Sistema de Monitoreo y Alerta Temprana basado en Inteligencia Artificial para Áreas Protegidas

Autores:
Fabrizio Martin Contigiani
Gabriel Orlando Da Silva Schmies

Tutor:
Dr. Ing. Sergio Eduardo Moya

31 de diciembre de 2025

Resumen

El presente trabajo describe el diseño e implementación de un sistema de monitoreo y alerta temprana para áreas protegidas de la Selva Misionera, que combina tecnologías de Internet de las Cosas (IoT) con Inteligencia Artificial (IA) para la detección automática de fauna silvestre, personas y vehículos.

El sistema está compuesto por una red de nodos de captura basados en microcontroladores ESP32 equipados con cámaras, los cuales se comunican mediante el protocolo ESP-MESH para transmitir imágenes hacia un nodo raíz. Este nodo actúa como puerta de enlace, reenviando las imágenes a un servidor remoto a través de Transmission Control Protocol (TCP)/IP.

En el servidor, las imágenes son procesadas por un servicio de inferencia basado en SpeciesNet, un modelo de detección de objetos desarrollado por Google que utiliza YOLOv5. Este modelo permite identificar y clasificar especies de fauna silvestre, así como detectar la presencia de humanos y vehículos, generando alertas automáticas ante posibles intrusiones.

La arquitectura del servidor incluye una aplicación web desarrollada en Django para la gestión de imágenes, un bot de Telegram para el envío de notificaciones en tiempo real, y una base de datos PostgreSQL para el almacenamiento persistente. Todo el sistema está contenedorizado mediante Docker para facilitar su despliegue.

Los resultados demuestran la viabilidad de implementar un sistema de vigilancia inteligente de bajo costo para áreas protegidas, capaz de operar de manera autónoma y alertar a los administradores ante eventos relevantes.

Palabras Clave - Cámaras Trampa, Internet de las Cosas, Inteligencia Artificial, Monitoreo de Fauna, Detección de Intrusos, ESP-MESH, SpeciesNet, Selva Misionera

Agradecimientos

Queremos expresar nuestro sincero agradecimiento a todas las personas e instituciones que hicieron posible la realización de este trabajo.

En primer lugar, a nuestro tutor, el Dr. Ing. Sergio Eduardo Moya, por su guía, dedicación y apoyo constante a lo largo del desarrollo de este proyecto.

A la Universidad Nacional de Misiones y a la Facultad de Ingeniería, por brindarnos la formación académica y los recursos necesarios para nuestra educación profesional.

A nuestras familias, por su apoyo incondicional, paciencia y comprensión durante todos estos años de estudio.

A nuestros compañeros y amigos, por compartir este camino y hacer más llevadero el proceso de aprendizaje.

Finalmente, a todos aquellos que de una u otra manera contribuyeron a la culminación de este trabajo.

Índice general

Índice de figuras

Índice de tablas

Índice de códigos fuente

Glosario

alerta temprana Sistema de detección y notificación que permite identificar y comunicar eventos críticos con mínima latencia, posibilitando respuestas preventivas antes de que el daño se materialice. 24

AP-STA Modo de operación en el que un dispositivo actúa simultáneamente como Punto de Acceso (AP) y Estación (STA). 36, 51

aprendizaje automático Rama de la inteligencia artificial que permite a los sistemas aprender y mejorar automáticamente a partir de datos sin ser programados explícitamente, también conocido como machine learning. 18, 19, 37

autocuración Capacidad de una red mesh para reconfigurarse automáticamente cuando un nodo falla, permitiendo que los nodos afectados encuentren rutas alternativas sin intervención manual. 36, 51

batería 18650 Celda de batería de iones de litio con formato cilíndrico de 18 mm de diámetro y 65 mm de longitud. Ampliamente utilizada en dispositivos electrónicos portátiles por su alta densidad energética y capacidad de recarga. 29, 41, 48, 53, 55, 66, 98

Bosque Atlántico Ecorregión de bosques subtropicales húmedos que se extiende por Brasil, Paraguay y Argentina, conocida también como Mata Atlántica, uno de los biomas más biodiversos del planeta. 22, 26

bot de Telegram Programa automatizado que interactúa con usuarios a través de la plataforma de mensajería Telegram. 1, 16, 29, 45, 47, 56, 96

bounding box Rectángulo que delimita la ubicación de un objeto detectado en una imagen. 37, 52, 53, 58

CAD Diseño Asistido por Computadora (Computer-Aided Design), software utilizado para crear modelos 2D y 3D de objetos. 41

callback Función que se pasa como argumento a otra función y que será invocada cuando ocurra un evento específico, permitiendo la programación asíncrona y orientada a eventos. 37

cámara trampa Dispositivo de captura de imágenes activado por movimiento, utilizado comúnmente para monitoreo de fauna silvestre. 16, 18, 19, 20, 25, 27, 28, 30, 31, 35, 37, 38, 64, 65

carcasa Estructura protectora que alberga los componentes electrónicos de un dispositivo, protegiéndolos del ambiente externo. 41, 43

caza furtiva Práctica ilegal de captura o matanza de animales silvestres sin autorización, a menudo de especies protegidas o en peligro de extinción. 16, 17, 21, 23, 24, 26, 27, 34

clasificación taxonómica Proceso de identificar y categorizar organismos biológicos en grupos jerárquicos como familia, género y especie. 18, 20, 28, 38, 58

código G Lenguaje de programación utilizado para controlar máquinas de fabricación por control numérico, incluyendo impresoras 3D. Define la trayectoria de la herramienta, velocidades y temperaturas. 43

código abierto Software cuyo código fuente está disponible públicamente para que cualquiera pueda estudiarlo, modificarlo y distribuirlo. 17, 19, 20, 27, 38

convertidor buck Circuito regulador de voltaje de tipo reductor (step-down) que convierte un voltaje de entrada mayor a un voltaje de salida menor con alta eficiencia. Utilizado para alimentar dispositivos electrónicos desde baterías de mayor voltaje. 29, 41, 48, 49, 53, 66, 98

Corredor Verde Corredor biológico de aproximadamente 1,1 millones de hectáreas en la provincia de Misiones, Argentina, que conecta áreas protegidas para permitir el desplazamiento de fauna silvestre y mantener la conectividad genética de las poblaciones. 22, 23, 26

deep sleep Modo de bajo consumo extremo en microcontroladores donde la mayoría de los periféricos y el procesador principal se apagan, manteniendo solo un núcleo mínimo capaz de despertar el sistema ante eventos externos como señales de sensores o temporizadores. 39, 54, 65

despliegue continuo Práctica de ingeniería de software que automatiza el proceso de publicación de cambios de código a producción, permitiendo entregas frecuentes y confiables. También conocido como Continuous Deployment (CD). 44

detección de objetos Técnica de visión por computadora que identifica y localiza objetos dentro de una imagen, generalmente mediante bounding boxes. 16, 18, 37

Django Framework de desarrollo web de alto nivel escrito en Python, que sigue el patrón modelo-vista-plantilla. 1, 35, 39, 40, 42, 43, 52, 53

Docker Plataforma de contenedorización que permite empaquetar aplicaciones junto con sus dependencias para facilitar el despliegue. 1, 39, 43, 44, 51, 52, 56, 92

Docker Compose Herramienta para definir y ejecutar aplicaciones Docker multi-contenedor mediante archivos YAML. 39, 43, 53

edge computing Paradigma de computación distribuida que acerca el procesamiento y el almacenamiento de datos a la ubicación donde se necesitan, con el fin de mejorar los tiempos de respuesta y ahorrar ancho de banda, también conocido como computación en el borde. 19, 21

ESP-IDF Espressif IoT Development Framework, entorno oficial de desarrollo para microcontroladores ESP32. 39, 43, 48, 56

ESP-MDF Espressif Mesh Development Framework, framework para desarrollo de redes mesh sobre ESP-IDF. 30, 37, 39, 43, 48, 49, 54, 56

ESP-MESH Protocolo de red mesh desarrollado por Espressif para microcontroladores ESP32, basado en Wi-Fi. 1, 16, 27, 28, 30, 35, 36, 37, 47, 48, 50, 51, 64

ESP32 Microcontrolador de bajo costo y bajo consumo con Wi-Fi y Bluetooth integrados, fabricado por Espressif Systems. 1, 16, 19, 21, 27, 28, 29, 35, 36, 39, 42, 43, 48, 64

firmware Software de bajo nivel almacenado en la memoria de un dispositivo que controla su funcionamiento básico. 39, 42, 44, 45

fragmentación del hábitat Proceso por el cual un ecosistema continuo se divide en parches aislados, reduciendo la conectividad entre poblaciones de especies y afectando su viabilidad genética y capacidad de dispersión. 24

Git Sistema de control de versiones distribuido de código abierto, diseñado para gestionar proyectos de software de cualquier tamaño con velocidad y eficiencia. 44

GitHub Actions Plataforma de integración y despliegue continuo (CI/CD) integrada en GitHub que permite automatizar flujos de trabajo de desarrollo, pruebas y despliegue. 44

grado de protección IP Código estándar definido por la norma IEC 60529 que indica el nivel de protección de un dispositivo contra la intrusión de sólidos (primer dígito) y líquidos (segundo dígito). Por ejemplo, IP67 indica protección total contra polvo y contra inmersión temporal en agua. 29

hotspot de biodiversidad Región biogeográfica que alberga una concentración excepcional de especies endémicas y que ha experimentado una pérdida significativa de hábitat, concepto introducido por Norman Myers en 1988. 22

impresión 3D Proceso de fabricación aditiva que crea objetos tridimensionales mediante la deposición de material capa por capa. 41

inferencia Proceso de utilizar un modelo de aprendizaje automático entrenado para realizar predicciones sobre nuevos datos. 1, 9, 19, 85, 86, 88

Kanban Metodología ágil de gestión de proyectos que visualiza el flujo de trabajo mediante un tablero con columnas, limita el trabajo en progreso y promueve la entrega continua de valor. 42

LoRa Tecnología de comunicación inalámbrica de largo alcance (Long Range) y bajo consumo de energía, que utiliza modulación por espectro ensanchado. 19, 21

LoRaWAN Protocolo de red que utiliza la tecnología LoRa para conectar dispositivos a internet, optimizado para bajo consumo y comunicaciones de largo alcance. 19

Mata Atlántica Nombre en portugués del Bosque Atlántico, utilizado principalmente en Brasil donde se encuentra el 92 % de la extensión original de este bioma. 22

MegaDetector Modelo de detección de objetos desarrollado por Microsoft para identificar automáticamente animales, personas y vehículos en imágenes de cámaras trampa. 20, 27, 28, 38, 47, 51, 52, 64

mesh Topología de red donde cada nodo puede conectarse con múltiples nodos vecinos, permitiendo rutas alternativas para la transmisión de datos. 17, 18, 19, 21, 27, 28, 29, 30, 36, 40, 42, 44, 45, 47, 49, 50, 57, 64

Monumento Natural Categoría de protección legal en Argentina que confiere a una especie o área natural el máximo nivel de protección, prohibiendo cualquier forma de caza, captura o comercio. 22

Mwifi Componente del ESP-MDF que proporciona APIs de alto nivel para comunicación en redes ESP-MESH. 37, 39, 48, 49, 51

nodo Dispositivo individual que forma parte de una red mesh. 1, 36

nodo raíz Nodo principal de una red mesh que actúa como puerta de enlace hacia redes externas. 1, 36

PostgreSQL Sistema de gestión de bases de datos relacional de código abierto. 1, 39, 53, 56

power save Modo de ahorro de energía en redes ESP-MESH donde los nodos reducen su consumo limitando el tiempo de transmisión Wi-Fi activa, manteniendo el CPU funcionando para procesar eventos. 48, 53, 54, 62

PyTorch Biblioteca de aprendizaje automático de código abierto basada en la biblioteca Torch, utilizada para aplicaciones como visión computacional y procesamiento de lenguaje natural. 20, 37

reserva de biosfera Área protegida reconocida por la UNESCO bajo el programa El Hombre y la Biosfera (MAB), que integra conservación de la biodiversidad con uso sostenible de recursos naturales. 24

selva paranaense Nombre alternativo para la Selva Misionera, que forma parte del bioma del Bosque Atlántico y se extiende por la cuenca del río Paraná en Argentina, Paraguay y Brasil. 22

Selva Misionera Porción argentina del Bosque Atlántico, ubicada principalmente en la Provincia de Misiones, caracterizada por su alta biodiversidad y vegetación subtropical. 1, 16, 21, 22, 23, 24, 26, 30

sensor PIR Sensor de Infrarrojo Pasivo que detecta cambios en la radiación infrarroja emitida por objetos en movimiento, comúnmente utilizado para detectar presencia de animales o personas. 25, 40

servicios ecosistémicos Beneficios que los ecosistemas proporcionan a la sociedad humana, incluyendo regulación climática, provisión de agua limpia, control de erosión, polinización y secuestro de carbono. 23

SpeciesNet Modelo de detección y clasificación de fauna silvestre desarrollado por Google, basado en YOLOv5. 1, 16, 18, 20, 27, 28, 35, 38, 39, 42, 47, 51, 52, 53, 56, 64, 85, 86, 88, 93, 94, 95

tiempo real Procesamiento de datos que ocurre con latencia mínima, permitiendo respuestas inmediatas a eventos. 16, 17, 18, 19, 21, 25, 26, 30

tráfico de fauna Comercio ilegal de animales silvestres vivos o sus partes, considerado la cuarta actividad ilícita más lucrativa a nivel mundial y una de las principales causas de pérdida de biodiversidad. 24

visión por computadora Campo de la inteligencia artificial que permite a las computadoras interpretar y comprender información visual del mundo real. 17, 18, 37

YOLOv5 Versión 5 del modelo You Only Look Once, arquitectura de red neuronal para detección de objetos en tiempo real. 1, 18, 19, 37, 51

Siglas

API Application Programming Interface. 51

ASA Acrilonitrilo Estireno Acrilato. 41, 66

CMOS Complementary Metal-Oxide-Semiconductor. 31, 32

CNN Red Neuronal Convolucional. 18, 37

EAP Establecimiento Agropecuario Productivo. 97

FDM Modelado por Deposición Fundida. 41

FreeRTOS Free Real-Time Operating System. 39

IA Inteligencia Artificial. 1, 16, 17, 18, 19, 20, 21, 25, 26, 27, 28, 29, 30, 35, 38, 40, 45, 47, 64, 65, 96

IoT Internet de las Cosas. 1, 16, 17, 35

IR Infrarrojo. 25, 30, 32, 40

JPEG Joint Photographic Experts Group. 32, 40, 47, 48, 51, 52

OTA Over-The-Air. 39

PETG Tereftalato de Polietileno Glicol. 41, 66

PIR Infrarrojo Pasivo. 25, 28, 29, 30, 31, 32, 34, 35, 42, 45, 47, 48, 64, 65, 66

PLA Ácido Poliláctico. 41, 43, 49, 98, 99

RSSI Received Signal Strength Indicator. 36, 51

SoC System on Chip. 39

TCP Transmission Control Protocol. 1, 37

YOLO You Only Look Once. 18, 19, 37

Capítulo 1

Introducción

La conservación de la fauna silvestre y la protección de áreas naturales representan desafíos críticos en la actualidad. En regiones de alta biodiversidad como la Selva Misionera, la pérdida de hábitat, la caza furtiva y la intrusión humana en ecosistemas protegidos amenazan el equilibrio ecológico y la supervivencia de numerosas especies. Ante esta problemática, surge la necesidad de implementar sistemas de monitoreo que permitan vigilar estas áreas de manera continua y eficiente.

Tradicionalmente, el monitoreo de fauna silvestre se ha realizado mediante cámara trampas, dispositivos que capturan imágenes cuando detectan movimiento. Sin embargo, estos sistemas convencionales presentan limitaciones significativas: requieren revisión manual periódica, no permiten alertas en tiempo real, y generan grandes volúmenes de datos que deben ser analizados manualmente por expertos.

El avance de tecnologías como el IoT y la IA ofrece nuevas posibilidades para superar estas limitaciones. La combinación de redes de sensores inalámbricos con algoritmos de detección de objetos permite desarrollar sistemas capaces de identificar automáticamente fauna silvestre, personas y vehículos, generando alertas inmediatas ante eventos relevantes.

El presente trabajo, desarrollado en el marco de la Universidad Nacional de Misiones, propone el diseño e implementación de un sistema de monitoreo y alerta temprana para áreas protegidas de la región, que integra una red de nodos de captura basados en ESP32 comunicados mediante ESP-MESH, un servidor de procesamiento con IA basado en SpeciesNet, y un sistema de notificaciones a través de bot de Telegram.

1.1. Contexto y motivación

Las áreas protegidas enfrentan amenazas constantes que van desde la caza ilegal de especies en peligro hasta la intrusión de personas no autorizadas y vehículos en zonas restringidas. Los guardaparques y administradores de estas áreas frecuentemente carecen de los recursos humanos y tecnológicos necesarios para mantener una vigilancia efectiva sobre extensas superficies de terreno, muchas veces en ubicaciones remotas con acceso limitado a infraestructura de comunicaciones.

