de las imágenes.

Las anotaciones en `labels/` deben estar en formato YOLO, es decir, cada archivo de texto (`.txt`) contiene líneas que describen los objetos detectados en la imagen, utilizando la estructura: `<clase,  $x_{centro}$ ,  $y_{centro}$ , ancho, alto >` (valores normalizados entre 0 y 1).

Este formato asegura que el modelo YOLO interprete la estructura del conjunto de datos correctamente, permitiéndole proceder sin problemas durante el entrenamiento y la validación, así como durante la inferencia utilizando el marco respectivo.

Listing 4.1: Estructura del dataset

```
dataset/  
  train/  
    images/  
      img1.jpg  
      img2.jpg  
    labels/  
      img1.txt  
      img2.txt  
  val/  
    images/  
      img3.jpg  
      img4.jpg  
    labels/  
      img3.txt  
      img4.txt  
dataset.yaml
```

4.1.2. Código para el entrenamiento

Para la ejecución del entrenamiento, se utilizó Google Colab Pro, que proporciona acceso a unidades de procesamiento gráfico (GPU) dedicadas con una mayor cantidad de memoria de video (VRAM). Esto permitió la utilización de un modelo de YOLO más complejo, así como la configuración de hiperparámetros más avanzados, optimizando el rendimiento del entrenamiento y mejorando la capacidad de detección del modelo.

Listing 4.2: Código para definir la ruta principal

```
!nvidia-smi

import os
HOME = os.getcwd()
print(HOME)
```

- Comprobar la disponibilidad de GPU NVIDIA para acelerar el entrenamiento.
- Ruta principal de trabajo del entorno definida.

Listing 4.3: Instalar paquetes para el uso de YOLOv11

```
%pip install "ultralytics<=8.3.40" supervision roboflow

import ultralytics
ultralytics.checks()
```

- Se instaló YOLOv11 y los paquetes auxiliares supervision y roboflow para ayudarnos a gestionar un conjunto de datos.

- `ultralytics.checks()` asegura que todas las dependencias se hayan instalado correctamente.

Listing 4.4: Importar librerías YOLO

```
from ultralytics import YOLO
from IPython.display import display, Image
```

- Se importa `YOLOv11` para cargar el modelo preentrenado (`yolov11m.pt`).
- Se importa `display` para visualizar imágenes dentro del notebook.

Listing 4.5: Montar Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

- Google Drive montado en `/content/drive` para cargar el dataset y guardar modelos entrenados.

Listing 4.6: Ruta al Dataset

```
dataset_path = "/content/drive/MyDrive/dataset_garbage"
```

```
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"No se encontro el dataset
    en {dataset_path}")
```

- Ruta al Dataset

- Valida que el Dataset ya está presente en Google Drive y solo entonces pasa al entrenamiento.

Listing 4.7: Ruta de guardado del entrenamiento

```
save_path = "/content/drive/MyDrive/YOLOv11_Weights/"  
os.makedirs(save_path, exist_ok=True)
```

- Directorio para guardar modelos entrenados (best.pt).
- La carpeta se creará automáticamente si no existe.

Listing 4.8: Ruta de guardado del entrenamiento

```
!yolo task=detect mode=train model=yolov11m.pt data={  
    dataset_path}/data.yaml epochs=300 imgsz=800 plots=  
    True patience=7 project={save_path} name="  
    garbage_classification"
```

- Modelo base: yolov11m.pt → Transferencia de aprendizaje para mejorar la detección.
- Dataset: Referido bajo data.yaml, que define las subclases papel/cartón, metal y plástico
- Epoch: 300 → para optimizar la precisión del modelo.
- imgsz: 800x800 → solo reajuste para mejorar la calidad de detección.
- Guardado de gráficos (plots=True) → Se generan gráficas de entrenamiento.
- Patience de 10 epochs (patience=10) → Si no hay mejoras en 10 iteraciones, el entrenamiento se detiene automáticamente.

- Ubicación de los pesos entrenados: `save_path` → Los modelos se guardan en Google Drive.

4.2. Código para Captura de Imágenes en Tiempo Real

A continuación se encuentra el código que se conecta con la cámara del Dispositivo 1 y lee imágenes en tiempo real.

Listing 4.9: Captura de Imágenes en Tiempo Real

```
import cv2

print("Conectando a Iriun Webcam en indice 0...")
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

if not cap.isOpened():
    print(" No se pudo abrir la camara Iriun. Asegurate
          de que esta corriendo en tu celular y PC.")
    exit()

print(" Camara Iriun Webcam conectada correctamente.")

while True:
    ret, frame = cap.read()

    if not ret:
        print("No se recibio imagen de la camara.
              Verifica la conexion con Iriun.")
```

```
        break

cv2.imshow("Vista en tiempo real - Iriun Webcam",
           frame)

if cv2.waitKey(1) & 0xFF == ord('q') :
    break

cap.release()
cv2.destroyAllWindows()
```

4.2.1. Explicación del Código

El código existente realiza los siguientes pasos:

1. Inicialización de la Cámara (cv2.VideoCapture(0))

- Se utiliza el índice 0, ya que Iriun Webcam se reconoce como la primera cámara disponible en la PC.
- En caso de que no funcione, se puede probar otros índices manualmente (por ejemplo, cv2.VideoCapture(1), cv2.VideoCapture(2), etc.).

2. Verificación de Conexión

- Si la cámara no se abre correctamente, se imprime un mensaje de error en la terminal y el programa se cierra.

3. Captura y Visualización de Video

- Bucle infinito que captura y muestra en ventana imágenes (fotogramas). Si no se reciben imágenes, se cierra e informa del problema.

4. Terminación del Programa

- Si el usuario pulsa la tecla 'q', la cámara se cierra y se liberan los recursos.

4.3. Procesamiento de Imágenes con YOLOv11

Esta sección explica cómo se integra el modelo YOLOv11 en el sistema de captura de imágenes para la detección de residuos en tiempo real. El modelo procesa los fotogramas de video recogidos por la cámara del Dispositivo 1 (Iriun Webcam) y detecta y clasifica objetos en tres categorías: cartón/papel, metal y plástico.

4.3.1. Carga del Modelo YOLOv11m

El modelo entrenado YOLOv11m se ha guardado en el archivo best.pt, el cual se cargará utilizando la librería Ultralytics. Este modelo se encargará de detectar residuos en tiempo real.

Listing 4.10: Carga del modelo YOLOv11

```
from ultralytics import YOLO

model = YOLO("best.pt")

clsName = ["cardboard-paper", "metal", "plastic"]

print("Modelo YOLOv11m cargado correctamente.")
```

4.3.2. Procesamiento de Imágenes en Tiempo Real

Una vez que el modelo se haya cargado, debe ser aplicado a cada fotograma de video tomado por la cámara. YOLOv11 se aplicará en inferencia de transmisión para maximizar la velocidad en directo.

Listing 4.11: Procesamiento de Imágenes en Tiempo Real

```
import cv2
import math

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

if not cap.isOpened():
    print(" No se pudo abrir la camara Iriun.")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("No se recibio imagen de la camara.")
        break

    results = model(frame, stream=True, verbose=False)

    for res in results:
        for box in res.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])

            cls = int(box.cls[0])
            conf = math.ceil(box.conf[0] * 100)

            if conf < 60:
```

```
        continue

    color = (0, 255, 0) # Verde Carton-papel
    if cls == 1: # Metal
        color = (255, 0, 0) # Azul
    elif cls == 2: # Plastico
        color = (0, 0, 255) # Rojo

    cv2.rectangle(frame, (x1, y1), (x2, y2),
                  color, 2)

    text = f"{clsName[cls]} {conf}%"
    cv2.putText(frame, text, (x1, y1 - 10), cv2.
                FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

cv2.imshow("YOLOv11 - Deteccion de Residuos", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

4.3.3. Explicación del Código

La detección en tiempo real se realiza usando el modelo YOLOv11m en cada frame de Iriun Webcam:

Carga del Modelo YOLOv11m

- Se carga el modelo preentrenado best.pt para realizar la detección de residuos.

Captura de Video desde Iriun Webcam

- Se establece conexión con la cámara del Dispositivo 1 (cv2.VideoCapture(0)).
- Si la cámara no se abre correctamente, el programa termina.

Inferencia en Tiempo Real con YOLOv11m

- Se procesan los cuadros de video mediante model(frame, stream=True).
- Se extraen las coordenadas de las cajas delimitadoras, la clase detectada y su confianza.

Filtrado y Visualización de Resultados

- Las detecciones con baja confianza (menos del 60 %) se ignoran.
- Las cajas delimitadoras se dibujan con diferentes colores según la clase:
 - a) Cartón/Papel → Verde
 - b) Metal → Azul
 - c) Plástico → Rojo
- Se muestra el nombre de la clase junto con la confianza en la imagen.

Salida del Programa

- El sistema se ejecuta en un bucle continuo hasta que el usuario presiona 'q', tras lo cual libera la cámara y cierra la ventana.

4.4. Integración con MATLAB Mobile para Obtención de Coordenadas GPS

En esta sección se aborda la conexión entre el sistema de detección de residuos y el módulo de geolocalización mediante MATLAB Mobile. El objetivo es registrar la ubicación GPS del Dispositivo 2 cada vez que se detecte un residuo, almacenando la información en un archivo Excel para su posterior análisis.

4.4.1. Configuración del Servidor en MATLAB

Para recibir las coordenadas GPS desde MATLAB Mobile, se debe ejecutar un servidor TCP en MATLAB que escuche las solicitudes del sistema Python.

Listing 4.12: Código en MATLAB para Servidor TCP