Las cámaras trampa tradicionales, aunque útiles para la investigación científica, no fueron diseñadas para la vigilancia en tiempo real. Las imágenes capturadas permanecen almacenadas en tarjetas de memoria que deben ser recolectadas físicamente, lo que implica visitas frecuentes al campo y retrasos significativos entre la captura de un evento y su descubrimiento. Para

cuando se detecta una intrusión o un acto de caza furtiva, los responsables ya se encuentran lejos del área.

Es cierto que en la actualidad existen cámaras trampa más modernas que incorporan conectividad celular o satelital, permitiendo el envío remoto de imágenes. Sin embargo, estos dispositivos suelen tener un costo significativamente mayor, lo que limita su adopción masiva especialmente en países en vías de desarrollo, donde paradójicamente se concentra gran parte de la biodiversidad mundial. Además, muchas de estas soluciones comerciales dependen de servicios en la nube propietarios con costos de suscripción recurrentes.

La motivación de este proyecto surge de la necesidad de transformar el paradigma del monitoreo pasivo hacia un sistema activo de vigilancia inteligente, pero de manera accesible y de bajo costo. Un sistema que no solo capture imágenes, sino que las transmita en tiempo real, las analice automáticamente mediante algoritmos de IA, y genere alertas inmediatas cuando se detecten eventos de interés, ya sea la presencia de fauna silvestre para fines de investigación, o la detección de intrusos para fines de seguridad. Todo esto utilizando componentes de hardware económicos y software de código abierto.

1.2. Estructura del documento

El presente documento se organiza en nueve capítulos que describen de manera progresiva el desarrollo del sistema propuesto:

Capítulo 1 - Introducción: Presenta el contexto general del proyecto, la motivación y la estructura del documento.

Capítulo 2 - Antecedentes: Revisa trabajos relacionados, soluciones comerciales existentes y el estado del arte en sistemas de monitoreo de fauna y detección de intrusos.

Capítulo 3 - Planteamiento del Problema: Describe la problemática de las áreas protegidas, las limitaciones de los sistemas tradicionales y la justificación del proyecto.

Capítulo 4 - Objetivos y Alcance: Define los objetivos generales y específicos, así como el alcance y las limitaciones del trabajo.

Capítulo 5 - Marco Teórico: Presenta los fundamentos teóricos sobre IoT, redes mesh, IA aplicada a visión por computadora, y las tecnologías utilizadas.

Capítulo 6 - Metodología: Describe el enfoque metodológico, las etapas de desarrollo, las herramientas empleadas y las métricas de evaluación.

Capítulo 7 - Diseño e Implementación: Detalla la arquitectura del sistema, el diseño e implementación de cada componente: nodos de cámara, nodo raíz, red mesh, servicio de detección y servidor de aplicación.

Capítulo 8 - Pruebas y Resultados: Documenta las pruebas realizadas y analiza los resultados obtenidos en conectividad, detección, rendimiento y consumo energético.

Capítulo 9 - Conclusiones: Resume las conclusiones generales, los aportes del trabajo, trabajos futuros y recomendaciones.

Capítulo 2

Antecedentes

En los últimos años, el campo de la IA aplicada a la visión por computadora ha experimentado avances significativos. El desarrollo de arquitecturas de Redes Neuronales Convolucionales (CNNs) cada vez más eficientes, junto con la disponibilidad de grandes conjuntos de datos de entrenamiento, ha permitido crear modelos capaces de detectar y clasificar objetos en imágenes con una precisión sin precedentes [1]. Algoritmos como You Only Look Once (YOLO) [2] y sus sucesivas versiones, incluyendo YOLOv5 [3], han revolucionado la detección de objetos, permitiendo el procesamiento en tiempo real incluso en dispositivos con recursos limitados.

En el ámbito específico del monitoreo de fauna silvestre, estos avances han dado lugar a modelos especializados como SpeciesNet, desarrollado por Google [4], que combina detección de objetos con clasificación taxonómica para identificar especies a partir de imágenes de cámara trampas. Estos desarrollos han abierto nuevas posibilidades para automatizar el análisis de las enormes cantidades de imágenes que generan los sistemas de monitoreo.

Paralelamente, la tecnología de cámara trampas ha evolucionado desde dispositivos autónomos que almacenan imágenes localmente, hasta sistemas más sofisticados con conectividad celular o satelital que permiten la transmisión remota de datos. Este capítulo examina tanto los avances en IA aplicada a la visión por computadora, como la evolución de las soluciones de monitoreo de fauna, incluyendo sistemas comerciales y proyectos de investigación relacionados.

2.1. Trabajos relacionados

En el ámbito local, Barrero y Schmunck [5] desarrollaron en la Universidad Nacional de Misiones una microcámara de vigilancia orientada a la protección de fauna salvaje. Este trabajo propuso el diseño de un dispositivo compacto basado en microcontroladores capaz de capturar imágenes en áreas naturales. El proyecto sentó las bases para el desarrollo de soluciones de bajo costo adaptadas a las necesidades específicas de la región misionera, demostrando la viabilidad de utilizar hardware económico para aplicaciones de monitoreo ambiental. El presente trabajo extiende estos conceptos incorporando comunicación en red mesh, procesamiento con IA y un sistema de alertas en tiempo real.

En el contexto regional argentino, González et al. [6] presentaron en el Workshop de Investigadores en Ciencias de la Computación (WICC 2022) un sistema de IA para la multiclasificación de fauna en fotografías automáticas utilizadas en investigación científica. Este trabajo demuestra el creciente interés en la comunidad científica argentina por aplicar técnicas de aprendizaje automático al análisis de imágenes de cámara trampas, estableciendo un

precedente importante para proyectos de monitoreo de fauna en el país.

En el contexto internacional, diversos trabajos han explorado el uso de YOLOv5 para el análisis automatizado de imágenes de cámara trampas. Abood et al. [7] presentaron un enfoque innovador en la conferencia IEEE ICMNWC 2023, demostrando que los modelos de la familia YOLO pueden adaptarse eficazmente para la detección y clasificación de fauna silvestre. Los autores destacaron las ventajas de YOLOv5 en términos de velocidad de inferencia y precisión. De manera similar, Njathi et al. [8] propusieron en IEEE AFRICON 2023 un sistema eficiente de anotación de imágenes de cámaras trampa basado en YOLOv5, enfocándose en reducir el tiempo y esfuerzo requerido para etiquetar grandes conjuntos de datos. Ambos trabajos evidencian la idoneidad de los modelos YOLO para aplicaciones de monitoreo de fauna que requieren procesamiento eficiente de grandes volúmenes de imágenes.

Un antecedente particularmente relevante para el presente trabajo es el sistema propuesto por Whytock et al. [9], quienes desarrollaron cámaras trampa con IA capaces de enviar alertas en tiempo real a través de la red satelital Iridium. El estudio, realizado en Gabón (África Central), demostró la viabilidad de integrar procesamiento con IA directamente en dispositivos de campo para detectar fauna y enviar notificaciones inmediatas a investigadores y guardaparques. Aunque su solución utiliza conectividad satelital —con costos operativos significativos—, el concepto de alertas en tiempo real basadas en detección automática constituye un pilar fundamental del sistema propuesto en este trabajo. La diferencia principal radica en que nuestra solución emplea redes mesh locales y conectividad Wi-Fi/TCP, reduciendo considerablemente los costos de operación.

Otro proyecto directamente comparable es AiCatcher, desarrollado por Mallya [10], que propone una cámara trampa inteligente capaz de realizar inferencia directamente en el dispositivo de campo. AiCatcher utiliza una Raspberry Pi como unidad de procesamiento para ejecutar modelos de detección en el borde (edge computing), y emplea módulos LoRa (Adafruit LoRa Bonnets) configurados en una red LoRaWAN para la transmisión de datos, convirtiendo las señales recibidas en mensajes SMS mediante Twilio. Si bien este enfoque es innovador, el uso de Raspberry Pi presenta un consumo energético significativamente mayor, limitando la autonomía del dispositivo en campo. En contraste, nuestra solución emplea microcontroladores ESP32 de bajo consumo para la captura y transmisión, delegando el procesamiento de IA a un servidor remoto. Además, el uso de redes mesh en nuestra propuesta permite una topología de red más flexible y autoorganizada.

2.2. Estado del arte

Vélez et al. [11] realizaron una evaluación exhaustiva de las plataformas disponibles para el procesamiento de imágenes de cámara trampas utilizando IA. Su estudio, publicado en *Methods in Ecology and Evolution*, comparó múltiples herramientas en términos de precisión, facilidad de uso y requisitos computacionales. Los autores concluyeron que, si bien existen diversas opciones de código abierto y comerciales, la elección de la plataforma adecuada depende del contexto específico de cada proyecto, incluyendo el volumen de imágenes, las especies objetivo y los recursos disponibles. Este análisis proporciona un marco de referencia valioso para la selección de tecnologías en proyectos de monitoreo de fauna.

Tabak et al. [12] demostraron la aplicabilidad del aprendizaje automático para clasificar especies animales en imágenes de cámaras trampa, alcanzando precisiones superiores al 90 % en la identificación de especies comunes. Su trabajo destacó la importancia de contar con

conjuntos de datos de entrenamiento representativos de las condiciones locales para optimizar el rendimiento de los modelos.

Por su parte, Steenweg et al. [13] abordaron el desafío de escalar las redes de cámara trampas para el monitoreo de biodiversidad a nivel global. Los autores argumentaron que la integración de sensores remotos en redes coordinadas, combinada con técnicas de análisis automatizado, representa el futuro del monitoreo de fauna silvestre, permitiendo obtener datos de biodiversidad a escalas espaciales y temporales sin precedentes.

Entre las plataformas más recientes, Hernández et al. [14] presentaron PyTorch-Wildlife, un framework colaborativo de aprendizaje profundo desarrollado por Microsoft AI for Good. Esta plataforma, similar en objetivos a SpeciesNet de Google, ofrece modelos preentrenados para la detección y clasificación de fauna silvestre en imágenes de cámara trampas. PyTorch-Wildlife se distingue por su enfoque en la facilidad de uso y la integración con el ecosistema PyTorch, facilitando tanto el despliegue de modelos existentes como el entrenamiento de modelos personalizados para especies específicas. La existencia de múltiples frameworks de código abierto respaldados por grandes empresas tecnológicas evidencia la relevancia del problema y la madurez de las soluciones disponibles.

Cabe destacar también MegaDetector, desarrollado originalmente por Microsoft AI for Earth [15], que se ha convertido en una herramienta fundamental en el procesamiento de imágenes de cámara trampas. A diferencia de los clasificadores de especies, MegaDetector se especializa en la detección genérica de animales, humanos y vehículos, funcionando como un primer filtro que reduce drásticamente el volumen de imágenes a analizar. Esta herramienta ha sido adoptada por más de 60 organizaciones de conservación a nivel mundial, demostrando reducciones de hasta el 90 % en el tiempo de procesamiento de datos. Su arquitectura modular permite integrarlo como etapa previa a clasificadores más específicos como SpeciesNet.

2.3. Soluciones comerciales existentes

En el mercado actual, existen diversas soluciones comerciales diseñadas para el monitoreo remoto de fauna. Las cámara trampas con conectividad celular, como las series REVEAL de Tactacam [16] y las líneas Flex de Spypoint [17], son las más extendidas. Estos dispositivos permiten capturar imágenes de alta resolución y enviarlas a través de redes LTE a una aplicación móvil propietaria. Algunas de sus características principales incluyen disparo rápido (menor a 0,5 s), visión nocturna por infrarrojos y, en modelos recientes, la capacidad de solicitar fotografías o videos bajo demanda.

Sin embargo, estas soluciones presentan limitaciones críticas para su adopción masiva en proyectos de conservación a gran escala o en regiones remotas. En primer lugar, dependen enteramente de la infraestructura de red celular; en áreas de selva densa como la de Misiones, la falta de cobertura inalámbrica en el interior de las reservas suele ser la norma, invalidando el uso de estos dispositivos para alertas remotas. En segundo lugar, el costo operativo es elevado, ya que además de que cada cámara en sí tiene un costo de adquisición significativo, cada dispositivo requiere un plan de datos independiente, con suscripciones mensuales que pueden oscilar entre los 5 USD y 15 USD por unidad [16], [17].

En cuanto a la gestión de datos, plataformas como Wildlife Insights [18] ofrecen una infraestructura robusta basada en la nube para el almacenamiento y análisis de imágenes mediante IA. Esta herramienta permite filtrar imágenes vacías automáticamente y realizar la clasificación taxonómica de especies, facilitando la colaboración entre investigadores. No obstante, Wildlife

Insights está orientada principalmente al procesamiento posterior (post-hoc) de los datos recolectados manualmente de las tarjetas SD, y no está diseñada para la vigilancia y respuesta inmediata ante eventos de intrusión o caza furtiva en tiempo real.

2.4. Análisis comparativo

A fin de situar las necesidades identificadas frente a las alternativas y trabajos discutidos anteriormente, se presenta a continuación un análisis comparativo que resume las principales diferencias técnicas y operativas. En la tabla 2.1 se contrastan las características de las categorías de soluciones analizadas: cámaras trampa tradicionales, soluciones comerciales celulares [16], [17], el sistema satelital de Whytock et al. [9] y el enfoque basado en edge computing de Mallya (AiCatcher) [10].

Tabla 2.1: Comparativa de soluciones de monitoreo y trabajos relacionados.

Característica	**Tradicional**	**Celular [16]**	**Satelital [9]**	**LoRa [10]**
Alertas en tiempo real	No	Sí	Sí	Sí
Infraestructura requerida	Ninguna	Operador Celular	Red Satelital	Puerta de enlace LoRa
Procesamiento de IA	Post-hoc	No / Limitado	Edge (Hardware Dedicado)	Edge (Raspberry Pi)
Costo de adquisición	Bajo-Medio	Medio-Alto	Muy Alto	Medio-Alto
Costo Operativo	Bajo	Alto (Plan mensual)	Muy Alto (Iridium)	Bajo (SMS/Twilio)
Autonomía energética	Muy Alta	Media	Media-Baja	Baja (Raspberry Pi)
Escalabilidad	Laboriosa	Por suscripción	Costo-prohibitiva	Media (LoRa)
Dependencia de terceros	No	Muy Alta	Muy Alta	Media (Twilio)

Como se desprende de la comparación, si bien existen soluciones de vanguardia, el acceso a alertas inmediatas suele estar condicionado por elevados costos operativos o dependencia de infraestructura de terceros (celular o satelital). Los trabajos de investigación como AiCatcher [10] y el sistema de Whytock et al. [9] demuestran la viabilidad de la IA para el filtrado en tiempo real, pero enfrentan desafíos en cuanto a consumo energético y costos de transmisión.

En este sentido, se identifica una oportunidad para un sistema que combine la eficiencia en el consumo de los microcontroladores ESP32 con la capacidad de extensión de cobertura de las redes mesh, permitiendo un monitoreo inteligente y de bajo costo operativo adaptado a las condiciones reales observadas en la Selva Misionera.

Capítulo 3

Planteamiento del Problema

3.1. Contexto regional

La provincia de Misiones, ubicada en el extremo noreste de Argentina, alberga uno de los ecosistemas más diversos y amenazados del continente [19]. El marco geográfico de este proyecto se sitúa en el corazón de la selva paranaense, una región que requiere estrategias de conservación urgentes y tecnificadas.

3.1.1. La Selva Misionera y el Bosque Atlántico

La Selva Misionera representa el remanente continuo más extenso del Bosque Atlántico —también conocido como Mata Atlántica— en el Cono Sur. Originalmente, este bioma cubría aproximadamente 1,3 millones de kilómetros cuadrados, extendiéndose desde la costa atlántica de Brasil (92 % de su extensión) hasta el este de Paraguay (6 %) y el noreste de Argentina (2 %) [20].

Sin embargo, debido a siglos de expansión agrícola, ganadera y forestal, hoy solo sobrevive entre el 12 % y 17 % de su extensión original, lo que lo convierte en uno de los hotspot de biodiversidad más amenazados del planeta [19]. Se estima que el Bosque Atlántico ha perdido aproximadamente el 88 % de su cobertura vegetal nativa, siendo reemplazado por paisajes dominados por agricultura, pasturas y áreas urbanas.

Misiones conserva cerca de 1,1 millones de hectáreas de este bosque a través del Corredor Verde, un sistema de áreas protegidas interconectadas que posiciona a la provincia como un refugio crítico para la biodiversidad regional y global [21].

3.1.2. Biodiversidad y especies en peligro

Esta región es hogar de una concentración excepcional de biodiversidad: a pesar de ocupar menos del 0,5 % del territorio argentino, la Selva Misionera alberga más del 50 % de la biodiversidad del país [21]. Se estima que contiene alrededor de 3000 especies de plantas vasculares, 554 especies de aves, 120 de mamíferos, 79 de reptiles y 55 de anfibios.