```
clear all; clc;

server = tcpserver("0.0.0.0", 55000, "
    ConnectionChangedFcn", @(s,e) disp(" Conexion
    establecida"));

disp(" Servidor MATLAB listo. Esperando solicitudes de
    Python...");

while true
    data = readline(server);

    if strcmp(data, "solicitar_gps")
        m = mobiledev;
        pause(2);
```

```

lat = m.Latitude;
lon = m.Longitude;

if isnan(lat) || isnan(lon)
    fprintf(" GPS no disponible. Verifica MATLAB
           Mobile.\n");
    writeline(server, "NaN,NaN");
else
    gps_data = sprintf("%.6f,%.6f", lat, lon);
    writeline(server, gps_data);
    fprintf("GPS enviado: %s\n", gps_data);
end
end
end

```

- Este código en MATLAB abre un servidor TCP en el puerto 55000 y escucha solicitudes de Python.
- Obtiene la latitud y longitud del Dispositivo 2 a través de MATLAB Mobile y las envía de vuelta a Python cuando recibe una solicitud.
- En ausencia de datos GPS, devuelve "NaN,NaN" para evitar registros incorrectos.

4.4.2. Solicitud de Coordenadas GPS desde Python

En el sistema Python, se debe establecer una conexión con el servidor TCP en MATLAB y solicitar las coordenadas cuando se detecte un residuo.

Listing 4.13: Solicitud de Coordenadas GPS desde Python

```
import socket
```

```
import time

MATLAB_HOST = "localhost"
MATLAB_PORT = 55000

lat_gps = None
lon_gps = None
ultimo_tiempo_gps = 0

def check_matlab_server():

    try:
        test_socket = socket.create_connection((
            MATLAB_HOST, MATLAB_PORT), timeout=1)
        test_socket.close()
        return True
    except (socket.timeout, ConnectionRefusedError):
        return False

def solicitar_gps():
    global lat_gps, lon_gps, ultimo_tiempo_gps

    if time.time() - ultimo_tiempo_gps < 3:
        return

    if not check_matlab_server():
```

```
print(" MATLAB no esta disponible. Omitiendo
      solicitud de GPS.")
return

try:
    client_socket = socket.socket(socket.AF_INET,
                                   socket.SOCK_STREAM)
    client_socket.connect((MATLAB_HOST, MATLAB_PORT)
                           )
    client_socket.sendall(b"solicitar_gps\n")

    data = client_socket.recv(1024).decode("utf-8").
        strip()
    lat, lon = data.split(",")

    if lat == "NaN" or lon == "NaN":
        print(" No se pudo obtener GPS.")
    else:
        lat_gps = lat
        lon_gps = lon
        ultimo_tiempo_gps = time.time()
        print(f" GPS recibido: Latitud {lat_gps},
              Longitud {lon_gps}")

    client_socket.close()
except Exception as e:
    print(f" No se pudo obtener GPS: {e}")
```

Configuración de la Conexión con MATLAB

- **MATLAB_HOST:** Especifica la dirección IP donde MATLAB está ejecutando el servidor TCP.
- **MATLAB_PORT:** el puerto de escucha de MATLAB para la solicitud.
- **MAX_INTENTOS:** numero máximo de intentos en caso de que MATLAB no responda inmediatamente.

Si el código de MATLAB se ejecuta en la misma máquina que Python, se usa "localhost". Si MATLAB se ejecuta en alguna otra máquina, se debe reemplazar "localhost" por la IP de esa otra máquina.

Creación del Socket y Solicitud de GPS

- Se crea un socket TCP/IP con socket.AF_INET y socket.SOCK_STREAM.
- Se establece un timeout de 3 segundos para evitar bloqueos si MATLAB no responde.
- Se intenta conectar al servidor MATLAB en el puerto 55000.

Envío de la Solicitud de GPS

- Se envía el mensaje "*solicitar_gps*" al servidor MATLAB para que este devuelva la ubicación.

Recepción de Datos y Manejo de Errores

- Se recibe la respuesta de MATLAB, que debe contener las coordenadas GPS en el formato [-33.453120, -70.651154]
- Si MATLAB no pudo obtener datos de GPS, podría devolver "*NaN, NaN*"

Manejo de Fallos de Conexión

- Si MATLAB no responde en 3 segundos, se muestra una alerta y se reintenta.
- Errores manejados:
 - **socket.timeout:** Ocurre cuando MATLAB no responde en el tiempo esperado.
 - **ConnectionRefusedError:** Ocurre cuando el servidor de MATLAB no está ejecutándose.

4.5. Almacenamiento de Datos en Excel

En esta sección, abordaremos cómo el sistema almacena las detecciones de residuos junto con sus coordenadas GPS en un archivo Excel utilizando la librería pandas en Python.

4.5.1. Objetivo del Módulo de Almacenamiento

El sistema debe registrar todas las detecciones de residuos junto con la información de geolocalización en un archivo Excel para su posterior análisis.

Datos que se almacenan en cada detección:

- Clase del residuo detectado (cardboard-paper, metal, plastic).
- Nivel de confianza en la detección.
- Latitud y Longitud obtenidas desde MATLAB Mobile.
- Fecha y Hora en que se registró la detección.

Listing 4.14: Código para Guardar Detecciones en Excel

```
import pandas as pd
import os

filename = "detecciones_residuos.xlsx"
backup_filename = "detecciones_residuos_backup.xlsx"

header = ["Clase", "Confianza", "Latitud", "Longitud", "
    Fecha y Hora"]

lat_gps = None
lon_gps = None

def guardar_deteccion(clase, confianza):

    global lat_gps, lon_gps

    if lat_gps is None or lon_gps is None:
        print(" No se guardo la deteccion porque no hay
            GPS disponible.")
        return

    if confianza < 60:
        print(f"Deteccion de {clase} ignorada (confianza
            {confianza}%")
        return

    nueva_deteccion = pd.DataFrame([[clase, confianza,
        lat_gps, lon_gps, pd.Timestamp.now()]], columns=
```

```
header)

try:
    if os.path.exists(filename):
        df = pd.read_excel(filename)
        df = pd.concat([df, nueva_deteccion],
                        ignore_index=True)
    else:
        df = nueva_deteccion

    df.to_excel(filename, index=False)
    print(f"Deteccion guardada: {clase} ({confianza
        }%) en {lat_gps}, {lon_gps}")

except PermissionError:
    print(" No se pudo guardar en Excel. Creando
        copia de seguridad...")
    df.to_excel(backup_filename, index=False)
```

Validación de datos GPS antes de guardar

- Si la ubicación GPS aún no ha sido recibida, la detección no se almacena.

Filtrado de detecciones con baja confianza

- Se descartan detecciones con una confianza menor al 60 %, evitando falsos positivos.

Manejo de concurrencia en Excel

- Si el archivo Excel está en uso, se crea un respaldo (detecciones_residuos_backup.xlsx) en caso de error.

4.5.2. Eliminación Automática de Registros Duplicados

Para evitar registros redundantes en el archivo Excel, se debe filtrar por latitud y longitud. Se implementa una función que limpia los datos al finalizar la ejecución del sistema.

Listing 4.15: Código para la eliminación Automática de Registros Duplicados

```
def eliminar_duplicados():  
  
    if os.path.exists(filename):  
        df = pd.read_excel(filename)  
  
        df = df.drop_duplicates(subset=["Latitud", "  
            Longitud"], keep="first")  
  
        df.to_excel(filename, index=False)  
        print(" Registros duplicados eliminados  
            correctamente.")
```

La función se ejecutará antes de cerrar el programa para garantizar que no haya entradas repetidas en el archivo de detecciones.

4.6. Módulo de Visualización

- El video en vivo capturado desde la cámara del Dispositivo 1 (Iriun Webcam).

- Las detecciones resaltadas y etiquetadas con bounding boxes (YOLOv11m).
- El nivel de confianza y el tipo de residuo detectado.

Listing 4.16: Código para el módulo de Visualización

```
import cv2
import time
import threading
import tkinter as tk
from tkinter import Label, PhotoImage
from PIL import Image, ImageTk
from ultralytics import YOLO
import math

model = YOLO("best.pt")

clsName = ["cardboard-paper", "metal", "plastic"]

ventana = tk.Tk()
ventana.title("Sistema de Deteccion de Residuos -
YOLOv11m")
ventana.geometry("1280x720")

imagenF = PhotoImage(file="setUp/Deteccion de basura.png
")
background = Label(ventana, image=imagenF)
background.place(x=0, y=0, relwidth=1, relheight=1)
```