Entre las especies más emblemáticas se encuentra el yaguareté (*Panthera onca*), declarado Monumento Natural provincial en 1988 y nacional en 2001. Según el último censo binacional realizado en 2024, se estima una población de aproximadamente 84 individuos en el Corredor Verde entre Argentina y Brasil, con un rango estimado entre 64 y 110 ejemplares [22]. Esta

cifra representa casi la mitad de los menos de 250 yaguaretés adultos que se estima sobreviven en todo el territorio argentino.

Otras especies de importancia para la conservación incluyen el tapir (*Tapirus terrestris*), el oso hormiguero gigante (*Myrmecophaga tridactyla*), la yacutinga (*Aburria jacutinga*), el mono caí (*Sapajus nigritus*) y el águila harpía (*Harpia harpyja*). La presencia de estos grandes vertebrados es un indicador clave del estado de salud del ecosistema, pero su monitoreo en densas zonas de selva subtropical es extremadamente complejo.

3.1.3. Importancia ecológica de la región

La importancia de la Selva Misionera trasciende la mera preservación de especies; actúa como regulador climático, protector de cuencas hídricas y proveedor de servicios ecosistémicoss esenciales para la región, incluyendo la regulación del ciclo hidrológico, el secuestro de carbono y la protección contra la erosión del suelo.

El marco legal para la protección de estos ecosistemas está dado por la Ley 26.331 de Presupuestos Mínimos de Protección Ambiental de los Bosques Nativos [23], que establece un sistema de ordenamiento territorial basado en tres categorías de conservación:

Categoría I (Rojo): Sectores de muy alto valor de conservación que no deben transformarse. En Misiones, gran parte del Corredor Verde se encuentra clasificado en esta categoría.

Categoría II (Amarillo): Sectores de mediano valor de conservación donde se permite el aprovechamiento sostenible y la restauración.

Categoría III (Verde): Sectores de bajo valor de conservación que pueden transformarse parcialmente.

La provincia de Misiones gestiona actualmente 104 áreas naturales protegidas que suman aproximadamente 1,1 millones de hectáreas, lo que impone la necesidad de una vigilancia constante para evitar la degradación de estas áreas protegidas frente a actividades ilícitas como la caza furtiva y la tala clandestina.

3.2. Problemáticas de conservación en la región

A pesar de los esfuerzos institucionales y del marco legal establecido por la Ley 26.331 [23], las áreas protegidas de Misiones enfrentan amenazas persistentes que comprometen la integridad de sus ecosistemas.

3.2.1. Deforestación y fragmentación del hábitat

La pérdida de cobertura boscosa por actividades agrícolas no autorizadas y la extracción ilegal de madera noble continúan siendo problemas recurrentes. Un estudio de la Facultad de Agronomía de la Universidad de Buenos Aires reveló que entre 1990 y 2020 se perdieron aproximadamente 130 000 hectáreas de bosque nativo solo en el Corredor Verde, lo que representa un 13 % del área original [24]. El análisis indica que el 77 % de la deforestación ocurre en parcelas menores a 50 hectáreas, frecuentemente asociadas a ocupaciones espontáneas para cultivos de subsistencia.

Si bien las políticas de control han mostrado resultados positivos —con una reducción del 18 % en la deforestación durante 2025, alcanzando 4118 hectáreas anuales frente al promedio histórico de 5000 hectáreas [25]—, la fragmentación del hábitat sigue obligando a las especies de gran tamaño, como el yaguareté, a desplazarse por áreas no protegidas. Esto aumenta el riesgo de conflictos con humanos, atropellamientos en rutas, y reduce la viabilidad genética de las poblaciones aisladas.

3.2.2. Caza furtiva y tráfico de fauna

La caza furtiva representa uno de los mayores desafíos para la conservación en la región. A pesar de la prohibición total de la caza en la provincia, la incursión de cazadores en el interior de parques provinciales y reservas privadas es frecuente [26]. El problema presenta dos dimensiones principales: una cultural, llevada a cabo por residentes locales como actividad de subsistencia, y otra de carácter económico, impulsada por grupos organizados que buscan especies valiosas para el tráfico de fauna.

Los puntos más críticos se concentran en las zonas fronterizas con Brasil, especialmente en áreas como la reserva de biosfera Yabotí y los parques provinciales Piñalito, Urugua-í y Horacio Foerster. Entre las especies más afectadas se encuentran el tapir (*Tapirus terrestris*), la paca (*Cuniculus paca*), corzuelas, y aves como tucanes, guacamayos y loros. Los cazadores utilizan técnicas de acecho, trampas y cebaderos que no solo afectan a las especies objetivo, sino que degradan la fauna en su totalidad y ponen en riesgo la seguridad de los guardaparques durante los operativos de control.

3.2.3. Intrusión en áreas protegidas

La falta de un control perimetral efectivo en las más de 106 áreas protegidas de la provincia permite la entrada de personas y vehículos no autorizados para actividades ilícitas. Entre las más frecuentes se encuentran:

- Pesca ilegal en cursos de agua dentro de reservas.
- Desmonte encubierto para expansión de cultivos.
- Instalación de campamentos de caza con infraestructura permanente.

Sin un sistema de alerta temprana, estas intrusiones frecuentemente solo se descubren a posteriori durante patrullajes de rutina, cuando el daño ambiental ya ha sido perpetrado y los responsables se encuentran lejos del área.

3.2.4. Desafíos de la vigilancia en el terreno

La vigilancia efectiva de áreas protegidas en la Selva Misionera enfrenta obstáculos inherentes a las características del terreno y la extensión de las superficies a cubrir. Con aproximadamente 780 000 hectáreas distribuidas en más de 106 áreas protegidas, cualquier estrategia de monitoreo debe contemplar las siguientes limitaciones:

Extensión y accesibilidad: Las vastas superficies boscosas presentan terreno accidentado, cursos de agua y vegetación densa que dificultan el desplazamiento y limitan el alcance de las patrullas terrestres.

Cobertura de comunicaciones: La ausencia de infraestructura celular en zonas profundas de selva impide la comunicación en tiempo real y la coordinación de respuestas ante eventos detectados.

Latencia en la detección: El uso de cámara trampas tradicionales, si bien genera datos valiosos para investigación, almacena las imágenes localmente en tarjetas de memoria que deben ser retiradas físicamente, introduciendo demoras de semanas o meses entre la captura y el análisis.

Volumen de datos: Las cámaras convencionales generan grandes cantidades de imágenes, muchas disparadas por movimiento de vegetación o fauna no relevante, requiriendo un esfuerzo manual significativo para su revisión.

Estas limitaciones evidencian la necesidad de sistemas tecnológicos capaces de transmitir alertas en tiempo real, extender la cobertura de comunicaciones mediante redes autoorganizadas, y filtrar automáticamente las imágenes relevantes mediante técnicas de IA.

3.3. Sistemas de monitoreo actuales

Actualmente, el monitoreo biológico y de vigilancia en las áreas protegidas de Misiones emplea una combinación de métodos tradicionales que, si bien han demostrado ser efectivos para la investigación científica, presentan brechas operativas significativas cuando se aplican a la seguridad ambiental y la detección de actividades ilícitas.

3.3.1. Cámaras trampa convencionales

El uso de cámara trampas pasivas constituye el estándar de referencia para el monitoreo de fauna silvestre a nivel mundial [13]. Estos dispositivos se instalan en puntos estratégicos —senderos de fauna, fuentes de agua, zonas de paso— y capturan imágenes automáticamente mediante sensores de calor y movimiento (sensor PIR).

Las cámara trampas tradicionales presentan las siguientes características operativas:

- **Autonomía energética:** Funcionan con baterías AA o paneles solares, permitiendo operación autónoma durante meses.
- **Almacenamiento local:** Las imágenes se guardan en tarjetas SD que deben ser retiradas físicamente para su análisis.
- **Activación pasiva:** El sensor Infrarrojo Pasivo (PIR) detecta cambios de temperatura asociados al movimiento de animales o personas.
- **Operación silenciosa:** Los modelos con flash infrarrojo (Infrarrojo (IR)) permiten capturas nocturnas sin alertar a la fauna.

En Misiones, estas cámaras son utilizadas tanto por organismos gubernamentales como por organizaciones de conservación para el monitoreo del yaguareté, realizar censos de fauna, y documentar la biodiversidad de las reservas [22].

3.3.2. Patrullajes terrestres

Complementariamente, el Cuerpo de Guardaparques realiza patrullajes periódicos a pie, en vehículo o a caballo. Estos recorridos permiten detectar indicios de actividad ilícita (campamentos, trampas, rastros de tala) y establecer presencia disuasoria en las áreas protegidas.

Sin embargo, la efectividad de los patrullajes está limitada por factores logísticos: la densidad de la vegetación restringe la visibilidad, los desplazamientos consumen tiempo considerable, y la cobertura territorial depende directamente de los recursos humanos disponibles.

3.3.3. Brechas tecnológicas identificadas

El análisis de los sistemas actuales revela brechas críticas que limitan la capacidad de respuesta ante eventos de interés:

Latencia en la información: Las imágenes capturadas por cámaras tradicionales permanecen almacenadas localmente durante semanas o meses hasta su recolección. Un evento de caza furtiva registrado por una cámara solo será conocido mucho después de ocurrido, impidiendo cualquier acción disuasoria o legal inmediata.

Clasificación manual: El alto volumen de imágenes capturadas —frecuentemente disparadas por movimiento de vegetación, aves, o pequeños mamíferos no relevantes— demanda cientos de horas-hombre para su revisión y clasificación manual [12].

Ausencia de alertas: Los sistemas actuales no generan notificaciones en tiempo real. La información fluye de manera unidireccional desde el campo hacia los centros de análisis, sin retroalimentación inmediata.

Conectividad limitada: Las soluciones comerciales con transmisión celular, aunque existen en el mercado [16], [17], resultan inviables en las zonas interiores de la selva donde no existe cobertura de red móvil, además de representar costos operativos elevados para despliegues a escala.

Estas brechas fundamentan la necesidad de desarrollar un sistema que combine la robustez de las cámaras de campo con capacidades de transmisión en red, procesamiento automático mediante IA, y generación de alertas en tiempo real.

3.4. Justificación del proyecto

El análisis de las problemáticas de conservación y las limitaciones de los sistemas de monitoreo actuales permite identificar una brecha tecnológica que este proyecto propone abordar.

3.4.1. Urgencia de la problemática

La Selva Misionera representa el último remanente continuo del Bosque Atlántico en Argentina, albergando más del 50 % de la biodiversidad del país en menos del 0,5 % de su territorio [21]. La presión sobre este ecosistema es constante: entre 1990 y 2020 se perdieron 130 000 hectáreas de bosque nativo solo en el Corredor Verde [24], mientras que especies emblemáticas

como el yaguareté mantienen poblaciones críticamente bajas —aproximadamente 84 individuos según el último censo [22].

La caza furtiva, la tala ilegal y la intrusión en áreas protegidas continúan siendo amenazas activas que requieren respuestas inmediatas. Sin embargo, los sistemas de monitoreo actuales operan con latencias de semanas o meses, imposibilitando cualquier acción preventiva o disuasoria.

3.4.2. Oportunidad tecnológica

La convergencia de tres desarrollos tecnológicos recientes crea una oportunidad única para abordar esta problemática:

Microcontroladores de bajo costo: Plataformas como el ESP32 ofrecen capacidades de procesamiento, conectividad Wi-Fi y gestión de energía a costos inferiores a los 10 USD por unidad, permitiendo despliegues a escala con presupuestos limitados.

Redes mesh autoorganizadas: Protocolos como ESP-MESH permiten extender la conectividad en áreas sin infraestructura celular, creando redes que se adaptan dinámicamente a la topología del terreno y toleran fallos de nodos individuales.

Inteligencia artificial para visión: Modelos de código abierto como SpeciesNet [4] y MegaDetector [15] permiten clasificar automáticamente imágenes de cámara trampas, distinguiendo fauna silvestre de personas y vehículos con alta precisión.

3.4.3. Requisitos del sistema propuesto

Dada la crítica situación de biodiversidad en Misiones y las brechas identificadas en los sistemas actuales, se justifica el desarrollo de una solución tecnológica que cumpla con los siguientes requisitos:

1. **Bajo costo de implementación:** Permitiendo un despliegue distribuido con presupuesto limitado, utilizando hardware basado en ESP32 y software de código abierto.
2. **Conectividad independiente:** Empleando redes mesh que no requieran infraestructura celular preexistente, adaptándose a la topología irregular de la selva subtropical.
3. **Procesamiento inteligente:** Integrando modelos de IA como SpeciesNet para filtrar y clasificar imágenes automáticamente, reduciendo el volumen de datos a revisar manualmente.
4. **Alertas en tiempo real:** Generando notificaciones inmediatas ante la detección de eventos críticos, ya sea la presencia de fauna en peligro para fines de investigación, o la detección de intrusos para fines de seguridad.

Este proyecto no solo representa un aporte técnico en el área de redes de sensores e IA aplicada a la conservación, sino que constituye una herramienta de aplicación directa para fortalecer la vigilancia de las áreas protegidas de la región, democratizando el acceso a tecnologías de monitoreo inteligente que tradicionalmente han estado reservadas a instituciones con mayores recursos.

Capítulo 4

Objetivos y Alcance

4.1. Objetivo general

Demostrar la viabilidad técnica de un sistema de monitoreo de bajo costo para áreas protegidas, basado en nodos de cámara con conectividad mesh y clasificación automática de imágenes mediante IA. El sistema deberá ser capaz de detectar la presencia de animales, personas y vehículos, así como realizar una clasificación taxonómica de la fauna identificada. Todo esto reduciendo significativamente el tiempo de procesamiento de datos desde días o semanas —típico de las cámara trampas tradicionales— a minutos.

4.2. Objetivos específicos

1. **Diseñar e implementar un nodo de cámara autónomo** basado en microcontroladores ESP32, capaz de capturar imágenes mediante activación por sensor PIR y transmitirlas a través de una red mesh.

2. **Desarrollar una red mesh autoorganizada** utilizando el protocolo ESP-MESH, que permita extender la conectividad en áreas sin cobertura celular mediante el encadenamiento de nodos intermedios.

3. **Implementar un servidor de procesamiento** capaz de recibir las imágenes transmitidas por la red, clasificarlas automáticamente mediante modelos de IA preentrenados, y generar notificaciones ante la detección de eventos de interés.

4. **Integrar un modelo de clasificación de fauna** como SpeciesNet o MegaDetector, adaptado para identificar las principales categorías de interés: fauna silvestre, personas y vehículos.

5. **Validar el funcionamiento del sistema** en condiciones controladas, evaluando métricas de rendimiento como latencia de transmisión, consumo energético, precisión de clasificación y estabilidad de la red.

4.3. Alcance del proyecto

El presente proyecto tiene como alcance el desarrollo de un prototipo funcional que demuestre la viabilidad del concepto propuesto. Se contemplan los siguientes aspectos:

4.3.1. Dentro del alcance

- Diseño y construcción de nodos de cámara basados en módulos ESP32-CAM con sensor PIR comercial.
- Implementación de la red mesh con capacidad de transmisión de imágenes comprimidas.
- Desarrollo del servidor de recepción, almacenamiento y clasificación de imágenes.
- Integración de un modelo de IA preentrenado para la clasificación automática.
- Sistema de notificaciones mediante bot de bot de Telegram.
- Pruebas de funcionamiento en entorno controlado (laboratorio).
- Validación a pequeña escala en condiciones de campo reales, bajo circunstancias climáticas favorables, dadas las limitaciones de protección del prototipo.

4.3.2. Fuera del alcance

- Despliegue en áreas protegidas reales o ambientes de selva densa.
- Diseño de carcasas con grado de protección IP para uso permanente en exteriores.
- Entrenamiento de modelos de IA específicos para fauna de la región.
- Certificación de equipos para uso comercial o institucional.
- Integración con sistemas de gestión de áreas protegidas existentes.

4.4. Limitaciones

Es importante explicitar las limitaciones técnicas inherentes al prototipo desarrollado, tanto para establecer expectativas adecuadas como para orientar trabajos futuros.

4.4.1. Limitaciones de hardware

Autonomía energética: Los nodos de cámara ESP32 presentan un consumo energético significativo durante la transmisión Wi-Fi. Si bien se implementan estrategias de ahorro de energía, la autonomía con baterías es limitada comparada con cámaras trampa comerciales optimizadas durante años para este propósito. El prototipo utiliza un módulo convertidor buck basado en el circuito integrado LM2596, que permite alimentar el sistema con packs de baterías 18650 en diversas configuraciones (2S, 3S, 4S), proporcionando flexibilidad en el voltaje de entrada y permitiendo mayores capacidades de autonomía.

Sensor de movimiento: Se utiliza un sensor PIR domótico convencional, cuyas características de detección están optimizadas para uso en interiores. Esto implica que el sistema es capaz de detectar confiablemente animales de gran porte (tapires, yaguaretés, ciervos) y personas, pero puede no activarse consistentemente ante fauna de menor tamaño como aves o pequeños mamíferos.