```
lblVideo = Label(ventana)
lblVideo.place(x=320, y=120) o

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

if not cap.isOpened():
    print(" No se pudo abrir la camara Iriun.")
    exit()

def actualizar_video():
    ret, frame = cap.read()
    if not ret:
        print("No se recibio imagen de la camara.")
        return

    results = model(frame, stream=True, verbose=False)

    for res in results:
        for box in res.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cls = int(box.cls[0])
            conf = math.ceil(box.conf[0] * 100)

            if conf < 60:
                continue

            color = (0, 255, 0) # Verde Cardboard/paper
```

```
if cls == 1: # Metal
    color = (255, 0, 0) # Azul
elif cls == 2: # Plastico
    color = (0, 0, 255) # Rojo

cv2.rectangle(frame, (x1, y1), (x2, y2),
              color, 2)

text = f"{clsName[cls]} {conf}%"
cv2.putText(frame, text, (x1, y1 - 10), cv2.
            FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
img = Image.fromarray(frame)
imgtk = ImageTk.PhotoImage(image=img)

lblVideo.imgtk = imgtk
lblVideo.configure(image=imgtk)
lblVideo.after(10, actualizar_video)

actualizar_video()
ventana.mainloop()

cap.release()
```

Configuración de la Interfaz Gráfica (Tkinter)

- Se crea la ventana principal con Tkinter `tk.Tk()`.
- Se establece un título para la interfaz.

- Se define un tamaño de ventana de 1280x720 píxeles, ajustado al diseño de la imagen de fondo.

Carga de la Imagen de Fondo

- Se carga una imagen de fondo (Deteccion de basura.png) desde la carpeta setUp/.
- Se crea un Label que contiene la imagen de fondo.
- Se usa .place(x=0, y=0, relwidth=1, relheight=1) para asegurarse de que la imagen cubra toda la ventana.

Posicionamiento del Video Sobre la Imagen de Fondo

- Se crea un Label (lblVideo) que servirá para mostrar el video procesado por YOLOv11m.
- Se coloca en una posición específica (x=320, y=120) para alinearlo con la imagen de fondo.

Captura de Video desde Iriun Webcam

- Se abre la cámara Iriun Webcam usando cv2.VideoCapture(0).
- Se verifica que la cámara esté disponible antes de continuar.
- Si la cámara no responde, se muestra un mensaje de error y se detiene el programa.

Actualización del Video en la Interfaz Gráfica

- Se lee un frame de la cámara cap.read().
- Si la cámara deja de enviar imágenes, se muestra un mensaje de error y se detiene el proceso.

Procesamiento del Video con YOLOv11

- Se aplica YOLOv11m sobre el frame capturado.
- Se identifican los objetos detectados y se extraen sus coordenadas (x1, y1, x2, y2).
- Se descartan detecciones con confianza menor a 60 %.
- Se dibujan bounding boxes alrededor de los objetos detectados.
- Se muestra la clase del objeto junto con su nivel de confianza.

Conversión de la Imagen para Mostrarla en la Interfaz

- Convierte la imagen capturada a formato RGB (necesario para Tkinter).
- Convierte la imagen a un formato compatible con PhotoImage de Tkinter.
- Muestra la imagen en lblVideo.
- Actualiza la imagen cada 10 ms para mantener el video en tiempo real.

Iniciar la Interfaz y Mostrar el Video

- Llama a la función actualizar_video() para comenzar a mostrar la detección en vivo.
- Ejecuta ventana.mainloop() para mantener la interfaz abierta hasta que el usuario la cierre.

Liberar la Cámara al Cerrar la Ventana

- Se libera la cámara cuando la ventana se cierra para evitar bloqueos.

4.7. Implementación de todos los módulos

4.7.1. Montaje

Para la instalación de los dos dispositivos necesarios para el correcto funcionamiento del proyecto, se implementaron soluciones de montaje adaptadas a cada uno. En el caso del primer dispositivo, fue necesario adquirir un soporte para celulares Fig.4.1 al cual se le integró una pieza impresa en 3D Fig.4.2, diseñada específicamente para garantizar un ajuste seguro en la estructura del robot. Por otro lado, el montaje del segundo dispositivo fue más sencillo, ya que, mediante el uso de una carcasa, se logró un anclaje estable al adherirla con adhesivo directamente a la superficie del robot.



Figura 4.1: Soporte NEEWER SP-10

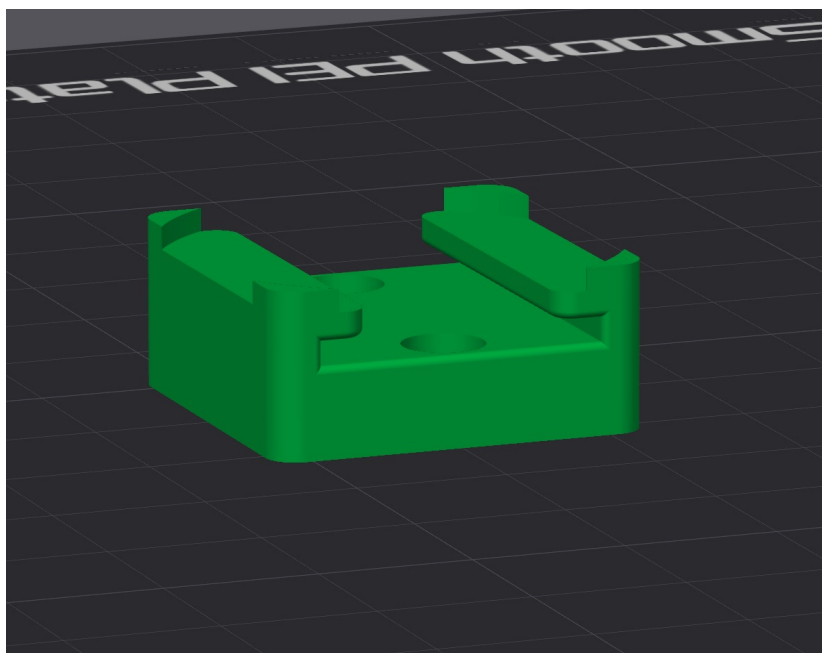


Figura 4.2: Pieza impresa 3d



Figura 4.3: Montaje Final

4.7.2. Iniciar aplicaciones

Una vez montados los dispositivos, es necesario iniciar la aplicación Iriun Webcam tanto en el notebook como en el Dispositivo 1, asegurando que todos los dispositivos estén conectados a la misma red. Posteriormente, se debe verificar que la señal de video se está transmitiendo correctamente al notebook. Una vez confirmada la recepción del video, se procede a iniciar MATLAB Mobile en el Dispositivo 2. Dentro de la aplicación, es necesario acceder al apartado de sensores y activar la opción de geolocalización para habilitar el registro de la posición del robot.

4.7.3. Iniciar códigos

Para garantizar el correcto funcionamiento del proyecto, es necesario ejecutar, en primer lugar, el código de MATLAB 4.12 encargado de montar el servidor para compartir la geolocalización del robot. Una vez que el servidor esté operativo y la transmisión de datos confirmada, se procede a ejecutar el código Main final, que integra todas las funcionalidades del sistema y permite la detección, clasificación y registro de residuos junto con sus respectivas coordenadas geográficas.

Listing 4.17: Código para el módulo de Visualización

```
import cv2
import time
import threading
import socket
import pandas as pd
import os
import tkinter as tk
from tkinter import Label, PhotoImage
from PIL import Image, ImageTk
```

```
from ultralytics import YOLO
import math

model = YOLO("best.pt")

clsName = ["cardboard-paper", "metal", "plastic"]

MATLAB_HOST = "localhost"
MATLAB_PORT = 55000

filename = "detecciones_residuos.xlsx"
backup_filename = "detecciones_residuos_backup.xlsx"

lat_gps = None
lon_gps = None
ultimo_tiempo_gps = 0

ventana = tk.Tk()
ventana.title("Sistema de Deteccion de Residuos -
YOLOv11m")
ventana.geometry("1280x720")

imagenF = PhotoImage(file="setUp/Deteccion de basura.png
")
background = Label(ventana, image=imagenF)
background.place(x=0, y=0, relwidth=1, relheight=1)

lblVideo = Label(ventana)
```

```
lblVideo.place(x=320, y=120)

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

if not cap.isOpened():
    print("No se pudo abrir la camara Iriun.")
    exit()

def solicitar_gps():
    global lat_gps, lon_gps, ultimo_tiempo_gps

    if time.time() - ultimo_tiempo_gps < 3:
        return

    try:
        client_socket = socket.socket(socket.AF_INET,
                                       socket.SOCK_STREAM)
        client_socket.settimeout(3)
        client_socket.connect((MATLAB_HOST, MATLAB_PORT)
                               )
        client_socket.sendall(b"solicitar_gps\n")

        data = client_socket.recv(1024).decode("utf-8").
            strip()
        client_socket.close()

        lat, lon = data.split(",")
```

```
if lat.lower() == "nan" or lon.lower() == "nan":
    print("No se recibio una ubicacion valida
          desde MATLAB.")
else:
    lat_gps = float(lat)
    lon_gps = float(lon)
    ultimo_tiempo_gps = time.time()
    print(f"GPS recibido: Latitud {lat_gps},
          Longitud {lon_gps}")

except (socket.timeout, ConnectionRefusedError) as e
:
    print(f"No se pudo obtener GPS: {e}")

def guardar_deteccion(clase, confianza):
    global lat_gps, lon_gps

    if lat_gps is None or lon_gps is None:
        print("No se guardo la deteccion porque no hay
              GPS disponible.")
        return

    if confianza < 50:
        print(f"Deteccion de {clase} ignorada (confianza
              {confianza}%)")
        return
```

```
nueva_deteccion = pd.DataFrame([[clase, confianza,
                                lat_gps, lon_gps, pd.Timestamp.now()]],
                                columns=["Clase", "
                                         Confianza", "
                                         Latitud", "Longitud
                                         ", "Fecha y Hora"])

try:
    if os.path.exists(filename):
        df = pd.read_excel(filename)
        df = pd.concat([df, nueva_deteccion],
                        ignore_index=True)
    else:
        df = nueva_deteccion

    df.to_excel(filename, index=False)
    print(f"Deteccion guardada: {clase} ({confianza
        }%) en {lat_gps}, {lon_gps}")

except PermissionError:
    print("No se pudo guardar en Excel. Creando
          copia de seguridad...")
    df.to_excel(backup_filename, index=False)

def eliminar_duplicados():
    if os.path.exists(filename):
        df = pd.read_excel(filename)
```

```
df = df.drop_duplicates(subset=["Latitud", "
    Longitud"], keep="first")
df.to_excel(filename, index=False)
print("Registros duplicados eliminados
    correctamente.")

def cerrar_programa(event):
    print("Cerrando programa...")
    cap.release()
    eliminar_duplicados()
    ventana.destroy()
    print("Programa cerrado correctamente.")

def actualizar_video():
    ret, frame = cap.read()
    if not ret:
        print("No se recibio imagen de la camara.")
        return

    results = model(frame, stream=True, verbose=False)

    for res in results:
        for box in res.bboxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cls = int(box.cls[0])
            conf = math.ceil(box.conf[0] * 100)

            if conf < 50:
```

```
        continue

    color = (0, 255, 0)

    if cls == 1:
        color = (255, 0, 0)
    elif cls == 2:
        color = (0, 0, 255)

    cv2.rectangle(frame, (x1, y1), (x2, y2),
                  color, 2)
    text = f"{clsName[cls]} {conf}%"
    cv2.putText(frame, text, (x1, y1 - 10), cv2.
                FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

    if time.time() - ultimo_tiempo_gps >= 3:
        hilo_gps = threading.Thread(target=
            solicitar_gps)
        hilo_gps.start()
        guardar_deteccion(clsName[cls], conf)

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    img = Image.fromarray(frame)
    imgtk = ImageTk.PhotoImage(image=img)

    lblVideo.imgtk = imgtk
    lblVideo.configure(image=imgtk)
    lblVideo.after(10, actualizar_video)
```

```
ventana.bind("<KeyPress-q>", cerrar_programa)

actualizar_video()