Operación diurna: El módulo de cámara utilizado incorpora un lente con filtro IR, lo que limita su operación a condiciones de luz diurna. A diferencia de las cámaras trampa comerciales que incluyen iluminación infrarroja para capturas nocturnas, el prototipo desarrollado no cuenta con esta capacidad.

Resolución de imagen: Además de las limitaciones propias del sensor OV2640, los protocolos ESP-MESH y ESP-MDF imponen restricciones en el tamaño máximo de los datos transmitidos por paquete. Esto limita la resolución efectiva de las imágenes que pueden transmitirse a través de la red, requiriendo compresión adicional que afecta la calidad final.

4.4.2. Consideraciones sobre el tiempo de respuesta

Es necesario clarificar el uso del término tiempo real en el contexto de este proyecto. En sistemas embebidos, "tiempo real" típicamente se refiere a la capacidad de responder a eventos dentro de plazos determinísticos estrictos, frecuentemente en el orden de ms o µs.

En el contexto de este proyecto, la noción de tiempo real se utiliza en un sentido operativo diferente: la capacidad de reducir la latencia de procesamiento de imágenes desde **días o semanas** a **minutos**. Los tiempos prolongados característicos de las cámara trampas tradicionales se deben a la necesidad de desplazarse físicamente hasta cada dispositivo para recolectar las tarjetas de memoria, y posteriormente procesar las imágenes de forma manual o semiautomática. Este cambio de escala temporal, aunque no constituye tiempo real en el sentido determinístico, representa una mejora de varios órdenes de magnitud que habilita respuestas operativas antes imposibles.

4.4.3. Limitaciones de alcance

Rango de comunicación: El alcance de cada nodo mesh depende de las condiciones del terreno y la vegetación. En zonas de selva densa, puede ser necesario un mayor número de nodos intermedios para cubrir distancias equivalentes a las alcanzables en campo abierto.

Ancho de banda: La transmisión de imágenes a través de la red mesh consume ancho de banda significativo. Capturas simultáneas en múltiples nodos pueden generar congestión y aumentar la latencia.

Clasificación de especies: Los modelos de IA utilizados están entrenados con datasets globales que pueden no incluir todas las especies presentes en la Selva Misionera, afectando la precisión de clasificación taxonómica a nivel de especie.

Capítulo 5

Marco Teórico

5.1. Cámaras Trampa

Las cámara trampas constituyen una de las herramientas más valiosas para el estudio de la fauna silvestre en entornos naturales. Estos dispositivos, diseñados para capturar imágenes de manera automática cuando detectan movimiento o calor, permiten documentar la presencia y comportamiento de animales sin requerir la presencia continua de investigadores, minimizando así la perturbación del ecosistema [13].

5.1.1. Evolución histórica

El concepto de fotografía automática de fauna tiene sus orígenes a finales del siglo XIX, cuando el naturalista George Shiras III desarrolló las primeras técnicas de fotografía nocturna con disparador por cable. Sin embargo, las cámaras trampa modernas surgieron en la década de 1990 con la miniaturización de componentes electrónicos y el desarrollo de sensores de movimiento de bajo consumo [27].

La evolución tecnológica ha atravesado varias etapas significativas:

- **Era analógica (1990-2000):** Primeras cámaras comerciales con película fotográfica y sensores PIR básicos.
- **Digitalización (2000-2010):** Transición a sensores digitales Complementary Metal-Oxide-Semiconductor (CMOS) y almacenamiento en tarjetas de memoria.
- **Alta definición (2010-2020):** Mejoras en resolución, capacidad de video HD, y reducción significativa de costos.
- **Conectividad (2020-presente):** Incorporación de módulos celulares, Wi-Fi y satelitales para transmisión remota de datos.

Esta evolución ha democratizado el acceso a la tecnología, pasando de equipos de alto costo utilizados exclusivamente en investigación científica a dispositivos accesibles para proyectos de conservación con recursos limitados.

5.1.2. Componentes principales

Una cámara trampa moderna está compuesta por varios elementos integrados que trabajan en conjunto para detectar, capturar y almacenar imágenes de fauna:

Sensor de movimiento: Típicamente un sensor PIR que detecta cambios en la radiación infrarroja emitida por cuerpos calientes en movimiento. El sensor tiene un campo de detección definido por su ángulo y alcance, generalmente entre 10 m y 20 m.

Módulo de cámara: Sensor de imagen CMOS con resoluciones que varían desde 5 hasta 48 megapíxeles en modelos actuales. La calidad del lente y el tamaño del sensor influyen directamente en la nitidez de las imágenes, especialmente en condiciones de baja luminosidad.

Sistema de iluminación: Para capturas nocturnas, las cámaras incluyen LEDs IR o flash blanco. Los LEDs infrarrojos (850 nm o 940 nm) permiten capturas sin alertar a los animales, aunque con menor calidad de color que el flash blanco.

Almacenamiento: Tarjetas de memoria SD o microSD, con capacidades típicas de 32 GB a 512 GB, permitiendo almacenar miles de imágenes o cientos de horas de video antes de requerir descarga.

Alimentación: Baterías AA, baterías de litio recargables, o paneles solares auxiliares. La autonomía depende de la frecuencia de activaciones y puede variar de semanas a varios meses.

Carcasa protectora: Gabinete resistente a la intemperie, generalmente con clasificación IP65 o superior, que protege los componentes electrónicos de lluvia, polvo e impactos.

5.1.3. Funcionamiento técnico

El ciclo de operación de una cámara trampa sigue una secuencia bien definida que optimiza el consumo energético mientras maximiza la probabilidad de captura:

1. **Modo de reposo:** La cámara permanece en estado de bajo consumo, con solo el sensor PIR activo monitoreando cambios térmicos en su campo de visión.

2. **Detección:** Cuando el sensor detecta un cambio significativo en la radiación infrarroja —indicativo de un cuerpo caliente en movimiento—, genera una señal de activación.

3. **Activación:** El microprocesador despierta el sistema de cámara, proceso que introduce una latencia conocida como *trigger time* o tiempo de disparo, que varía entre 0,1 s y 2 s según el modelo.

4. **Captura:** Se toman una o más fotografías (ráfaga), o se inicia una grabación de video de duración configurable.

5. **Almacenamiento:** Las imágenes se comprimen (generalmente en formato Joint Photographic Experts Group (JPEG)) y se graban en la tarjeta de memoria junto con metadatos como fecha, hora, temperatura y fase lunar.

6. **Retorno al reposo:** Tras un período de recuperación configurable (para evitar múltiples disparos del mismo evento), la cámara vuelve al modo de bajo consumo.

La efectividad de la captura depende de parámetros configurables como el ángulo del sensor, la sensibilidad de detección, el tiempo de disparo, y el intervalo mínimo entre capturas consecutivas.

5.1.4. Tipos y clasificación

Las cámaras trampa pueden clasificarse según diferentes criterios técnicos:
Por tipo de sensor de activación:

Pasivas (PIR): Utilizan sensores infrarrojos pasivos que detectan el calor emitido por animales. Son las más comunes por su bajo consumo y costo, pero pueden fallar con animales de sangre fría o en días muy calurosos cuando la diferencia térmica es mínima.

Activas (haz infrarrojo): Emplean un emisor y receptor de haz infrarrojo; la captura se dispara cuando el animal interrumpe el haz. Ofrecen mayor precisión posicional, pero requieren instalación más compleja con componentes separados.

Por tipo de iluminación nocturna:

Flash blanco: Proporciona imágenes a color de alta calidad incluso de noche, pero puede alertar o espantar a algunos animales.

LED infrarrojo visible (850 nm): Emite un tenue resplandor rojo visible para algunos animales, pero menos intrusivo que el flash blanco.

LED infrarrojo invisible (940 nm, "no-glow"): Completamente invisible para humanos y la mayoría de los animales, aunque produce imágenes de menor contraste y alcance reducido.

Por conectividad:

Autónomas: Almacenan datos localmente, requiriendo visitas físicas para recuperar las imágenes.

Celulares: Transmiten imágenes mediante redes 3G/4G/5G a servidores remotos o aplicaciones móviles.

Wi-Fi: Permiten descarga inalámbrica cuando el usuario se encuentra en rango cercano.

Satelitales: Utilizan redes como Iridium o Globalstar para transmisión desde ubicaciones sin cobertura celular, con costos operativos elevados.

5.1.5. Aplicaciones en conservación

Las cámaras trampa se han convertido en herramientas indispensables para múltiples aspectos de la investigación y gestión de fauna silvestre:

Inventarios de biodiversidad: Permiten documentar la riqueza de especies en un área protegida, incluyendo especies esquivas, nocturnas o de baja densidad que serían difíciles de detectar mediante observación directa.

Monitoreo poblacional: Mediante la identificación de individuos por patrones naturales (manchas, rayas, cicatrices), es posible estimar abundancia poblacional utilizando modelos de captura-recaptura [21].

Estudios de comportamiento: La operación continua 24/7 permite documentar patrones de actividad, uso de hábitat, interacciones entre especies y comportamiento reproductivo.

Detección de amenazas: Las cámaras pueden registrar evidencia de actividades de caza furtiva, tala ilegal, o intrusión humana en áreas protegidas, aunque con la limitación de que la información solo está disponible al recuperar físicamente los dispositivos.

5.1.6. Limitaciones técnicas

A pesar de su utilidad, las cámaras trampa convencionales presentan limitaciones que motivan el desarrollo de sistemas más avanzados:

Latencia de información: Las imágenes permanecen almacenadas localmente hasta que un investigador visita el sitio, introduciendo demoras de semanas o meses entre la captura y el análisis.

Falsas activaciones: Movimiento de vegetación por viento, cambios bruscos de temperatura, o pequeños animales no objetivo generan un alto porcentaje de imágenes vacías o irrelevantes, que en algunos estudios supera el 80 % del total capturado.

Volumen de datos: Proyectos con múltiples cámaras generan miles o millones de imágenes que requieren clasificación manual, proceso que consume cientos de horas-hombre.

Mantenimiento de campo: Las visitas periódicas para cambio de baterías y tarjetas de memoria implican costos logísticos y potencial perturbación del sitio de monitoreo.

Zona de detección fija: El campo de visión del sensor PIR limita el área efectiva de monitoreo, requiriendo múltiples unidades para cubrir extensiones significativas.

Estas limitaciones fundamentan la necesidad de integrar las cámaras trampa con tecnologías de comunicación inalámbrica e inteligencia artificial, tal como propone el presente trabajo.

5.2. Internet de las Cosas (IoT)

El IoT representa un paradigma de conectividad donde objetos físicos —equipados con sensores, capacidad de procesamiento y conectividad de red— intercambian datos con otros dispositivos y sistemas a través de internet o redes locales. Esta tecnología ha experimentado un crecimiento exponencial en las últimas décadas, habilitando aplicaciones que van desde la automatización del hogar hasta el monitoreo ambiental a gran escala [13].

En el contexto de la conservación ambiental, el IoT permite la creación de redes de monitoreo autónomas capaces de operar en ubicaciones remotas con mínima intervención humana. Los sistemas de cámara trampas inteligentes representan una aplicación específica de este paradigma, donde la combinación de sensores, procesamiento embebido y conectividad inalámbrica extiende las capacidades tradicionales de monitoreo de fauna [9].

5.2.1. Arquitecturas IoT

Una arquitectura típica de IoT se compone de tres capas fundamentales:

Capa de Percepción: Formada por los sensores y actuadores que interactúan directamente con el entorno físico. En este proyecto, incluye los sensores PIR y los módulos de cámara.

Capa de Red: Encargada de la transmisión de datos desde los nodos sensores hacia los sistemas de procesamiento. Aquí se implementa la red ESP-MESH que permite la comunicación entre nodos.

Capa de Aplicación: Donde los datos son procesados, almacenados y presentados al usuario final. Corresponde al servidor Django con el modelo de IA y el sistema de notificaciones.

El sistema desarrollado en esta tesis abarca estas tres capas, utilizando nodos ESP32 en la percepción, ESP-MESH en la red y un servidor Django con SpeciesNet en la aplicación.

5.2.2. Protocolos de comunicación inalámbrica

La elección del protocolo de comunicación es crítica en entornos rurales o silvestres donde la infraestructura de red tradicional es inexistente. Cada tecnología presenta compromisos entre alcance, consumo energético y ancho de banda:

Bluetooth Low Energy (BLE): Ideal para corto alcance (hasta 100 m) y muy bajo consumo, pero con ancho de banda insuficiente para transmisión de imágenes.

ZigBee: Diseñado para redes de sensores con bajo consumo, pero limitado en velocidad de transferencia y complejidad de implementación.

LoRa/LoRaWAN: Excelente para muy largo alcance (varios kilómetros) con bajo consumo, pero el ancho de banda extremadamente limitado (decenas de bytes por segundo) lo hace inadecuado para transmisión de imágenes [9].

Wi-Fi (802.11): Ofrece alto ancho de banda suficiente para imágenes, pero típicamente requiere infraestructura centralizada y tiene mayor consumo energético.

Las redes mesh basadas en Wi-Fi, como ESP-MESH, ofrecen un equilibrio adecuado: permiten la transmisión de imágenes gracias al ancho de banda del protocolo 802.11, mientras extienden la cobertura sin depender de una única estación base o infraestructura celular preexistente.

5.3. Redes Mesh

Una red de malla o mesh es una topología en la que los nodos se conectan entre sí de forma dinámica, cooperando para propagar los datos a través de la red. Esta arquitectura es especialmente robusta frente a fallos de nodos individuales, ya que la red puede reconfigurarse automáticamente para encontrar rutas alternativas [28].

5.3.1. Topologías de red

Las topologías de red determinan cómo se organizan y comunican los dispositivos:

Topología en estrella: Común en redes Wi-Fi domésticas, donde todos los dispositivos dependen de un único Punto de Acceso central. Si el AP falla, toda la red queda inoperativa.

Topología en árbol: Estructura jerárquica donde existe un nodo raíz que actúa como puerta de enlace hacia redes externas, nodos intermedios que extienden la cobertura, y nodos hoja que únicamente transmiten sus propios datos.

Topología mesh pura: Cada nodo puede comunicarse con múltiples vecinos, creando redundancia en las rutas de comunicación.

ESP-MESH implementa una topología de árbol con capacidades mesh: cada nodo puede actuar simultáneamente como estación (conectada a un padre) y como Punto de Acceso (para nodos hijos), utilizando el modo AP-STA. Esto facilita el despliegue en terrenos difíciles o boscosos, donde los obstáculos físicos limitan la línea de vista directa.

5.3.2. ESP-MESH: características técnicas

ESP-MESH es el protocolo desarrollado por Espressif basado en el estándar IEEE 802.11, que permite conectar cientos de dispositivos ESP32 en una sola red [28]. Sus principales características incluyen:

Autoorganización: Los nodos seleccionan automáticamente su padre óptimo basándose en la intensidad de señal (Received Signal Strength Indicator (RSSI)) y la carga de la red.

Autocuración: Si un nodo intermedio falla o pierde conectividad, sus nodos descendientes buscan y se conectan automáticamente a un nuevo padre, sin intervención manual.

Profundidad configurable: La red puede extenderse hasta una profundidad máxima definida por el usuario (típicamente 6-10 niveles).

Transmisión de datos: Soporta comunicación unicast (a un nodo específico), multicast (a un grupo) y broadcast (a todos los nodos).

5.3.3. Componente Mwifi del ESP-MDF

Mwifi es un componente del ESP-MDF que proporciona una capa de abstracción sobre ESP-MESH, simplificando el desarrollo de aplicaciones. Sus principales aportes incluyen:

- APIs de alto nivel para envío y recepción de datos fragmentados
- Gestión automática de la fragmentación de paquetes grandes (como imágenes) que exceden el tamaño máximo de trama Wi-Fi
- Soporte para transmisión TCP y UDP sobre la red mesh
- Eventos de callback para monitorear el estado de la topología
- Herramientas de depuración integradas

La capacidad de fragmentar y reensamblar datos es crítica para este proyecto, ya que las imágenes capturadas por los nodos exceden significativamente el tamaño máximo de paquete soportado por el protocolo mesh.

5.4. Inteligencia Artificial aplicada a visión por computadora

La visión por computadora busca emular la capacidad humana de interpretar imágenes digitales. Los avances recientes en este campo, impulsados por el aprendizaje automático y específicamente por el aprendizaje profundo (*deep learning*), han revolucionado el procesamiento automático de imágenes de cámara trampas [29].

5.4.1. Redes neuronales convolucionales (CNN)

Las CNNs son arquitecturas de redes neuronales especializadas en procesar datos con estructura de cuadrícula, como las imágenes [30]. Funcionan mediante la aplicación secuencial de filtros convolucionales que extraen características en niveles de abstracción creciente: las primeras capas detectan bordes y texturas básicas, mientras que las capas profundas reconocen patrones complejos como formas de animales o rostros humanos.

Esta capacidad de extracción automática de características —que elimina la necesidad de diseño manual de descriptores— ha convertido a las CNNs en la herramienta fundamental para la clasificación de fauna y la detección de intrusos en imágenes de cámara trampas [12].