ventana.mainloop()
```

4.7.4. Manejo del robot

En primer lugar, es fundamental verificar el nivel de carga de las baterías tanto del robot como del mando. Cabe destacar que el mando es un periférico incluido con el robot.

Para iniciar el robot, este debe encontrarse en su posición de reposo Fig.4.4). Una vez correctamente posicionado de forma manual, se debe presionar dos veces el botón de encendido de la Smart Battery Fig.2.5, asegurándose de mantener presionado el botón en la segunda pulsación hasta que el robot se encienda.

Una vez encendido el robot, se procede a iniciar el mando. Para ello, se debe presionar dos veces su botón de poder Fig.4.6, manteniéndolo presionado en la segunda pulsación hasta que el dispositivo se active correctamente.

El robot se encuentra en modo reposo, un estado en el que optimiza el consumo energético. Para este proyecto, es necesario que el robot ingrese en modo caminar, lo que permitirá su control mediante los sticks análogos del mando. Para activar este modo, se deben seguir los siguientes pasos:

1. **Modo de mantenerse en pie:** Presionar la combinación de botones L2 + A. Esta acción hará que el robot pase al modo mantenerse en pie, donde ajustará la activación de sus motores para sostenerse de manera estable.
2. **Modo Caminar:** Presionar nuevamente la combinación L2 + A. Esto permitirá que el robot cambie al modo caminar. Finalmente, para iniciar los movimientos, se debe presionar el botón START en el mando.

Una vez activado el modo caminar, el robot puede ser controlado mediante los

sticks analógicos del mando. El stick izquierdo permite desplazar el robot en diferentes direcciones, incluyendo avance, retroceso y movimientos laterales. Por otro lado, el stick derecho controla la rotación del robot, permitiendo realizar giros.

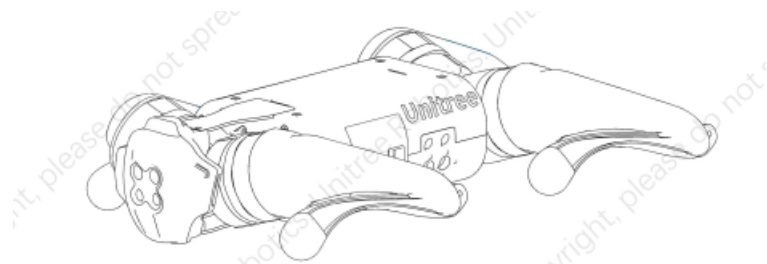


Figura 4.4: Posición Go1 en reposo

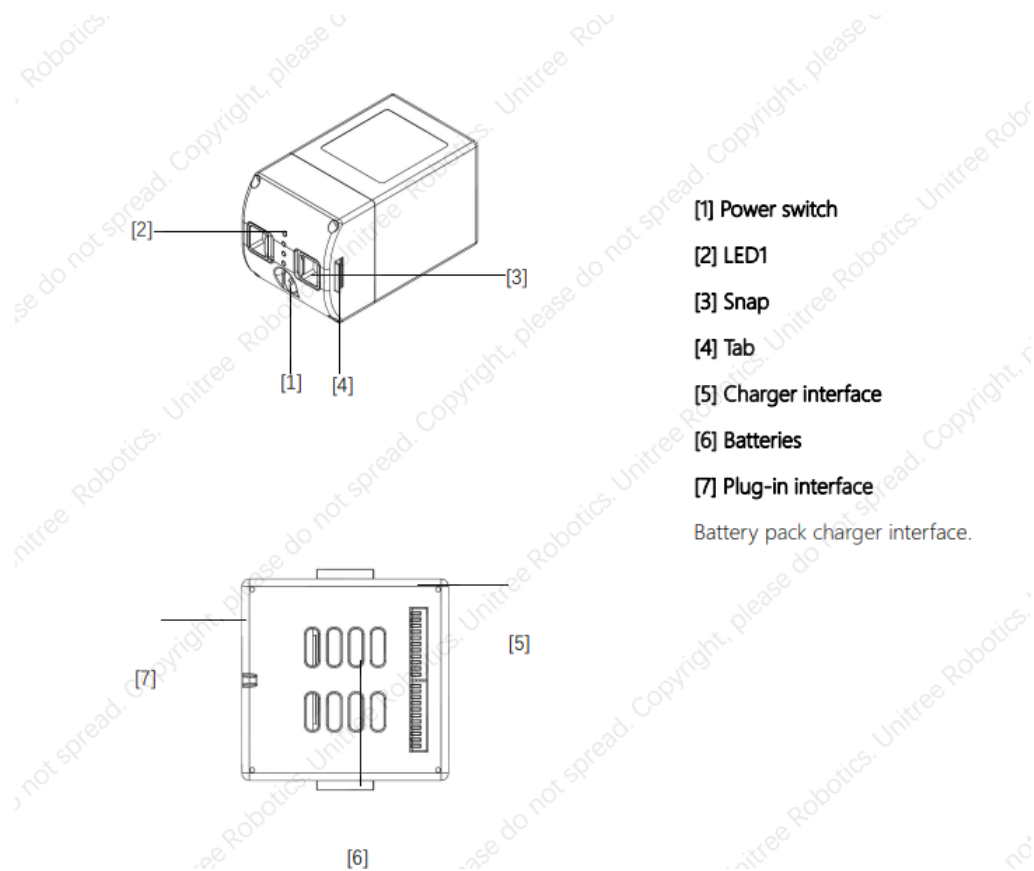


Figura 4.5: Batería inteligente

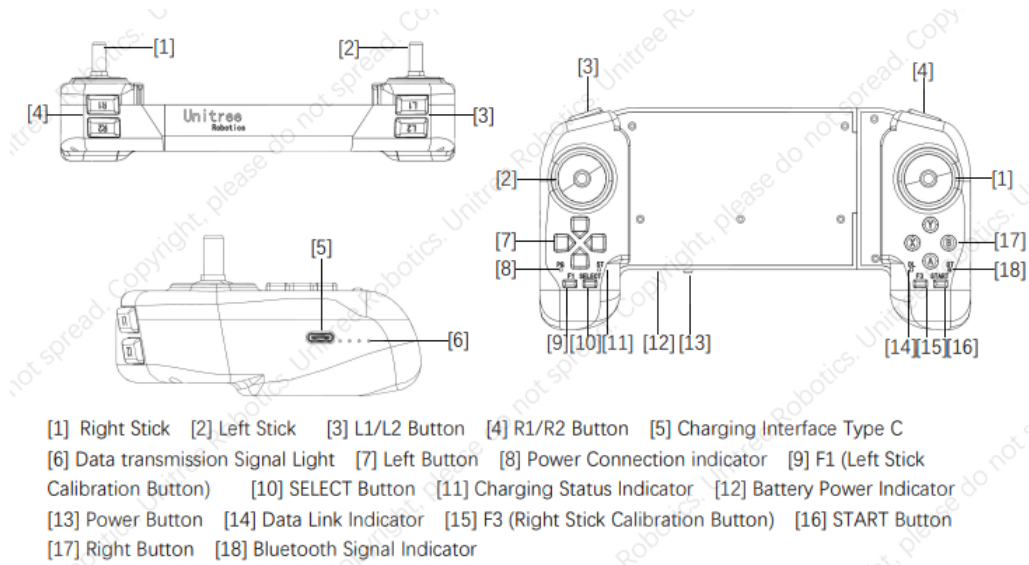


Figura 4.6: Mando Go1

4.7.5. Patrullaje

Con los códigos en ejecución y el robot en modo caminar, el operador podrá visualizar la transmisión de video proveniente de la cámara del Dispositivo 1 Fig.4.7 y controlar el desplazamiento del robot mediante el mando. En esta fase final del proyecto, el operador seguirá una ruta predefinida mientras supervisa la transmisión en tiempo real, donde se evidenciará la detección de residuos por parte del sistema.

Una vez que un residuo es identificado, el sistema almacenará en un archivo Excel la información correspondiente a su clase, nivel de confianza, ubicación geográfica (latitud y longitud), así como la fecha y hora del registro.

Al finalizar el patrullaje, el operador podrá cerrar el programa presionando la tecla "q", lo que provocará el cierre de la interfaz gráfica y la ejecución de un proceso de post-análisis en el archivo Excel. Durante esta etapa, el sistema eliminará registros duplicados para evitar redundancias en los datos recopilados.

Finalmente, se debe colocar el robot en modo reposo presionando la combinación de teclas L2 + B, apagar el robot manteniendo presionado el botón de encendido durante unos segundos y, posteriormente, apagar el mando siguiendo el mismo procedimiento. Una vez finalizado el proceso, el robot deberá ser almacenado de manera segura.

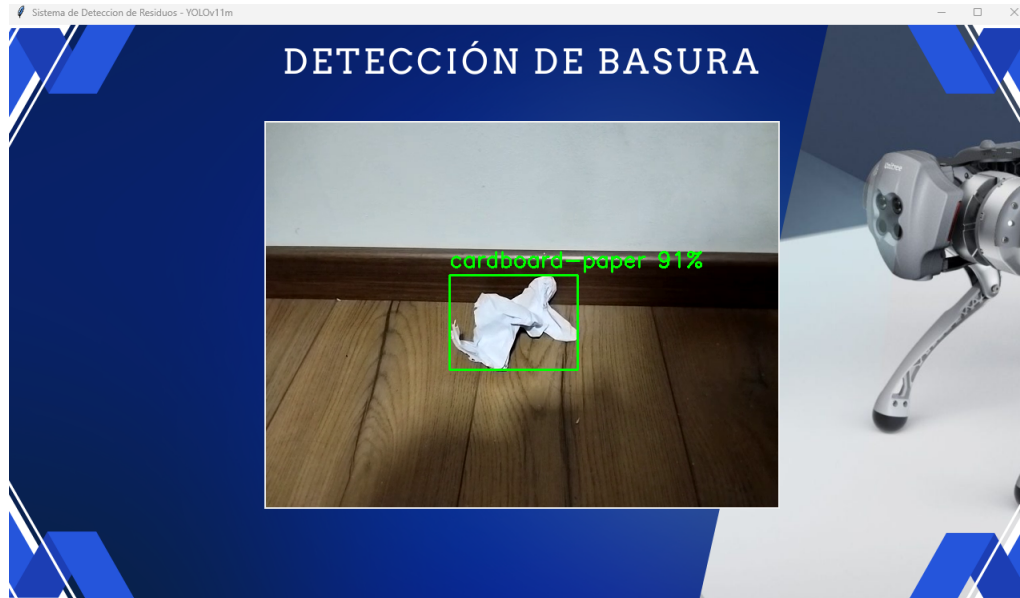


Figura 4.7: Interfaz gráfica

Capítulo 5

Resultados

5.1. Entrenamiento IA

5.1.1. Evaluación de la Pérdida Durante el Entrenamiento

En la Fig.5.1 y en la Fig.5.2 se observan las curvas de pérdida (loss curves) tanto para los datos de entrenamiento como de validación. Se destacan cuatro tipos de pérdida:

- **box_loss:** Se refiere a la precisión de las cajas delimitadoras (bounding boxes). La curva muestra una disminución constante, indicando que el modelo mejora progresivamente en la localización de objetos.
- **cls_loss:** Relacionada con la clasificación de los objetos detectados. La reducción de la pérdida a lo largo de las épocas sugiere que el modelo se está ajustando correctamente a las categorías de residuos.
- **seg_loss y dfl_loss:** Indicadores de segmentación y calidad de las predicciones de objetos. La convergencia de estas curvas a valores bajos confirma una mejora en la precisión del modelo.

El modelo se entrenó de manera eficiente, con pérdidas estabilizándose alrededor de valores bajos, lo que indica que no hay sobreajuste significativo.

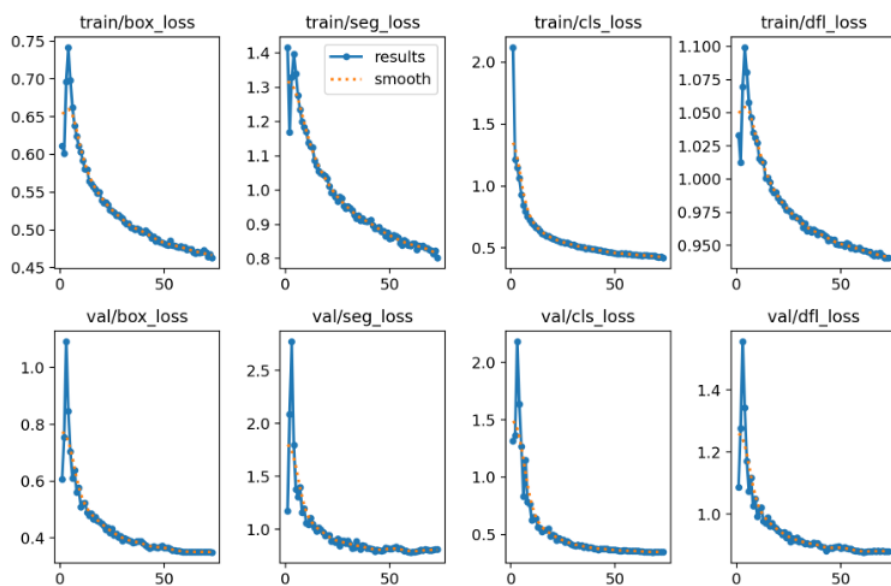


Figura 5.1: Resultados del entrenamiento

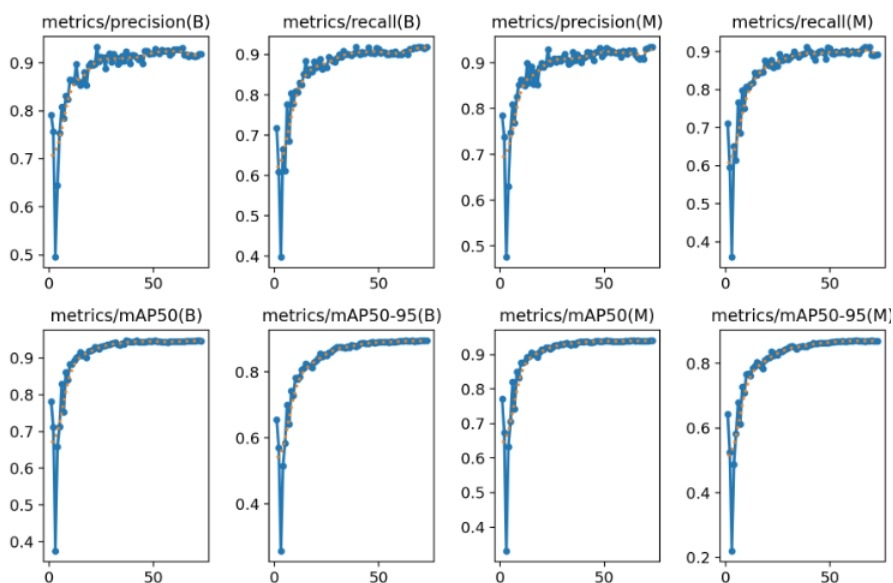


Figura 5.2: Resultados del entrenamiento

5.1.2. Matriz de Confusión Normalizada

La matriz de confusión Fig.5.3 permite evaluar el desempeño del modelo al clasificar correctamente cada categoría de residuos.

- **Cardboard-paper:** Se clasifica correctamente en 88 % de los casos, aunque presenta una tasa de confusión del 11 % con el fondo (background).