5.4.2. Detección de objetos con YOLO

YOLO (*You Only Look Once*) es uno de los algoritmos de detección de objetos más utilizados debido a su velocidad y precisión [2]. A diferencia de métodos tradicionales que analizan una imagen en múltiples pasadas, YOLO trata la detección como un problema de regresión único, prediciendo las bounding boxs y las probabilidades de clase simultáneamente para toda la imagen en una sola evaluación.

La arquitectura YOLOv5 [3], implementada sobre PyTorch, ofrece un excelente equilibrio entre rendimiento computacional y exactitud. Sus variantes (desde YOLOv5n para dispositivos

embebidos hasta YOLOv5x para máxima precisión) permiten adaptar el modelo a los recursos disponibles.

5.4.3. MegaDetector

MegaDetector es un modelo de detección desarrollado por Microsoft AI for Earth [15], específicamente diseñado para imágenes de cámara trampas. A diferencia de clasificadores que intentan identificar especies directamente, MegaDetector se enfoca en una tarea más simple: detectar la presencia y ubicación de tres categorías principales:

- Animales (sin distinción de especie)
- Personas
- Vehículos

Esta aproximación en dos etapas —primero detectar, luego clasificar— ha demostrado ser más robusta y generalizable que intentar resolver ambos problemas simultáneamente [11].

5.4.4. SpeciesNet de Google

SpeciesNet es un sistema de IA de código abierto desarrollado por Google [4], diseñado específicamente para el análisis de imágenes de cámara trampas. Combina las capacidades de detección con modelos de clasificación taxonómica entrenados con millones de imágenes de cientos de especies a nivel global.

El pipeline de SpeciesNet opera en dos fases:

1. **Detección:** Identifica regiones de interés en la imagen (animales, personas, vehículos) similar a MegaDetector.
2. **Clasificación:** Para las regiones identificadas como animales, aplica un clasificador taxonómico que predice familia, género y especie.

En este proyecto, SpeciesNet actúa como el núcleo de procesamiento del servidor, validando si las imágenes capturadas por los nodos contienen fauna de interés, personas o vehículos, y generando las alertas correspondientes.

5.5. Tecnologías de desarrollo

Para la implementación del sistema se han seleccionado herramientas que garantizan modularidad, facilidad de mantenimiento e interoperabilidad, priorizando tecnologías de código abierto cuando sea posible.

5.5.1. Microcontroladores ESP32

El ESP32 es un System on Chip (SoC) de bajo costo diseñado por Espressif Systems, que integra conectividad Wi-Fi (802.11 b/g/n) y Bluetooth en un único chip [31]. Sus principales características técnicas incluyen:

- Procesador Xtensa LX6 de doble núcleo a 240 MHz
- 520 KB de SRAM interna
- Soporte para memoria flash externa (hasta 16 MB)
- Múltiples modos de bajo consumo (deep sleep $< 10\,\mu A$)
- Rico conjunto de periféricos: SPI, I2C, UART, ADC, DAC, PWM

La variante ESP32-CAM incorpora un módulo de cámara OV2640 y ranura para tarjeta microSD, proporcionando una plataforma compacta para aplicaciones de captura de imagen con conectividad inalámbrica.

5.5.2. ESP-IDF y ESP-MDF

El ESP-IDF (IoT Development Framework) es el entorno de desarrollo oficial para el ESP32 [28], basado en Free Real-Time Operating System (FreeRTOS). Proporciona control preciso sobre el hardware, gestión de energía y acceso a todas las funcionalidades del chip.

Sobre este entorno, el ESP-MDF (Mesh Development Framework) agrega herramientas específicas para redes mesh:

- Componente Mwifi para comunicación mesh de alto nivel
- Gestión automática de topología y enrutamiento
- Capacidades de actualización remota de firmware Over-The-Air (OTA)
- Herramientas de depuración y monitoreo de red

5.5.3. Contenedorización con Docker

Docker es una plataforma de contenedorización que permite empaquetar aplicaciones junto con todas sus dependencias en unidades estandarizadas llamadas contenedores. El uso de Docker y Docker Compose en este proyecto facilita el despliegue del servidor de procesamiento en cualquier máquina, asegurando que el entorno de ejecución sea idéntico al de desarrollo.

Los componentes del servidor se organizan en contenedores separados:

- Contenedor de SpeciesNet para inferencia de IA
- Contenedor de base de datos PostgreSQL
- Contenedor del backend Django

5.5.4. Framework Django

Django es un framework web de alto nivel escrito en Python que fomenta el desarrollo rápido y un diseño limpio. Se utiliza para construir el servidor central que recibe las imágenes de la red mesh, interactúa con el sistema de IA y gestiona el almacenamiento persistente.

Su arquitectura MVT (Modelo-Vista-Plantilla) permite una separación clara entre la lógica de datos, el procesamiento de solicitudes y la presentación de resultados al usuario.

5.6. Sensores y actuadores

Los componentes físicos periféricos permiten la interacción del nodo con el entorno, detectando eventos y capturando información visual. La selección de estos componentes determina las capacidades y limitaciones del sistema.

5.6.1. Sensores de movimiento PIR

El sensor PIR detecta movimiento mediante la medición de cambios en los niveles de radiación infrarroja emitida por objetos en su campo de visión. El sensor contiene un elemento piroeléctrico dividido en dos zonas: cuando un objeto emisor de calor (como un humano o animal) cruza de una zona a otra, se genera una diferencia de voltaje que activa la detección.

Los sensores PIR domóticos utilizados en este proyecto tienen un rango de detección optimizado para objetos del tamaño de humanos, lo que limita la capacidad de detectar fauna de pequeño porte, pero es adecuado para animales medianos y grandes como tapires, yaguaretés o ciervos.

5.6.2. Módulos de cámara OV2640

El OV2640 es un sensor de imagen CMOS de 2 megapíxeles fabricado por OmniVision, ampliamente utilizado en aplicaciones embebidas por su bajo costo y consumo moderado. Sus especificaciones técnicas principales incluyen:

- Resolución máxima: 1600 × 1200 píxeles (UXGA)
- Formatos de salida: JPEG, RGB565, YUV422
- Interfaz: SCCB (compatible con I2C) para control, paralelo para datos
- Compresión JPEG integrada en hardware
- Consumo típico: 125 mW en operación activa

El módulo está integrado en placas de desarrollo como la ESP32-CAM, que incluye slot para tarjeta microSD, LED flash, y antena Wi-Fi externa opcional. La compresión JPEG en hardware es particularmente útil para reducir el tamaño de las imágenes antes de su transmisión por la red mesh.

El lente incluido en los módulos comerciales típicamente incorpora un filtro IR, lo que limita la operación a condiciones de luz diurna.

5.7. Diseño y fabricación digital

La protección física del hardware es fundamental para la operación del sistema en entornos exteriores, donde los componentes electrónicos están expuestos a condiciones ambientales adversas.

5.7.1. Modelado 3D

El diseño de la carcasa protectora se realiza mediante herramientas de CAD, permitiendo crear estructuras precisas adaptadas a los componentes específicos del nodo. Los requisitos principales del diseño incluyen:

- Protección contra humedad y polvo
- Aperturas dimensionadas para la lente de cámara y sensor PIR
- Espacio para el módulo convertidor buck y conexión a pack de baterías 18650 externo
- Puntos de montaje para fijación en árboles o postes
- Acceso a la ranura de tarjeta SD sin desmontar el dispositivo

5.7.2. Fabricación aditiva

La impresión 3D mediante tecnología Modelado por Deposición Fundida (FDM) es la técnica utilizada para materializar los diseños. Esta tecnología permite iterar rápidamente sobre prototipos y fabricar piezas personalizadas sin inversión en moldes.

La selección del material de impresión es crítica para la durabilidad. Para los prototipos de validación de este proyecto se utiliza Ácido Poliláctico (PLA), dado que las pruebas de campo se realizarán bajo condiciones climáticas favorables y no se planea exposición prolongada a la intemperie. Sin embargo, el mismo modelo 3D puede ser fabricado en materiales más resistentes como Tereftalato de Polietileno Glicol (PETG) o Acrilonitrilo Estireno Acrilato (ASA) en fases posteriores de despliegue que requieran operación permanente en exteriores.

Capítulo 6

Metodología

6.1. Enfoque metodológico

El desarrollo de este proyecto siguió un enfoque **iterativo e incremental**, donde cada componente del sistema fue desarrollado, probado y refinado antes de avanzar al siguiente. Esta aproximación permitió identificar y corregir problemas tempranamente, reduciendo el riesgo de fallas críticas en etapas avanzadas.

Se adoptó la metodología ágil Kanban para la gestión del proyecto, caracterizada por:

- Visualización del flujo de trabajo mediante un tablero con columnas (Por hacer, En progreso, En revisión, Completado).
- Limitación del trabajo en progreso para mantener el enfoque en tareas específicas.
- Entrega continua de funcionalidades incrementales.
- Flexibilidad para re-priorizar tareas según las necesidades emergentes.

Esta metodología resultó particularmente adecuada para un proyecto que combina hardware y software, donde los ciclos de iteración pueden variar significativamente entre componentes: el desarrollo de firmware requiere ciclos más largos de compilación y programación, mientras que el desarrollo del servidor permite iteraciones más rápidas.

6.2. Etapas del desarrollo

El proyecto se estructuró en las siguientes etapas principales, representadas en el diagrama de Gantt de la Figura 6.1:

1. **Investigación y diseño:** Revisión del estado del arte, definición de requisitos y diseño de la arquitectura general del sistema.
2. **Desarrollo del firmware:** Implementación del código para los nodos ESP32, incluyendo captura de imágenes, integración del sensor PIR y comunicación mesh.
3. **Desarrollo del servidor:** Implementación del backend Django, integración con SpeciesNet y sistema de notificaciones.

4. **Diseño y fabricación de hardware:** Modelado 3D de la carcasa e impresión de prototipos.

5. **Integración:** Conexión de todos los componentes del sistema y verificación del flujo completo de datos.

6. **Pruebas y validación:** Evaluación del sistema en condiciones controladas y de campo.

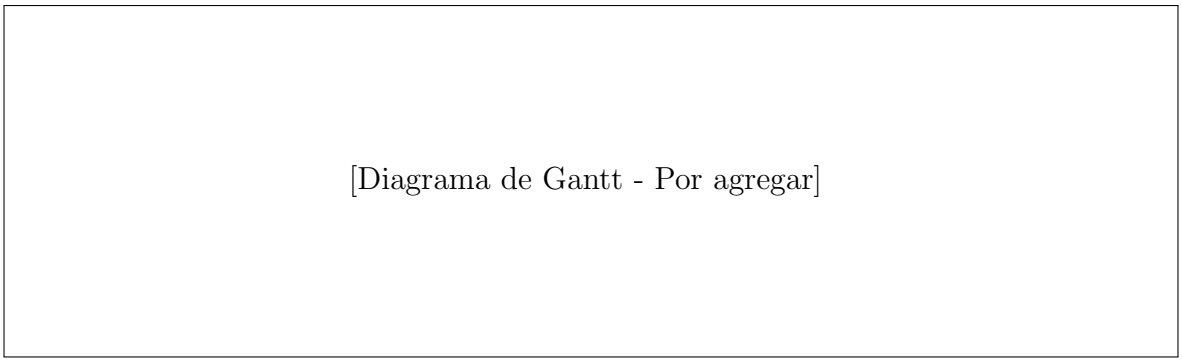

Figura 6.1: Cronograma de desarrollo del proyecto.

6.3. Herramientas y tecnologías utilizadas

6.3.1. Entorno de desarrollo

Editor de código: Visual Studio Code con extensiones para desarrollo en C/C++ (ESP-IDF), Python y Docker.

Framework de firmware: ESP-IDF y ESP-MDF para el desarrollo del código de los nodos ESP32.

Framework web: Django para el servidor de aplicación.

Bot de notificaciones: Librería `python-telegram-bot` para la implementación del sistema de alertas mediante Telegram.

Contenedorización: Docker y Docker Compose para el despliegue de servicios.

6.3.2. Diseño y fabricación 3D

Software de modelado: Autodesk Fusion 360 con licencia educativa para el diseño de la carcasa del nodo.

Software de laminado: PrusaSlicer para la preparación de modelos 3D y generación de código G para la impresora.

Impresora 3D: Creality Ender 3 con cama caliente, utilizando filamento PLA para los prototipos de validación.

6.3.3. Control de versiones y despliegue

El código fuente del proyecto fue organizado en cuatro repositorios Git alojados en GitHub:

1. Repositorio del firmware para nodos de cámara:
   https://github.com/fabcontigiani/mesh-node-capstone-project
2. Repositorio del firmware para nodo raíz:
   https://github.com/fabcontigiani/root-node-capstone-project
3. Repositorio del servidor de aplicación:
   https://github.com/fabcontigiani/server-capstone-project
4. Repositorio del servidor de inferencia y anotación de imágenes:
   https://github.com/fabcontigiani/wildlife-detection-capstone-project

6.3.4. Despliegue continuo

Se implementó una estrategia de despliegue continuo mediante GitHub Actions para automatizar las siguientes tareas:

- Construcción automática de imágenes Docker al realizar cambios en las ramas principales de los repositorios.
- Publicación de las imágenes en el registro de contenedores de GitHub (GitHub Container Registry).

Esta automatización garantizó que las versiones desplegadas del servidor siempre correspondieran al código más reciente validado, reduciendo errores humanos en el proceso de despliegue.

6.4. Métricas de evaluación

Para validar el cumplimiento de los objetivos del proyecto, se definieron las siguientes métricas de evaluación:

6.4.1. Métricas de rendimiento de red

Latencia de transmisión: Tiempo transcurrido desde la captura de una imagen en el nodo hasta su recepción completa en el servidor. Se mide en segundos y representa la capacidad del sistema para proporcionar información en tiempos operativamente útiles.

Distancia máxima entre nodos: Distancia física máxima a la cual dos nodos pueden mantener una conexión mesh funcional, medida en metros. Esta métrica determina la densidad de nodos necesaria para cubrir un área determinada.

Estabilidad de la red: Tasa de pérdida de paquetes y frecuencia de reconexiones durante operación prolongada. Una red estable debe mantener tasas de pérdida inferiores al 5 %.

6.4.2. Métricas de clasificación

Precisión de detección: Proporción de imágenes correctamente clasificadas como conteniendo animales, personas, vehículos o imágenes vacías.

Precisión de clasificación taxonómica: Para las imágenes con animales, proporción de especies correctamente identificadas a nivel de familia o género.

6.4.3. Métricas de consumo energético

Consumo promedio del nodo: Consumo eléctrico promedio del nodo de cámara durante un ciclo típico de operación (espera + captura + transmisión), medido en miliamperios (mA).

Autonomía estimada: Tiempo de operación estimado con una batería de capacidad conocida, calculado a partir del consumo promedio.

6.5. Estrategia de pruebas

6.5.1. Pruebas incrementales de firmware

Durante el desarrollo del firmware, cada módulo fue probado de forma aislada antes de su integración:

- Pruebas de captura de imagen con el módulo OV2640
- Pruebas de detección del sensor PIR
- Pruebas de conexión y transmisión mesh entre nodos
- Pruebas de consumo energético

6.5.2. Pruebas de integración

Una vez validados los componentes individuales, se realizaron pruebas de integración verificando:

- Transmisión de imágenes desde nodos de cámara hasta el servidor
- Procesamiento de imágenes por el modelo de IA
- Generación de notificaciones mediante el bot de bot de Telegram

6.5.3. Pruebas de sistema

Las pruebas del sistema completo se realizaron en dos fases:

Fase 1 - Pruebas de laboratorio: Validación del funcionamiento del sistema en condiciones controladas de interior, verificando la correcta integración de todos los componentes y midiendo las métricas de rendimiento base.

Fase 2 - Pruebas de campo: Instalación del sistema en un entorno exterior bajo condiciones climáticas favorables. Durante varias horas se monitoreó el funcionamiento del sistema, registrando las métricas de latencia, estabilidad de red, consumo energético y precisión de clasificación en condiciones realistas.

Capítulo 7

Diseño e Implementación del Sistema

7.1. Arquitectura general

El sistema propuesto se compone de tres capas principales que trabajan de forma coordinada para lograr la detección y notificación de eventos en tiempo operativo:

1. **Capa de percepción:** Formada por los nodos de cámara (mesh-node) equipados con sensor PIR y módulo de cámara, responsables de detectar movimiento y capturar imágenes.

2. **Capa de red:** Constituida por la red ESP-MESH que permite la comunicación entre nodos de cámara y el nodo raíz, extendiendo la cobertura sin infraestructura adicional.

3. **Capa de aplicación:** Compuesta por el servidor de procesamiento que recibe las imágenes, realiza la inferencia mediante IA y notifica al usuario a través de bot de Telegram.

El flujo de datos del sistema opera de la siguiente manera:

1. El sensor PIR detecta movimiento y genera una interrupción en el microcontrolador.

2. El nodo de cámara captura una imagen en formato JPEG.

3. La imagen se fragmenta y transmite a través de la red mesh hasta el nodo raíz.

4. El nodo raíz envía la imagen al servidor mediante una petición HTTP POST.

5. El servidor procesa la imagen con SpeciesNet (que incluye MegaDetector y modelos de clasificación taxonómica).

6. Si se detecta un objeto de interés con alto grado de confianza, el bot de Telegram notifica al usuario enviando la imagen original, la imagen anotada con las detecciones, y la lista de clasificaciones.

7. Las imágenes y metadatos se almacenan en la base de datos.

7.2. Nodo de cámara (mesh-node)

7.2.1. Selección de componentes

El nodo de cámara se construye alrededor de los siguientes componentes principales:

Microcontrolador: Módulo ESP32-CAM de AI-Thinker, que integra un ESP32 con cámara OV2640 y ranura para tarjeta microSD en una placa compacta.

Sensor de movimiento: Módulo HC-SR501, un sensor PIR de bajo costo con sensibilidad y tiempo de retardo ajustables mediante potenciómetros.

Regulador de voltaje: Módulo convertidor buck basado en el circuito integrado LM2596, que acepta voltajes de entrada entre 4,5 V y 40 V, permitiendo alimentación desde packs de baterías baterías 18650 en configuración 2S (7,4 V nominal).

7.2.2. Diseño del firmware

El firmware del nodo de cámara se desarrolló utilizando ESP-IDF y ESP-MDF, implementando las siguientes funcionalidades:

- Inicialización y configuración de la cámara OV2640
- Configuración del pin GPIO como interrupción externa para el sensor PIR
- Conexión a la red ESP-MESH como nodo hijo
- Captura y compresión de imágenes en formato JPEG
- Almacenamiento local de imágenes en tarjeta microSD como respaldo
- Transmisión de imágenes mediante Mwifi
- Gestión del modo power save para reducir consumo energético

7.2.3. Captura de imágenes

La captura de imágenes se realiza con los siguientes parámetros:

Resolución: 640 × 480 píxeles (VGA)

Formato: JPEG con compresión alta

Tamaño típico: Inferior a 8 kB por imagen, ajustando el nivel de compresión para cumplir con el límite de ESP-MDF

El tamaño de imagen se mantiene por debajo del límite de 8 kB impuesto por ESP-MDF para el tamaño máximo de paquete de aplicación. Este límite determina el nivel de compresión JPEG necesario, priorizando la transmisión confiable sobre la calidad de imagen.

7.2.4. Integración del sensor PIR

El sensor HC-SR501 se conecta a un pin GPIO del ESP32-CAM configurado como fuente de interrupción externa. Cuando el sensor detecta movimiento (cambio en la radiación infrarroja), genera una señal HIGH que dispara la rutina de captura de imagen.

El sensor permite ajustar:

- **Sensibilidad:** Distancia de detección (3 m a 7 m)
- **Tiempo de retardo:** Duración de la señal HIGH (5 s a 300 s)

7.2.5. Compresión y transmisión

La transmisión de imágenes a través de la red mesh utiliza el componente Mwifi del ESP-MDF. Dado que las imágenes se mantienen por debajo del límite de 8 KB, se transmiten como un único paquete de aplicación, simplificando el manejo de datos.

El componente Mwifi proporciona:

- Fragmentación automática a nivel de capa de enlace
- Ventana deslizante para control de flujo
- Retransmisión automática de paquetes perdidos
- Reensamblado de fragmentos en el destino

7.2.6. Carcasa para impresión 3D

El diseño de la carcasa se basó en un modelo público bajo licencia Creative Commons 4.0 International, disponible en Printables[1], el cual fue modificado utilizando Autodesk Fusion 360 para adaptarlo a los requerimientos específicos del proyecto. La carcasa fue impresa en PLA y contempla:

- Apertura frontal para la lente de la cámara
- Apertura superior para el domo del sensor PIR
- Espacio interno para el módulo convertidor buck basado en LM2596
- Orificio para cable de alimentación hacia pack de baterías externo
- Provisión para conexión de antena externa
- Ranura de acceso a la tarjeta microSD
- Puntos de montaje para fijación en superficies

El plano 2D del diseño final se incluye en el Anexo B (Figura B.1 y Figura B.2). El modelo 3D modificado está disponible públicamente en Printables[2].

[1] www.printables.com/model/520378-case-for-esp32_cam-trail-camera-pir-sensor-18650-b
[2] www.printables.com/model/1532349-case-for-esp32_cam-trail-camera-pir-sensor-buck-co

7.3. Nodo raíz (root-node)

7.3.1. Diseño del hardware

El nodo raíz utiliza una placa de desarrollo ESP32 DevKit V1, sin módulo de cámara, que actúa como puerta de enlace entre la red mesh y el servidor de aplicación.

7.3.2. Firmware del nodo raíz

El firmware del nodo raíz implementa:

- Inicialización como nodo raíz de la red ESP-MESH
- Recepción y reensamblado de imágenes fragmentadas desde los nodos hijos
- Conexión Wi-Fi a un router doméstico para acceso a internet
- Cliente HTTP para envío de imágenes al servidor

7.3.3. Conexión con servidor TCP

El nodo raíz envía las imágenes recibidas al servidor de aplicación mediante peticiones HTTP POST. Cada imagen se transmite como cuerpo de la petición junto con metadatos (identificador del nodo origen, timestamp).

La conexión al servidor se realiza a través de Wi-Fi conectado a un router doméstico con acceso a internet. En la sección de trabajos futuros se discute la posibilidad de reemplazar este enlace por tecnologías de mayor alcance para despliegues en zonas remotas.

7.3.4. Gestión de la red mesh

El nodo raíz es responsable de:

- Mantener la topología de la red mesh
- Gestionar la conexión y desconexión de nodos hijos
- Enrutar los datos desde cualquier nodo hasta sí mismo
- Proporcionar sincronización de tiempo a la red

La red está configurada para soportar hasta 6 niveles de profundidad, aunque en las pruebas realizadas se utilizaron únicamente 3 nodos.

7.4. Red mesh

7.4.1. Topología de la red

La red ESP-MESH se organiza en una topología de árbol donde:

Nodo raíz: Único punto de conexión con la red externa (router Wi-Fi)

Nodos intermedios: Pueden existir nodos que actúen como repetidores

Nodos hoja: Los nodos de cámara que capturan y transmiten imágenes

Cada nodo opera en modo AP-STA, actuando simultáneamente como punto de acceso para nodos hijos y como estación conectada a su nodo padre.

7.4.2. Protocolo de comunicación

La comunicación utiliza el protocolo ESP-MESH implementado sobre Wi-Fi 802.11, con las siguientes características:

- Autoorganización: los nodos seleccionan su padre óptimo basándose en RSSI
- Autocuración: reconexión automática ante fallos de nodos
- Soporte para mensajes unicast y broadcast
- Fragmentación y reensamblado de datos grandes mediante Mwifi

7.4.3. Formato de datos

Las imágenes se transmiten como datos binarios (JPEG) precedidos de un encabezado que incluye el tamaño total de la imagen y el identificador del nodo origen.

7.5. Servicio de detección (wildlife-detection)

7.5.1. Arquitectura del servicio

El servicio de detección se implementa como un contenedor Docker independiente que expone una Application Programming Interface (API) REST para inferencia de imágenes.

7.5.2. Contenedor Docker con SpeciesNet

El contenedor incluye:

- SpeciesNet con modelos preentrenados
- MegaDetector para detección de animales, personas y vehículos
- Modelos YOLOv5 para clasificación taxonómica
- Servidor HTTP basado en LitServe

7.5.3. API de inferencia

El servicio expone un endpoint `/predict` que:

1. Recibe una imagen en formato JPEG
2. Ejecuta el pipeline de detección y clasificación
3. Retorna las detecciones con bounding boxs, confianzas y etiquetas taxonómicas

7.5.4. Detección y clasificación

El pipeline de SpeciesNet opera en dos fases:

1. **Detección:** MegaDetector identifica regiones de interés clasificándolas como animal, persona o vehículo.
2. **Clasificación:** Para las regiones detectadas como animales, se aplican modelos de clasificación taxonómica que predicen familia, género y especie.

7.6. Servidor de aplicación (server)

7.6.1. Arquitectura Django

El servidor de aplicación se desarrolló utilizando Django, implementando:

- Endpoint HTTP para recepción de imágenes desde el nodo raíz
- Integración con el servicio de detección vía HTTP REST
- Almacenamiento de imágenes y metadatos
- Bot de Telegram para notificaciones
- Interfaz web para visualización de resultados

7.6.2. Gestión de imágenes

Las imágenes se almacenan en un volumen Docker compartido:

- Imágenes originales recibidas desde los nodos
- Imágenes anotadas con bounding boxs generadas por el servicio de detección

En la base de datos se almacenan únicamente los paths a las imágenes junto con los metadatos de detección.

7.6.3. Bot de Telegram y sistema de alertas

El sistema de alertas utiliza la librería `python-telegram-bot` para enviar notificaciones cuando se detectan objetos de interés con alto grado de confianza.

Cada notificación incluye:

- Imagen original capturada por el nodo
- Imagen anotada con las bounding boxs de las detecciones
- Información sobre cantidad y confianza de las detecciones
- Etiquetas de clasificación taxonómica de mayor confianza

7.6.4. Base de datos PostgreSQL

La base de datos PostgreSQL almacena:

- Registros de imágenes con paths y timestamps
- Resultados de detección (clases, confianzas, coordenadas)
- Clasificaciones taxonómicas
- Configuración del sistema

7.6.5. Despliegue con Docker Compose

El servidor se despliega mediante Docker Compose, orquestando los siguientes servicios:

- Contenedor de la aplicación Django
- Contenedor de PostgreSQL
- Contenedor del servicio de detección con SpeciesNet

7.7. Alimentación y consumo energético

Los nodos de cámara se alimentan mediante packs de baterías 18650 en configuración 2S (dos celdas en serie), proporcionando un voltaje nominal de 7,4 V. El módulo convertidor buck LM2596 reduce este voltaje a 5 V para alimentar el ESP32-CAM.

El consumo energético se midió en tres estados de operación:

Modo power save: Consumo reducido mientras el nodo espera eventos, con transmisión Wi-Fi limitada pero CPU activo.

Captura de imagen: Consumo elevado momentáneo durante la activación de la cámara y compresión JPEG.

Transmisión: Máximo consumo durante el envío de datos por Wi-Fi a través de la red mesh.

Es importante destacar que el sistema utiliza el modo power save de ESP-MDF en lugar de deep sleep. El modo power save reduce el consumo limitando el tiempo de transmisión Wi-Fi activa, mientras mantiene el CPU funcionando para responder a eventos de la red y del sensor PIR.

Capítulo 8

Pruebas y Resultados

8.1. Ambiente de pruebas

8.1.1. Configuración del hardware

Las pruebas se realizaron utilizando la siguiente configuración de hardware:

- 3 nodos de cámara (ESP32-CAM con sensor HC-SR501)
- 1 nodo raíz (ESP32 DevKit V1)
- Packs de baterías 18650 en configuración 2S
- Router Wi-Fi doméstico para conexión a internet
- Servidor local con procesador multinúcleo para inferencia (sin GPU), accesible mediante túnel VPN (Pangolin, basado en WireGuard) y proxy reverso (Traefik)

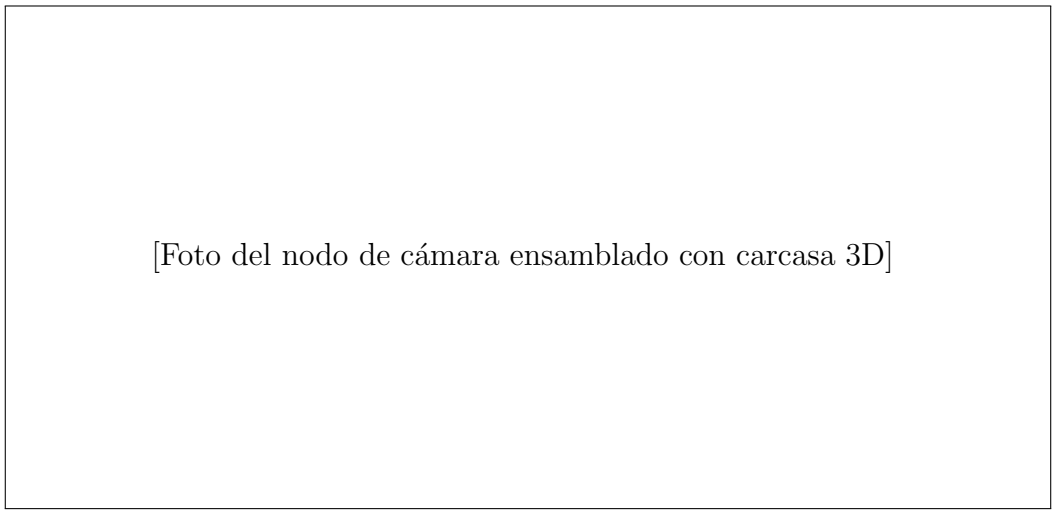

Figura 8.1: Nodo de cámara completamente ensamblado.

8.1.2. Configuración del software

El software se desplegó con la siguiente configuración:

- Firmware compilado con ESP-IDF v5.x y ESP-MDF
- Contenedores Docker ejecutándose en servidor Linux
- Base de datos PostgreSQL
- Servicio de inferencia con SpeciesNet y LitServe
- Bot de Telegram configurado para notificaciones

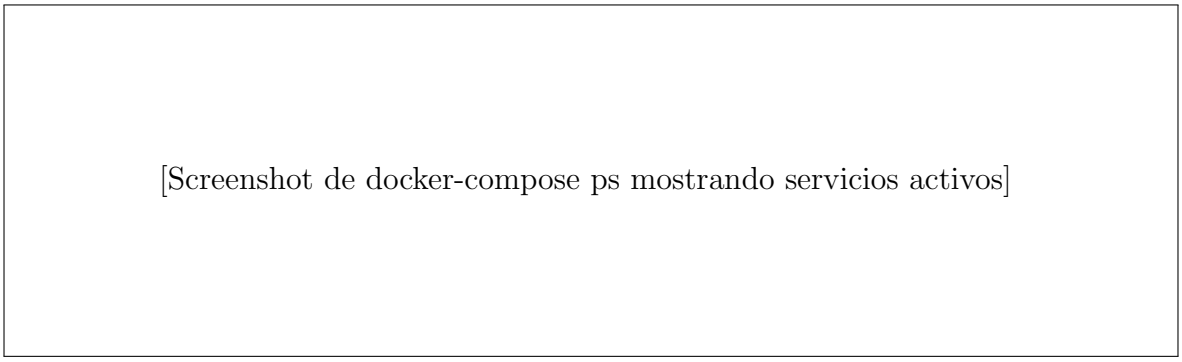

Figura 8.2: Servicios del servidor ejecutándose mediante Docker Compose.

8.2. Pruebas de laboratorio

Las pruebas de laboratorio se realizaron en condiciones controladas de interior, verificando el funcionamiento individual de cada componente antes de la integración.

8.2.1. Pruebas de captura de imagen

Se verificó la correcta captura de imágenes por parte del módulo OV2640, evaluando:

- Inicialización correcta de la cámara
- Calidad de imagen a diferentes niveles de compresión
- Tamaño de archivo resultante (objetivo: < 8 KB)
- Tiempo de captura y compresión

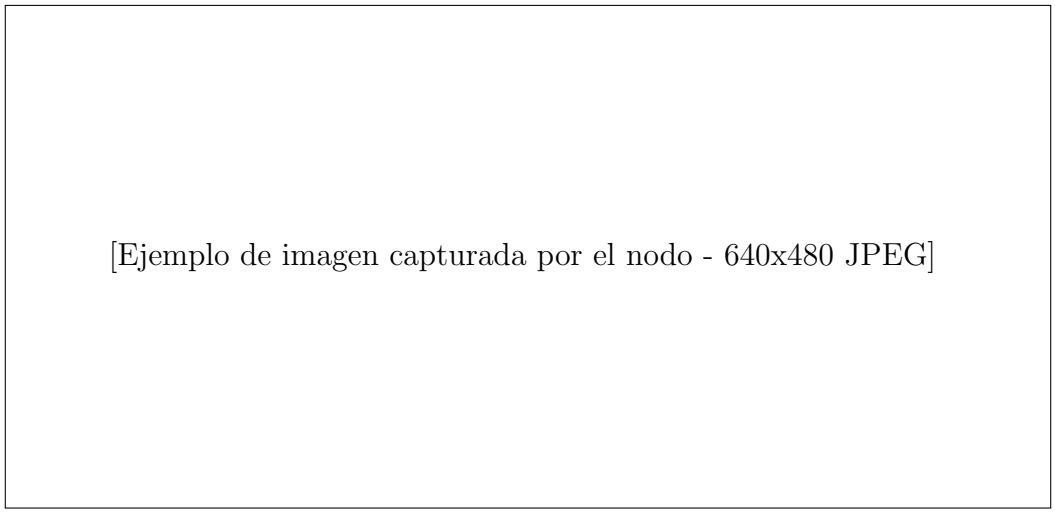

Figura 8.3: Ejemplo de imagen capturada por un nodo de cámara.