- **Metal:** La precisión es muy alta (96 %), con una mínima confusión con otras clases.
- **Plastic:** Se detecta con un 97 % de precisión, con confusión marginal con background.

La matriz de confusión confirma un alto rendimiento del modelo, con bajos niveles de error entre clases, aunque persisten algunas confusiones entre cardboard-paper y el fondo.

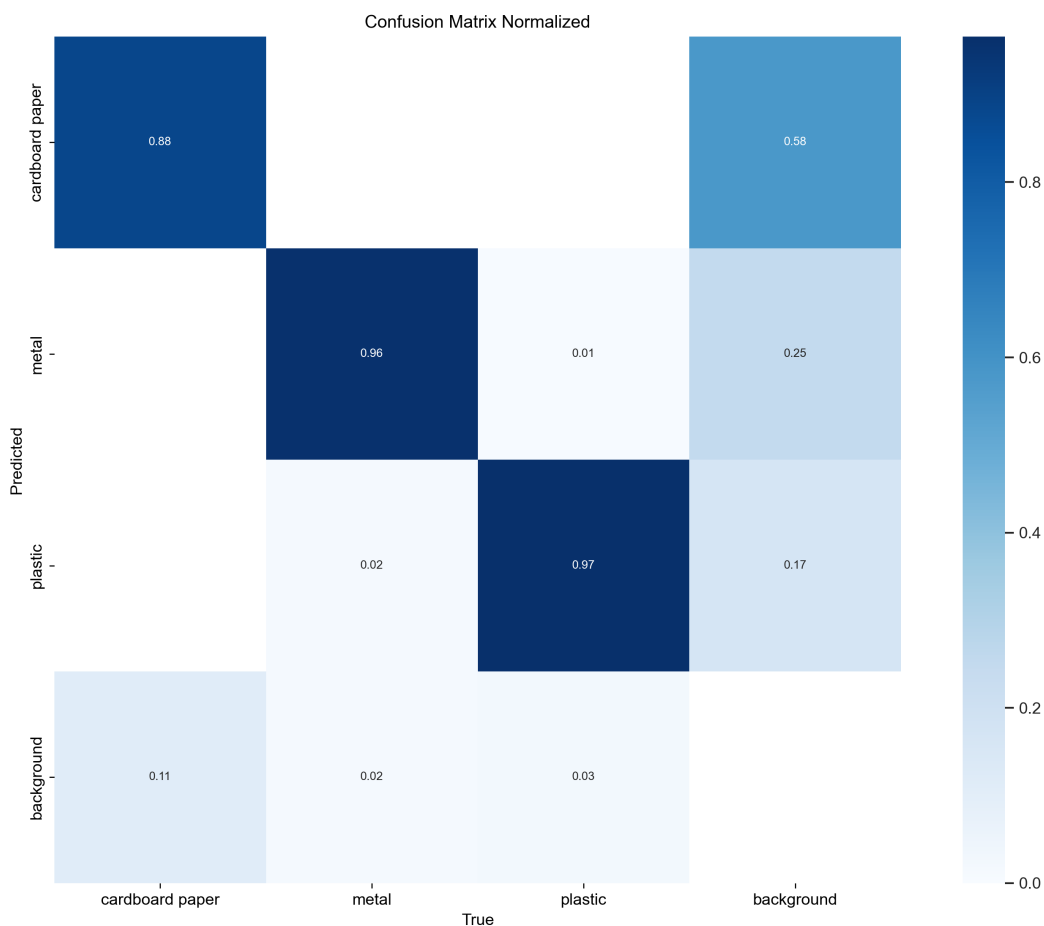


Figura 5.3: Matriz de Confusión Normalizada

5.1.3. Curvas de Precisión-Recall (PR)

Las curvas de Precision-Recall (PR) Fig.5.4 Fig.5.5 muestran cómo varía la precisión en función de la tasa de recall:

- Metal y Plastic alcanzan una precisión superior al 97 %, lo que indica que el modelo tiene un excelente rendimiento para estas clases.
- Cardboard-paper presenta una precisión de ~ 86 %, lo que sugiere margen de mejora en la clasificación de esta categoría.

El modelo tiene un buen equilibrio entre precisión y recall, con mejores resultados en metal y plastic, mientras que cardboard-paper podría beneficiarse de un dataset más balanceado o mejoras en la anotación.

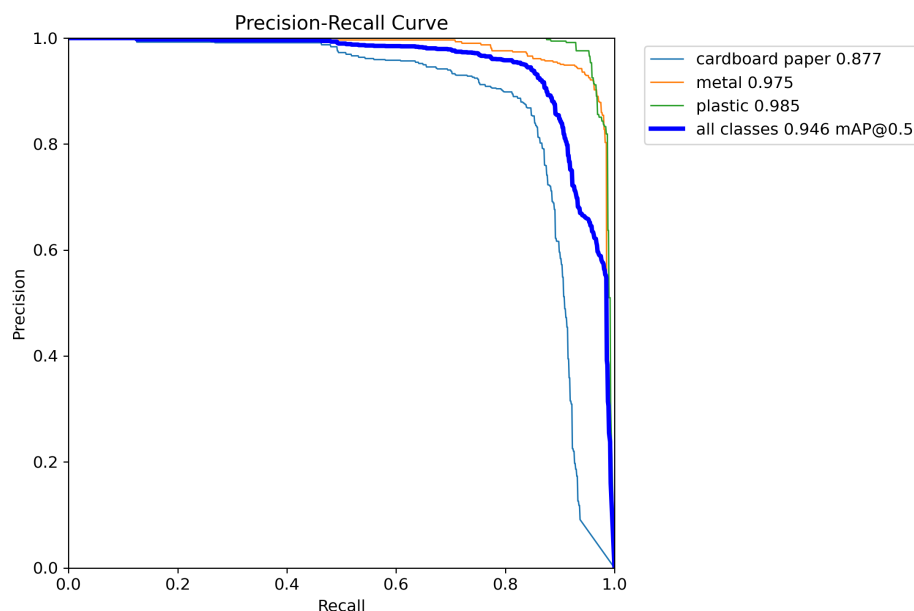


Figura 5.4: Box Precision-Recall

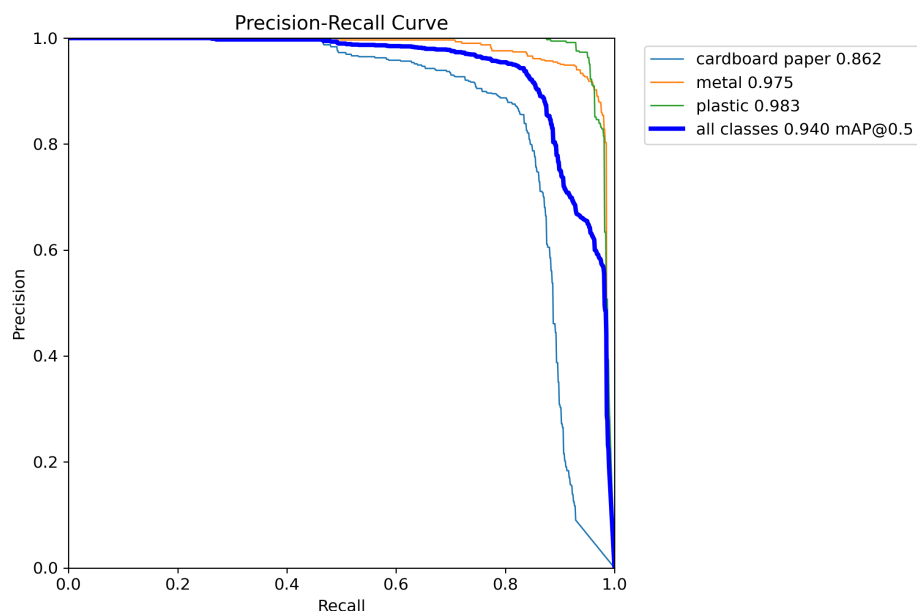


Figura 5.5: Mask Precision-Recall

5.1.4. Curvas F1-Score

Las curvas F1-Confidence Fig.5.6 y Fig.5.11, reflejan el punto óptimo de equilibrio entre precisión y recall:

F1-Score global alcanza 0.91, lo que indica un desempeño robusto del modelo. Clases metal y plastic muestran los valores más altos de F1 ($> 0,95$), mientras que cardboard-paper está ligeramente por debajo ($\sim 0,86$).

El modelo funciona bien en términos de balance entre precisión y recall, aunque se observa que cardboard-paper es la clase con menor rendimiento relativo.

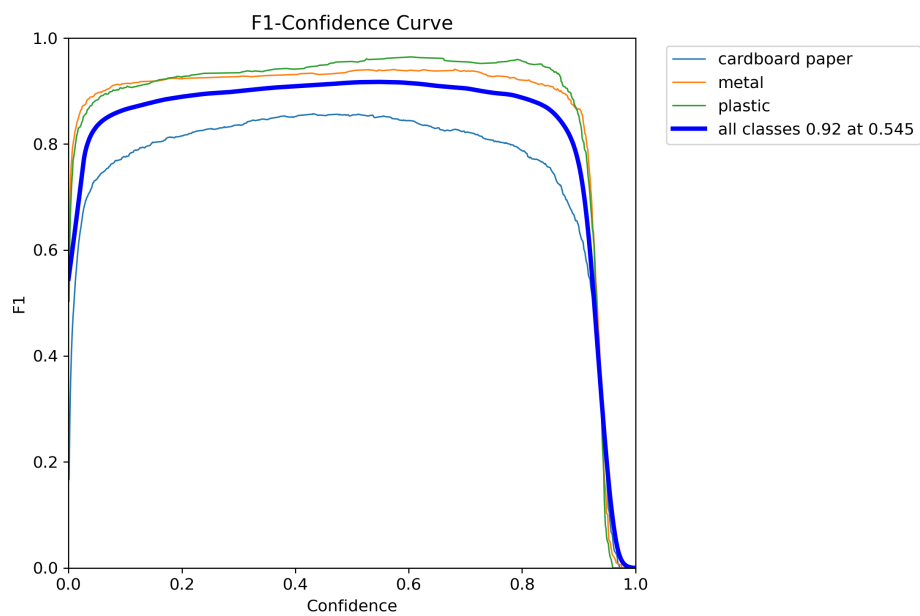


Figura 5.6: Box F1 Precision-Recall

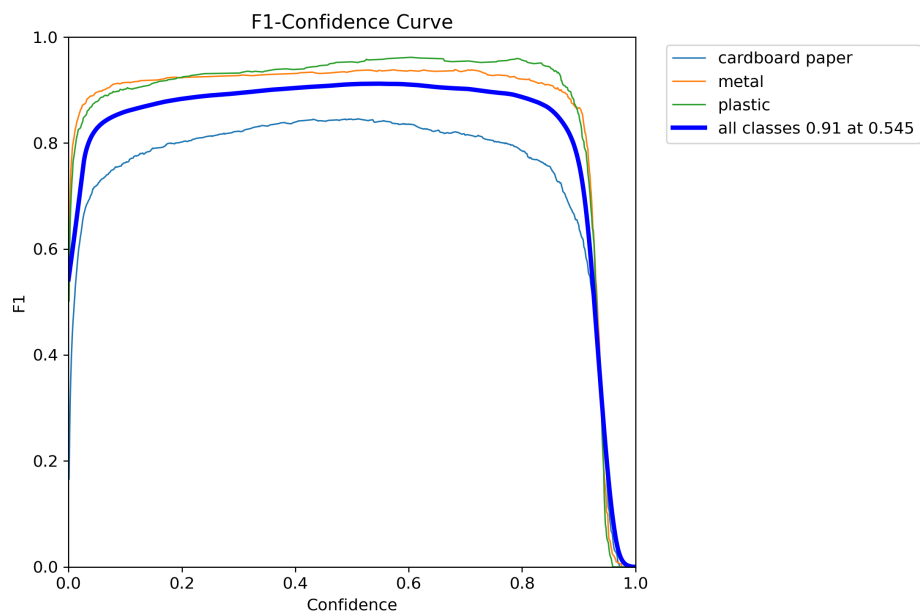


Figura 5.7: Mask F1 Precision-Recall

5.1.5. Modelo Entrenado

El modelo YOLOv11 entrenado muestra un desempeño sólido y efectivo, con métricas superiores al 85 % en todas las clases y un F1-score global de 0.91. Sin embargo, hay margen de mejora en la detección de cardboard-paper, donde se podría optimizar la clasificación con más datos o ajustes en el preprocesamiento.

5.2. Prueba del sistema

Para la prueba del sistema, se identificaron ubicaciones que contienen residuos de cada una de las clases evaluadas. Una vez iniciado el sistema, siguiendo los procedimientos descritos en los apartados 4.7.4 y 4.7.5, se procede a controlar el robot a lo largo de una ruta predefinida que atraviesa estos puntos, permitiendo así la detección y registro de los residuos en condiciones reales de operación.



Figura 5.8: Reconocimiento Cardboard



Figura 5.9: Reconocimiento Papel



Figura 5.10: Reconocimiento Metal



Figura 5.11: Reconocimiento Plastic

Una vez finalizada la ruta de patrullaje, se procedió con el cierre del sistema y la revisión del archivo Excel generado (Figura 5.12). Al analizar los datos registrados, se verificó que estos eran consistentes con la ruta seguida por el robot y con las ubicaciones en las que se encontraban los residuos detectados (se comparó con Google Maps).

No obstante, se identificaron algunos falsos positivos. Uno de ellos correspondió a la clase cardboard-paper, donde el sistema interpretó erróneamente el piso como un residuo de esta categoría. Otro caso se presentó en la clasificación de un residuo de plástico, que fue confundido con metal. Tal como se mencionó en el inciso 5.1.2, estas imprecisiones pueden atribuirse a la influencia del fondo en la detección y a los efectos de la iluminación, los cuales pueden generar variaciones en la percepción del modelo de IA. A pesar de estos errores, su incidencia es mínima, por lo que el sistema se considera exitoso en el cumplimiento de sus objetivos.

	A	B	C	D	E
1	Clase	Confianza	Latitud	Longitud	Fecha y Hora
2	cardboard	90	-33,034069	-71,591958	2025-02-25 15:07:46
3	cardboard	90	-33,034069	-71,591956	2025-02-25 15:07:49
4	cardboard	91	-33,034068	-71,591956	2025-02-25 15:07:58
5	cardboard	57	-33,034069	-71,591957	2025-02-25 15:08:11
6	cardboard	63	-33,034069	-71,591959	2025-02-25 15:08:32
7	metal	87	-33,033823	-71,591985	2025-02-25 15:09:22
8	metal	83	-33,033826	-71,591978	2025-02-25 15:09:25
9	metal	89	-33,033826	-71,591979	2025-02-25 15:10:34
10	metal	63	-33,033825	-71,591979	2025-02-25 15:11:42
11	metal	91	-33,033825	-71,59198	2025-02-25 15:11:53
12	metal	79	-33,033813	-71,591977	2025-02-25 15:12:08
13	cardboard	51	-33,033792	-71,591981	2025-02-25 15:14:13
14	plastic	94	-33,033791	-71,59197	2025-02-25 15:14:16
15	plastic	83	-33,033791	-71,591967	2025-02-25 15:15:27
16	metal	56	-33,033785	-71,591968	2025-02-25 15:15:30
17	plastic	90	-33,033784	-71,591968	2025-02-25 15:20:51
18	plastic	94	-33,033781	-71,591966	2025-02-25 15:20:57
19					
20					

Figura 5.12: Datos guardados en Excel

5.3. Evaluación de Requisitos Funcionales