8.2.2. Pruebas del sensor PIR

Se evaluó el funcionamiento del sensor HC-SR501:

- Respuesta a movimiento de personas a diferentes distancias
- Tiempo de respuesta desde detección hasta interrupción
- Configuración óptima de sensibilidad y retardo
- Tasa de falsas activaciones

Tabla 8.1: Resultados de pruebas del sensor PIR.

Parámetro	**Valor**
Distancia máxima de detección	[X] m
Tiempo de respuesta promedio	[X] ms
Falsas activaciones por hora	[X]

8.2.3. Pruebas de transmisión mesh

Se verificó la transmisión de imágenes a través de la red mesh:

- Establecimiento de conexión entre nodos
- Transmisión exitosa de imágenes completas
- Tiempo de transmisión desde nodo hasta servidor
- Comportamiento ante pérdida de conexión

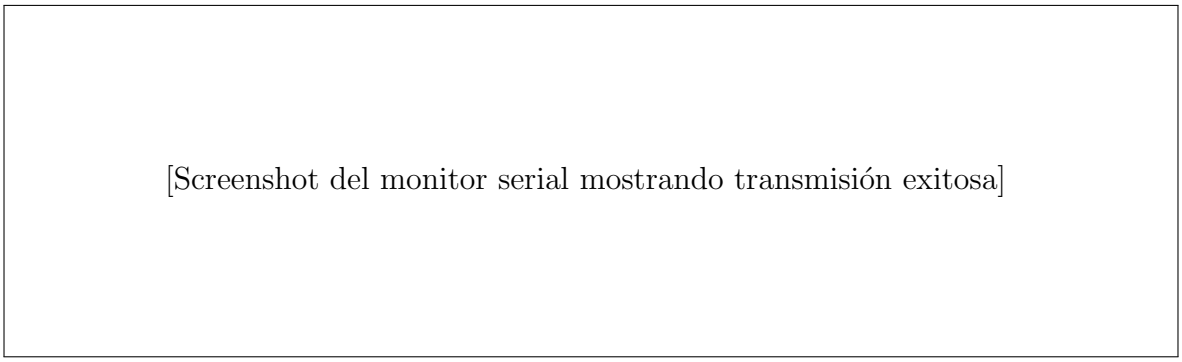

Figura 8.4: Log de transmisión de imagen a través de la red mesh.

8.2.4. Pruebas del sistema de clasificación

Se evaluó el pipeline de detección y clasificación con imágenes de prueba:

- Detección correcta de animales, personas y vehículos
- Precisión de las bounding boxs
- Clasificación taxonómica de animales detectados
- Tiempo de inferencia por imagen

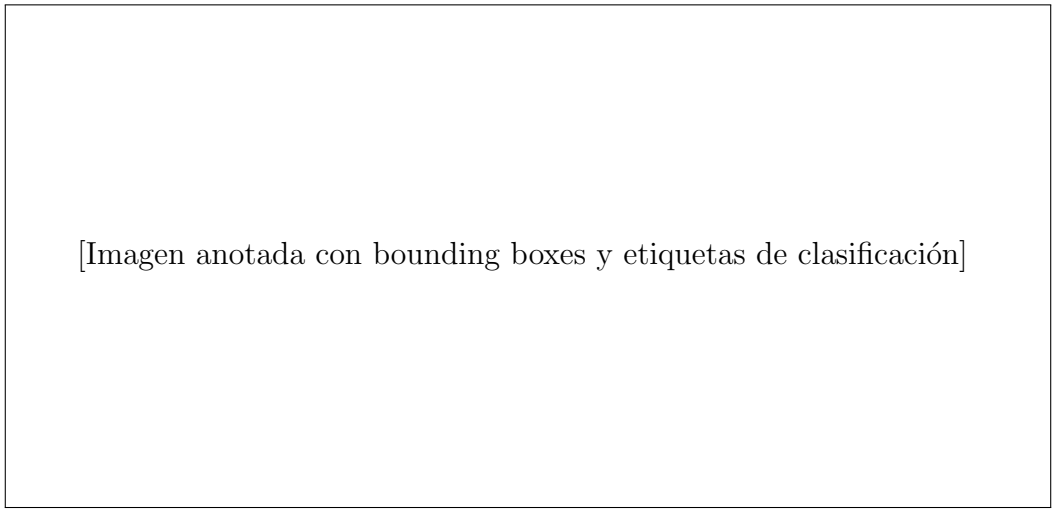

Figura 8.5: Ejemplo de imagen procesada por el sistema de detección.

8.2.5. Pruebas del sistema de alertas

Se verificó el funcionamiento del bot de Telegram:

- Recepción correcta de notificaciones
- Envío de imagen original y anotada

- Información de detecciones y clasificaciones
- Tiempo desde captura hasta notificación

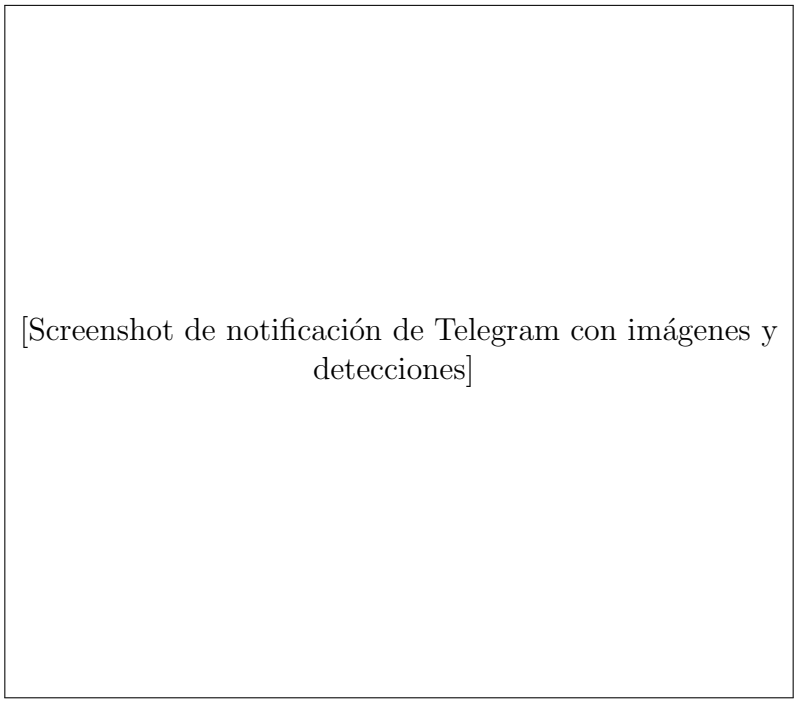

Figura 8.6: Notificación recibida en Telegram con resultados de detección.

8.3. Pruebas de campo

8.3.1. Configuración del entorno

Las pruebas de campo se realizaron en un entorno exterior, instalando los nodos de cámara en ubicaciones representativas de un despliegue real.

8.3.2. Condiciones de las pruebas

Las pruebas de campo se realizaron bajo las siguientes condiciones:

- Duración: [X] horas de operación continua
- Condiciones climáticas: [describir condiciones]
- Número de nodos desplegados: 3
- Distancia entre nodos: [X] metros
- Distancia al router Wi-Fi: [X] metros

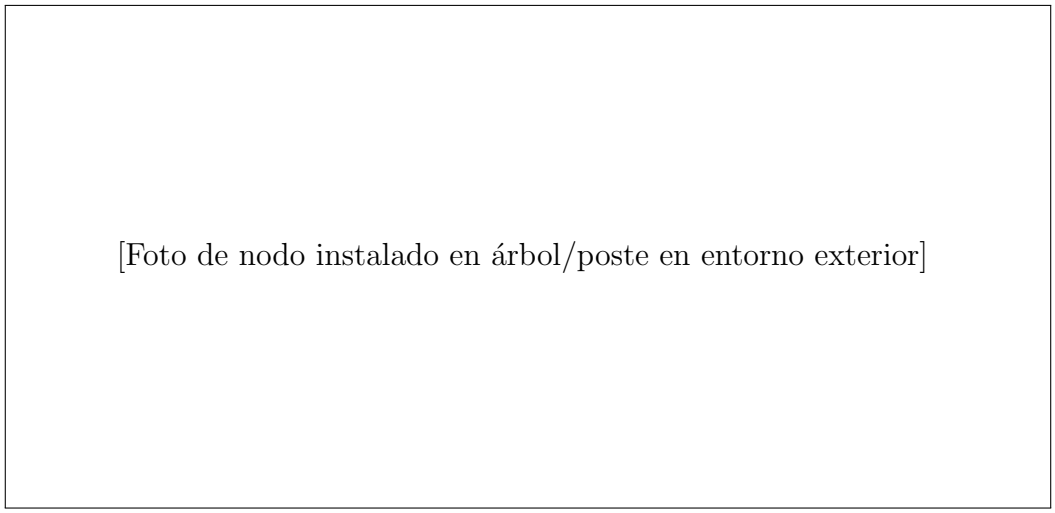

Figura 8.7: Nodo de cámara desplegado en entorno de pruebas de campo.

8.3.3. Resultados obtenidos

Durante las pruebas de campo se registraron los siguientes resultados:

- Imágenes capturadas: [X]
- Detecciones exitosas: [X]
- Falsas activaciones: [X]
- Tiempo de operación sin fallos: [X] horas

8.4. Métricas de rendimiento de red

8.4.1. Latencia de transmisión

Se midió el tiempo transcurrido desde la captura de imagen hasta la recepción en el servidor:

Tabla 8.2: Latencia de transmisión medida.

Métrica	**Valor**
Latencia mínima	[X] s
Latencia promedio	[X] s
Latencia máxima	[X] s
Desviación estándar	[X] s

8.4.2. Distancia máxima entre nodos

Se evaluó la distancia máxima a la cual los nodos pueden mantener conexión estable:

Tabla 8.3: Distancia máxima entre nodos.

Condición	Distancia máxima
Línea de vista directa	[X] m
Con obstáculos (vegetación)	[X] m
Interior (paredes)	[X] m

8.4.3. Estabilidad de la conexión

Se monitoreó la estabilidad de la red durante operación prolongada:

- Número de reconexiones: [X]
- Paquetes perdidos: [X]%
- Tiempo de reconexión promedio: [X] s

8.5. Métricas de clasificación

8.5.1. Precisión de detección

Se evaluó la precisión del sistema para detectar los diferentes tipos de objetos:

Tabla 8.4: Precisión de detección por categoría.

Categoría	Verdaderos +	Falsos +	Precisión
Animales	[X]	[X]	[X] %
Personas	[X]	[X]	[X] %
Vehículos	[X]	[X]	[X] %
Vacías	[X]	[X]	[X] %

8.5.2. Precisión de clasificación taxonómica

Para las imágenes con animales detectados, se evaluó la precisión de la clasificación taxonómica:

Tabla 8.5: Precisión de clasificación taxonómica.

Nivel taxonómico	Precisión
Familia	[X] %
Género	[X] %
Especie	[X] %

8.6. Métricas de consumo energético

8.6.1. Consumo promedio del nodo

Se midió el consumo eléctrico del nodo en diferentes estados de operación:

Tabla 8.6: Consumo energético por estado de operación.

Estado	**Consumo** (mA)
Modo power save	[X]
Captura de imagen	[X]
Transmisión Wi-Fi	[X]
Consumo promedio ponderado	[X]

8.6.2. Autonomía estimada

Basándose en el consumo promedio y la capacidad de las baterías:

- Capacidad del pack 2S: [X] mAh
- Consumo promedio: [X] mA
- Autonomía teórica: [X] horas
- Autonomía medida en pruebas: [X] horas

8.7. Análisis de resultados

Los resultados obtenidos demuestran la viabilidad del sistema propuesto para el monitoreo de áreas protegidas. A continuación se presenta un análisis de los principales hallazgos:

8.7.1. Cumplimiento de objetivos

Detección de fauna: [Análisis del desempeño en detección]

Latencia de respuesta: [Análisis de tiempos de respuesta]

Cobertura de red: [Análisis de alcance y estabilidad]

Autonomía energética: [Análisis de consumo y duración]

8.7.2. Limitaciones observadas

Durante las pruebas se identificaron las siguientes limitaciones:

- Limitación 1 observada
- Limitación 2 observada
- Limitación 3 observada

8.7.3. Comparación con objetivos planteados

Tabla 8.7: Comparación de resultados con objetivos planteados.

Métrica	**Objetivo**	**Resultado**
Latencia de respuesta	¡X min	[X] min
Precisión de detección	¿[X]%	[X]%
Autonomía	¿X horas	[X] horas
Cobertura	¿[X] m	[X] m

Capítulo 9

Conclusiones

9.1. Conclusiones generales

El presente trabajo demostró la viabilidad técnica de un sistema de monitoreo de bajo costo para áreas protegidas, basado en nodos de cámara con conectividad mesh y clasificación automática de imágenes mediante IA.

Los principales logros alcanzados incluyen:

- Se diseñó e implementó un sistema completo de monitoreo que integra hardware de bajo costo (ESP32-CAM, sensores PIR) con tecnologías de IA de vanguardia (SpeciesNet, MegaDetector).
- Se logró reducir significativamente el tiempo de procesamiento de imágenes desde días o semanas —típico de las cámara trampas tradicionales— a minutos, habilitando respuestas operativas antes imposibles.
- Se demostró que las redes mesh basadas en ESP-MESH son una alternativa viable para extender la cobertura de monitoreo en áreas sin infraestructura de red, manteniendo la capacidad de transmitir imágenes.
- Se validó la integración de modelos de detección y clasificación taxonómica en un pipeline de procesamiento automatizado, capaz de distinguir entre fauna, personas y vehículos.
- Se implementó un sistema de alertas en tiempo operativo mediante Telegram, proporcionando notificaciones inmediatas con imágenes anotadas y clasificaciones a los usuarios finales.

El costo total de los componentes de hardware para un nodo de cámara se mantiene significativamente por debajo de las cámaras trampa comerciales con capacidades similares de conectividad, validando la propuesta de un sistema de bajo costo accesible para proyectos de conservación con recursos limitados.

9.2. Aportes del trabajo

Este trabajo realiza los siguientes aportes:

Arquitectura de sistema integrado: Se propone una arquitectura que combina captura distribuida, transmisión mesh y procesamiento centralizado con IA, aplicable a diversos escenarios de monitoreo ambiental.

Prototipo funcional de bajo costo: Se desarrolló un prototipo completo y funcional utilizando componentes comerciales de bajo costo, con diseños de carcasa listos para fabricación mediante impresión 3D.

Integración de SpeciesNet en pipeline de tiempo operativo: Se demostró la viabilidad de utilizar modelos de clasificación taxonómica entrenados en grandes datasets de cámara trampas en un sistema de respuesta rápida.

Código abierto: Todo el código desarrollado (firmware, servidor, servicio de detección) está disponible en repositorios públicos de GitHub, permitiendo la reproducción y extensión del trabajo por parte de otros investigadores y desarrolladores.

9.3. Trabajos futuros

A partir de los resultados obtenidos, se identifican las siguientes líneas de trabajo futuro:

Visión nocturna: Incorporar módulos de cámara con capacidad infrarroja para permitir la captura de imágenes durante la noche, extendiendo significativamente la utilidad del sistema para monitoreo de fauna con hábitos nocturnos.

Conectividad de largo alcance: Reemplazar el enlace Wi-Fi entre el nodo raíz y el servidor por tecnologías de mayor alcance como LoRa, redes celulares (4G/LTE) o enlaces satelitales, permitiendo despliegues en zonas remotas sin cobertura de internet convencional.

Optimización del consumo energético: Investigar estrategias adicionales de ahorro de energía, incluyendo paneles solares y algoritmos de gestión de batería más sofisticados.

Implementación de nodo rama: Una alternativa considerada pero no implementada —para reducir la complejidad del prototipo— es el diseño de un tercer tipo de nodo: un "nodo rama" o repetidor dedicado que mantendría la red mesh activa. Esto permitiría que los nodos de cámara funcionen como nodos hoja y utilicen el modo deep sleep verdadero, despertando únicamente ante interrupciones del sensor PIR, conectándose brevemente a la red para transmitir la imagen, y volviendo a dormir. Esta arquitectura podría reducir drásticamente el consumo promedio de los nodos de cámara.

Procesamiento en el borde: Explorar la posibilidad de realizar inferencia directamente en los nodos de cámara utilizando modelos optimizados (TinyML), reduciendo el volumen de datos transmitidos y la latencia de respuesta.

Mejoras en clasificación: Entrenar o afinar modelos específicos para la fauna de la región del Bosque Atlántico, mejorando la precisión de clasificación para especies locales.

Interfaz de usuario: Desarrollar una interfaz web más completa que permita visualización en mapa, gestión de múltiples despliegues y generación de reportes automatizados.