RF-01: Captura de Imágenes en Tiempo Real

Se logró capturar imágenes en tiempo real desde la cámara del celular montado en el Go1 utilizando Iriun Webcam.. **Requisito cumplido.**

RF-02: Análisis de Imágenes con YOLOv11

YOLOv11 detecta correctamente las clases entrenadas con precisión aceptable. Se observaron buenos valores de mAP en las métricas de evaluación.**Requisito cumplido.**

RF-03: Solicitud de GPS a MATLAB Mobile

Se reciben coordenadas GPS, pero hubo errores de conexión en algunos intentos, lo que generó alertas en el sistema. Se requiere robustecer la comunicación con MATLAB Mobile.

RF-04: Almacenamiento de Detecciones en Excel

Los datos de detección son registrados en un archivo Excel con los campos requeridos. **Requisito cumplido.**

RF-05: Prevención de Duplicados en Registros

Se implementó un filtro que impide registrar múltiples detecciones en la misma ubicación dentro del intervalo especificado. **Requisito cumplido.**

RF-06: Visualización en la Interfaz Gráfica

La interfaz gráfica muestra el video en tiempo real con etiquetas visibles y resaltadas. **Requisito cumplido.**

RF-07: Control Manual del Go1

Se verificó el control manual del Go1 por parte del operador mediante el mando inalámbrico. **Requisito cumplido.**

RF-08: Exportación de Datos

Se permite la exportación de los datos de detección en formatos CSV y SQL. **Requisito cumplido.**

RF-09: Notificaciones sobre MATLAB Mobile

Se implementó un mecanismo de alerta cuando MATLAB Mobile no está disponible. **Requisito cumplido.**

RF-10: Eliminación Automática de Registros Duplicados

Se implementó la eliminación de registros duplicados en base a latitud y longitud. **Requisito cumplido.**

RF-11: Backup Automático del Archivo Excel

Se genera un respaldo automático cuando el archivo Excel está bloqueado. **Requisito cumplido.**

5.4. Evaluación de Requisitos No Funcionales

RNF-01: Precisión en la Detección de Residuos

La detección de residuos con YOLOv11 alcanzó una precisión superior al 85 %. Los valores de mAP@0.5 y mAP@0.5-0.95 indican un desempeño adecuado en la identificación de residuos de cardboard/paper, metal, plastic. **Requisito cumplido.**

RNF-02: Velocidad de Actualización de la Interfaz

La interfaz gráfica se actualiza en tiempo real, pero en sistemas sin GPU o con hardware limitado puede haber retrasos perceptibles en la detección. Se recomienda optimización en hardware de menor capacidad. **Requisito parcialmente cumplido.**

RNF-03: Mecanismo de Recuperación ante Fallos

Se implementó un sistema de manejo de errores que permite la recuperación automática en caso de fallos en MATLAB Mobile o en la lectura de imágenes. **Requisito cumplido.**

RNF-04: Compatibilidad del Software

El sistema ha sido probado y ejecutado correctamente en entornos con Python 3.8+, OpenCV, YOLOv11 y MATLAB Mobile, cumpliendo con los requerimientos de compatibilidad. **Requisito cumplido.**

RNF-05: Manejo de Concurrencia en el Archivo Excel

Se implementaron mecanismos para evitar bloqueos en el archivo Excel y garantizar la integridad de los datos en escenarios de acceso simultáneo. **Requisito cumplido.**

RNF-06: Requisitos de Hardware

El sistema funciona de manera óptima en equipos con GPU NVIDIA con soporte CUDA y al menos 8GB de RAM. Sin embargo, en sistemas sin GPU dedicada, el rendimiento se ve afectado. **Requisito parcialmente cumplido.**

RNF-07: Gestión de Errores en MATLAB Mobile

Se implementó una espera de 10 segundos antes de reintentar la solicitud de GPS cuando MATLAB Mobile no responde. **Requisito cumplido.**

RNF-08: Operación en Ambientes Exteriores

El sistema funciona en entornos exteriores con conexión Wi-Fi disponible, asegurando la transmisión de datos y detecciones en tiempo real. **Requisito cumplido.**

Capítulo 6

Conclusiones

El presente proyecto ha demostrado la viabilidad de un sistema de detección de residuos basado en visión artificial, geolocalización y robótica cuadrúpeda, con el objetivo de mejorar la gestión de residuos en entornos urbanos mediante el uso del robot Go1 de Unitree. La integración de tecnologías como YOLOv11 para la detección de objetos, MATLAB Mobile para la obtención de coordenadas GPS e Iriun Webcam para la captura de imágenes en tiempo real ha permitido desarrollar una solución innovadora capaz de identificar y registrar residuos en un entorno dinámico.

El sistema ha logrado combinar el procesamiento de imágenes en tiempo real con una interfaz intuitiva, permitiendo al operador visualizar y registrar detecciones con precisión. Se han implementado mecanismos de almacenamiento estructurado en Excel. Asimismo, la funcionalidad de eliminación de registros duplicados ha optimizado la calidad de la información recopilada, asegurando una mayor confiabilidad en la geolocalización de residuos.

Con ajustes y mejoras en la optimización del procesamiento y la comunicación, el sistema podría evolucionar hacia una solución completamente autónoma, ampliando su aplicabilidad en entornos industriales, municipales y ambientales. A través de su impacto positivo en la gestión de residuos y el monitoreo ambiental, este sistema representa una herramienta valiosa en la lucha contra la contaminación y en la promo-

ción de ciudades más limpias y sostenibles.

Bibliografía

- [1] World Urbanization Prospects - Chile

<https://population.un.org/wup/Country-Profiles/>

- [2] Convivir con basura: el futuro que no queremos - Banco Mundial

<https://www.bancomundial.org/es/news/feature/2019/03/06/convivir-con-basura-el-futuro-que-no-queremos>

- [3] Waste and human health - OMS

<https://iris.who.int/handle/10665/354227>

- [4] Solid waste management - UNEP

<https://www.unep.org/explore-topics/resource-efficiency/what-we-do/cities/solid-waste-management>

- [5] Solid waste management challenges for cities in developing countries-Lilliana Abarca-Guerrero

https://www.scielo.sa.cr/scielo.php?pid=S0379-39822015000200141&script=sci_arttext

- [6] Urban solid residues, garbage collectors and public health

<https://www.proquest.com/docview/1679438158?pq-origsite=gscholar&fromopenview=true&sourcetype=Scholarly%20Journals>

[7] Residuos-Sinia-Ministerio del Medio Ambiente

https://sinia.mma.gob.cl/wp-content/uploads/2022/01/C10-residuos-rema-2021.pdf?utm_source=chatgpt.com

[8] Proyección de la Generación, Reciclaje y Fracción Resto de los Residuos Sólidos Domiciliarios y Asimilables en Chile entre 2020 y 2040

https://consultasciudadanas.mma.gob.cl/storage/citizen/6301/Estudio%20Proyeccin%20Generacin%20RSM%20en%20Chile%202020-2040.pdf?utm_source=chatgpt.com

[9] Aplicaciones de la inteligencia Artificial en la gestión de residuos urbanos

<https://hablandoenvidrio.com/inteligencia-artificial-mejor-gestion>

[10] Casos de uso IoT en la gestión de residuos y reciclaje <https://secmotic.com/casos-uso-iot-gestion-residuos/#:~:text=La%20tecnologa%20IoT%2C%20puede%20ayudar,de%20recuperacin%20de%20materias%20valiosas.>

<https://secmotic.com/casos-uso-iot-gestion-residuos/#:~:text=La%20tecnologa%20IoT%2C%20puede%20ayudar,de%20recuperacin%20de%20materias%20valiosas.>

[11] Using Computer Vision to Clean Up San Francisco Streets

<https://blog.roboflow.com/computer-vision-clean-city-litter-trash/>

[12] Alcorcón pone en marcha su unidad de drones contra los vertidos ilegales

<https://www.ayto-alcorcon.es/es/comunicacion/alcorcon-pone-en-marcha-su-unidad-de-drones-contralos-vertidos-ilegales>

[13] Barcelona sostenible: la revolución de la inteligencia artificial en la gestión ambiental

<https://barcelona.impacthub.net/la-revolucion-de-la-inteligencia-artificial-en-la-gestion-ambiental>

[14] A Deep Dive into Accurate Garbage Classification Using Transfer Learning Techniques for Environmental Sustainability

<https://ieeexplore.ieee.org/document/10441003>

- [15] Techniques and Models for Addressing Occupational Risk Using Fuzzy Logic, Neural Networks, Machine Learning, and Genetic Algorithms: A Review and Meta-Analysis

<https://www.mdpi.com/2076-3417/15/4/1909>

- [16] Artificial intelligence for waste management in smart cities

[https://link.springer.com/article/10.1007/](https://link.springer.com/article/10.1007/s10311-023-01604-3)

[s10311-023-01604-3](https://link.springer.com/article/10.1007/s10311-023-01604-3)

- [17] ¿Qué es la visión artificial?

<https://www.ibm.com/es-es/topics/computer-vision>

- [18] LA VISIÓN ARTIFICIAL Y LOS CAMPOS DE APLICACIÓN ARTIFICIAL

https://www.researchgate.net/publication/360024177_

[LA_VISION_ARTIFICIAL_Y_LOS_CAMPOS_DE_APLICACION_](https://www.researchgate.net/publication/360024177_)

[ARTIFICIAL](https://www.researchgate.net/publication/360024177_)

- [19] Ultralytics YOLO11

<https://docs.ultralytics.com/models/yolo11/>

- [20] A Comprehensive Guide to YOLOv11 Object Detection

<https://www.analyticsvidhya.com/blog/2024/10/>

[yolov11-object-detection/](https://www.analyticsvidhya.com/blog/2024/10/)

- [21] Iriun Webcam: A Comprehensive User Guide

<https://iriun.net/iriun-webcam/>

- [22] Roboflow:Trashnet

<https://universe.roboflow.com/>

[polygence-project/trashnet-a-set-of-annotated-images-of-trash-that-dataset/20](https://universe.roboflow.com/)

- [23] trash mega Computer Vision Project

<https://universe.roboflow.com/studio-boson/>

[trash-mega/dataset/1](https://universe.roboflow.com/studio-boson/)

[24] Adquisición de Datos GPS y Representación en un Mapa

`la.mathworks.com/help/matlabmobile/ug/
acquire-gps-data-and-plot-your-location-and-speed-on-a-map.
html`

[25] Recopilación de datos de sensores con MATLAB Mobile

`https://la.mathworks.com/help/matlabmobile/ug/
sensor-data-collection-with-matlab-mobile.html?utm_
source=chatgpt.com`

[26] Página oficial de Unitree Robotics sobre el Go1

`https://www.unitree.com/go1`

[27] Manual Go1 de Unitree Robotics

`https://botland.store/img/art/inne/Go1_User_Manual_V1.
4_202112.pdf`