9.4. Recomendaciones

Basándose en la experiencia adquirida durante el desarrollo y pruebas del sistema, se realizan las siguientes recomendaciones para futuras implementaciones:

1. **Selección de ubicaciones:** Considerar cuidadosamente la distancia entre nodos y la presencia de obstáculos al planificar el despliegue. La vegetación densa reduce significativamente el alcance de la señal Wi-Fi.

2. **Protección contra elementos:** Para despliegues de largo plazo, utilizar materiales de impresión 3D resistentes a la intemperie (PETG o ASA) y considerar el uso de selladores adicionales.

3. **Capacidad de batería:** Dimensionar los packs de baterías según la frecuencia esperada de activaciones y la autonomía deseada. El diseño del prototipo con convertidor buck permite alimentación desde un amplio rango de voltajes de entrada (4,5 V a 40 V), facilitando el uso de packs de baterías 18650 en diversas configuraciones (2S, 3S, 4S).

4. **Selección del sensor de movimiento:** El sensor PIR domótico utilizado en el prototipo (HC-SR501) está optimizado para detectar personas y puede no activarse consistentemente ante fauna de menor porte. Para despliegues enfocados en monitoreo de fauna silvestre, se recomienda evaluar sensores PIR de mayor sensibilidad o tecnologías alternativas de detección de movimiento diseñadas para aplicaciones de vida silvestre.

5. **Monitoreo remoto:** Implementar sistemas de monitoreo de estado de los nodos (nivel de batería, conectividad) para identificar problemas antes de que resulten en pérdida de datos.

Capítulo 10

Bibliografía

[1] S. Schneider, G. W. Taylor y S. Kremer, «Deep Learning Object Detection Methods for Ecological Camera Trap Data,» en *2018 15th Conference on Computer and Robot Vision (CRV)*, IEEE, 2018, págs. 321-328. DOI: 10.1109/CRV.2018.00052.

[2] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection,» *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, págs. 779-788, 2016.

[3] G. Jocher. «YOLOv5 by Ultralytics.» Repositorio oficial de YOLOv5, visitado 27 de dic. de 2024. dirección: https://github.com/ultralytics/yolov5.

[4] T. Gadot et al., «To crop or not to crop: Comparing whole-image and cropped classification on a large dataset of camera trap images,» *IET Computer Vision*, 2024.

[5] G. Barrero y A. Schmunck, «Desarrollo de una microcámara de vigilancia para la protección de la fauna salvaje,» Tesis de Grado, Universidad Nacional de Misiones, 2023. dirección: https://drive.google.com/file/d/1jJDM-Mc8kHJw3Rcgl0tf4Mq3MxNyqXW5.

[6] F. González et al., «Inteligencia artificial para la multi-clasificación de fauna en fotografías automáticas utilizadas en investigación científica,» en *XXIV Workshop de Investigadores en Ciencias de la Computación (WICC 2022)*, Mendoza, Argentina, abr. de 2022, ISBN: 978-987-48222-3-9. dirección: https://sedici.unlp.edu.ar/handle/10915/144851.

[7] B. S. Z. Abood, M. B. M, Z. A. Almoussawi, N. Shilpa y A. M. Shakir, «Revolutionizing Wildlife Monitoring: A Novel Approach to Camera Trap Image Analysis with YOLOv5,» en *2023 3rd International Conference on Mobile Networks and Wireless Communications (ICMNWC)*, IEEE, dic. de 2023. DOI: 10.1109/ICMNWC60182.2023.10435785. dirección: http://dx.doi.org/10.1109/ICMNWC60182.2023.10435785.

[8] Y. Njathi, L. Wanjiku, L. Mugambi, J. N. Kabi, G. Kiarie y C. w. Maina, «Efficient Camera Trap Image Annotation Using YOLOv5,» en *2023 IEEE AFRICON*, IEEE, sep. de 2023. DOI: 10.1109/AFRICON55910.2023.10293724. dirección: http://dx.doi.org/10.1109/AFRICON55910.2023.10293724.

[9] R. C. Whytock et al., «Real-time alerts from AI-enabled camera traps using the Iridium satellite network: A case-study in Gabon, Central Africa,» *Methods in Ecology and Evolution*, vol. 14, n.º 3, págs. 867-874, 2023, ISSN: 2041-210X. DOI: 10.1111/2041-210X.14036. dirección: http://dx.doi.org/10.1111/2041-210X.14036.

[10] D. Mallya. «AiCatcher: Extending machine intelligence into the wild,» visitado 27 de dic. de 2024. dirección: https://deepakmallya.com/aicatcher.

[11] J. Vélez et al., «An evaluation of platforms for processing camera-trap data using artificial intelligence,» *Methods in Ecology and Evolution*, vol. 14, n.º 2, págs. 459-477, 2022, ISSN: 2041-210X. DOI: 10.1111/2041-210X.14044. dirección: http://dx.doi.org/10.1111/2041-210X.14044.

[12] M. A. Tabak et al., «Machine learning to classify animal species in camera trap images: Applications in ecology,» *Methods in Ecology and Evolution*, vol. 10, n.º 4, págs. 585-590, 2019. DOI: 10.1111/2041-210X.13120.

[13] R. Steenweg et al., «Scaling-up camera traps: monitoring the planet's biodiversity with networks of remote sensors,» *Frontiers in Ecology and the Environment*, vol. 15, n.º 1, págs. 26-34, 2017. DOI: 10.1002/fee.1448.

[14] A. Hernandez, Z. Miao, L. Vargas, R. Dodhia y J. Lavista, *Pytorch-Wildlife: A Collaborative Deep Learning Framework for Conservation*, 2024. arXiv: 2405.12930 [cs.CV]. dirección: https://arxiv.org/abs/2405.12930.

[15] S. Beery, D. Morris y S. Yang, «Efficient Pipeline for Camera Trap Image Review,» *arXiv preprint arXiv:1907.06772*, 2019. dirección: https://arxiv.org/abs/1907.06772.

[16] Tactacam. «Tactacam REVEAL Cellular Cameras,» visitado 27 de dic. de 2024. dirección: https://www.revealcellcam.com/.

[17] Spypoint. «Spypoint Cellular Trail Cameras,» visitado 27 de dic. de 2024. dirección: https://www.spypoint.com/.

[18] Wildlife Insights. «Wildlife Insights: A cloud-based platform for camera trap data,» visitado 27 de dic. de 2024. dirección: https://www.wildlifeinsights.org/.

[19] World Wildlife Fund. «Atlantic Forest: The most threatened tropical forest,» visitado 27 de dic. de 2024. dirección: https://www.worldwildlife.org/places/atlantic-forest.

[20] M. C. Ribeiro, J. P. Metzger, A. C. Martensen, F. J. Ponzoni y M. M. Hirota, «The Brazilian Atlantic Forest: How much is left, and how is the remaining forest distributed? Implications for conservation,» *Biological Conservation*, vol. 142, n.º 6, págs. 1141-1153, 2009. DOI: 10.1016/j.biocon.2009.02.021.

[21] M. S. Di Bitetti, G. Placci y L. A. Dietz, «A Biodiversity Vision for the Upper Paraná Atlantic Forest Ecoregion: Designing a Biodiversity Conservation Landscape and Setting Priorities for Conservation Action,» *World Wildlife Fund*, 2003.

[22] Fundación Vida Silvestre Argentina. «Censo de Yaguareté 2024: Resultados del monitoreo binacional en el Corredor Verde,» visitado 27 de dic. de 2024. dirección: https://www.vidasilvestre.org.ar/.

[23] Congreso de la Nación Argentina, *Ley 26.331 de Presupuestos Mínimos de Protección Ambiental de los Bosques Nativos*, Boletín Oficial de la República Argentina, Sancionada el 28 de noviembre de 2007, 2007.

[24] Facultad de Agronomía, Universidad de Buenos Aires. «Estudio sobre pérdida de bosque nativo en el Corredor Verde de Misiones (1990-2020),» visitado 27 de dic. de 2024. dirección: https://www.agro.uba.ar/.

[25] Ministerio de Ecología y Recursos Naturales Renovables de Misiones. «Informe de Deforestación 2025: Misiones alcanzó el nivel más bajo de pérdida de bosque nativo,» visitado 27 de dic. de 2024. dirección: https://ecologia.misiones.gob.ar/.

[26] Ministerio de Ecología y Recursos Naturales Renovables de Misiones. «Programa de Control de Caza Furtiva: En Misiones NO se caza,» visitado 27 de dic. de 2024. dirección: https://ecologia.misiones.gob.ar/.

[27] T. E. Kucera y R. H. Barrett, «A History of Camera Trapping,» en *Camera Traps in Animal Ecology: Methods and Analyses*, A. F. O'Connell, J. D. Nichols y K. U. Karanth, eds., Tokyo: Springer, 2011, págs. 9-26. DOI: 10.1007/978-4-431-99495-4_2.

[28] Espressif Systems. «ESP-MESH Programming Guide,» visitado 15 de ene. de 2024. dirección: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/esp-wifi-mesh.html.

[29] M. S. Norouzzadeh et al., «Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning,» en *Proceedings of the National Academy of Sciences*, vol. 115, 2018, E5716-E5725.

[30] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016. dirección: http://www.deeplearningbook.org.

[31] Espressif Systems, *ESP32 Technical Reference Manual*, Espressif Systems, 2023. dirección: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

Apéndice A

Esquemáticos del hardware

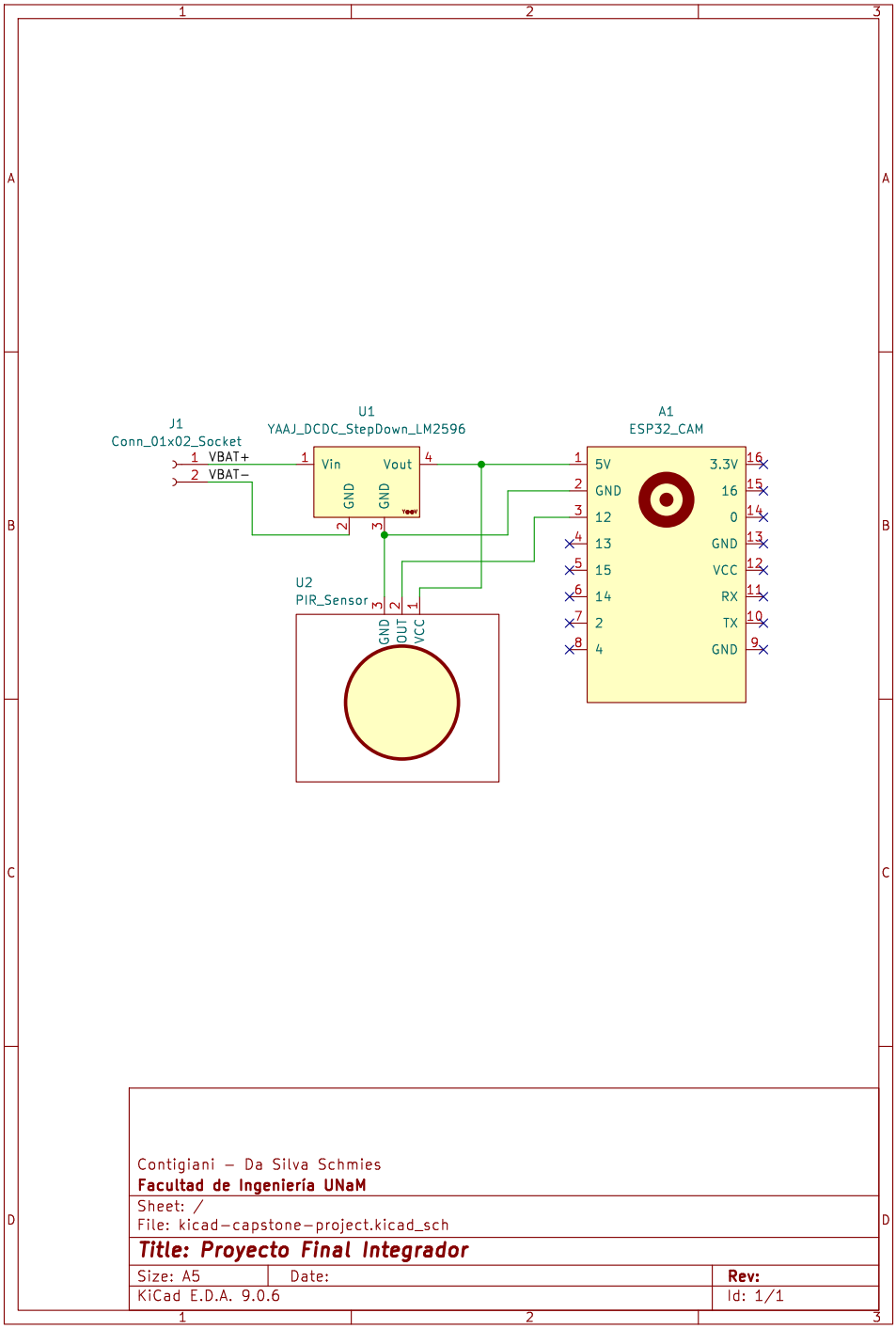

Figura A.1: Esquemático del nodo de cámara (mesh node).

Apéndice B

Planos de la carcasa

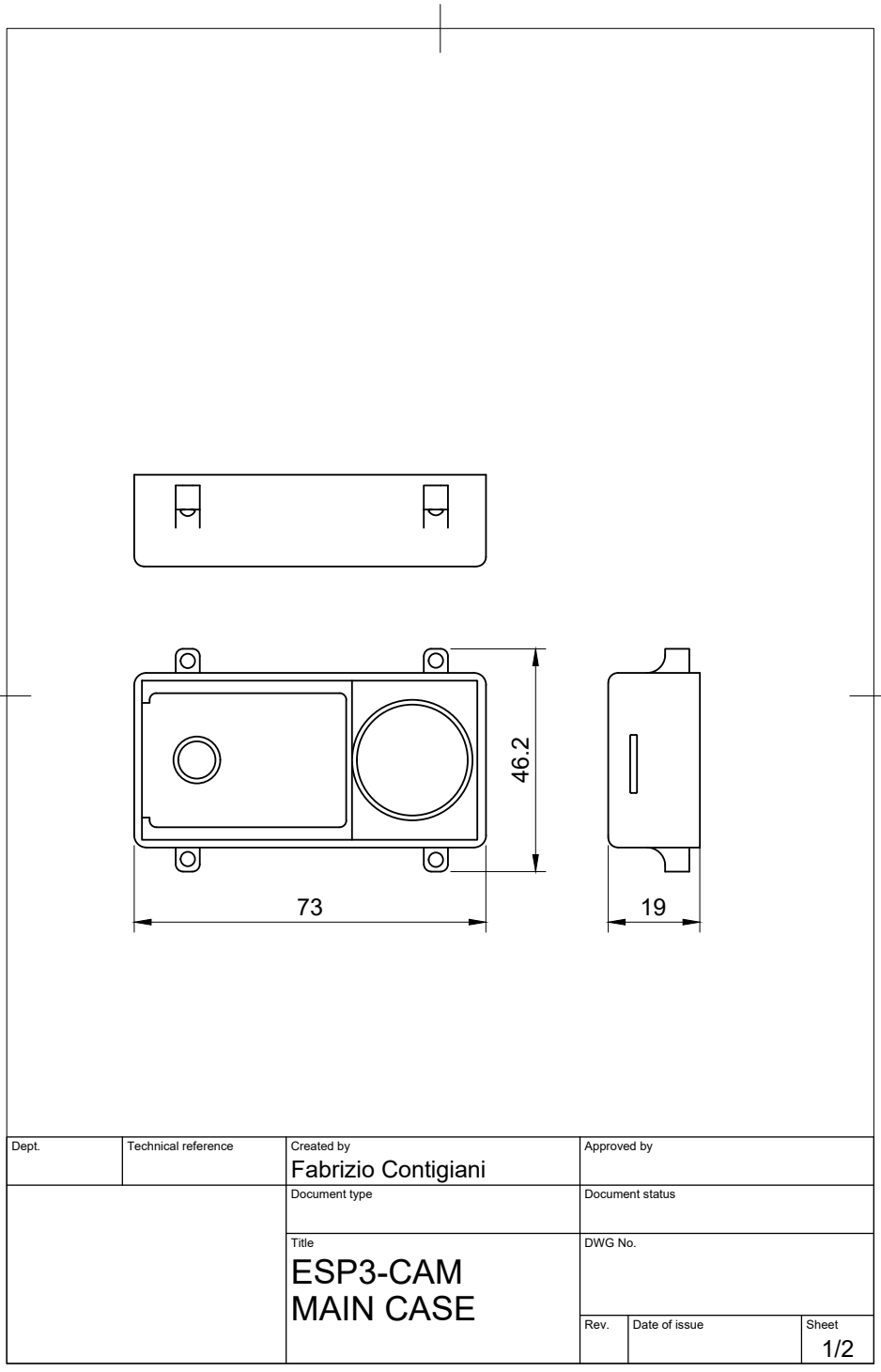

Figura B.1: Plano 2D de la carcasa principal del nodo de cámara.

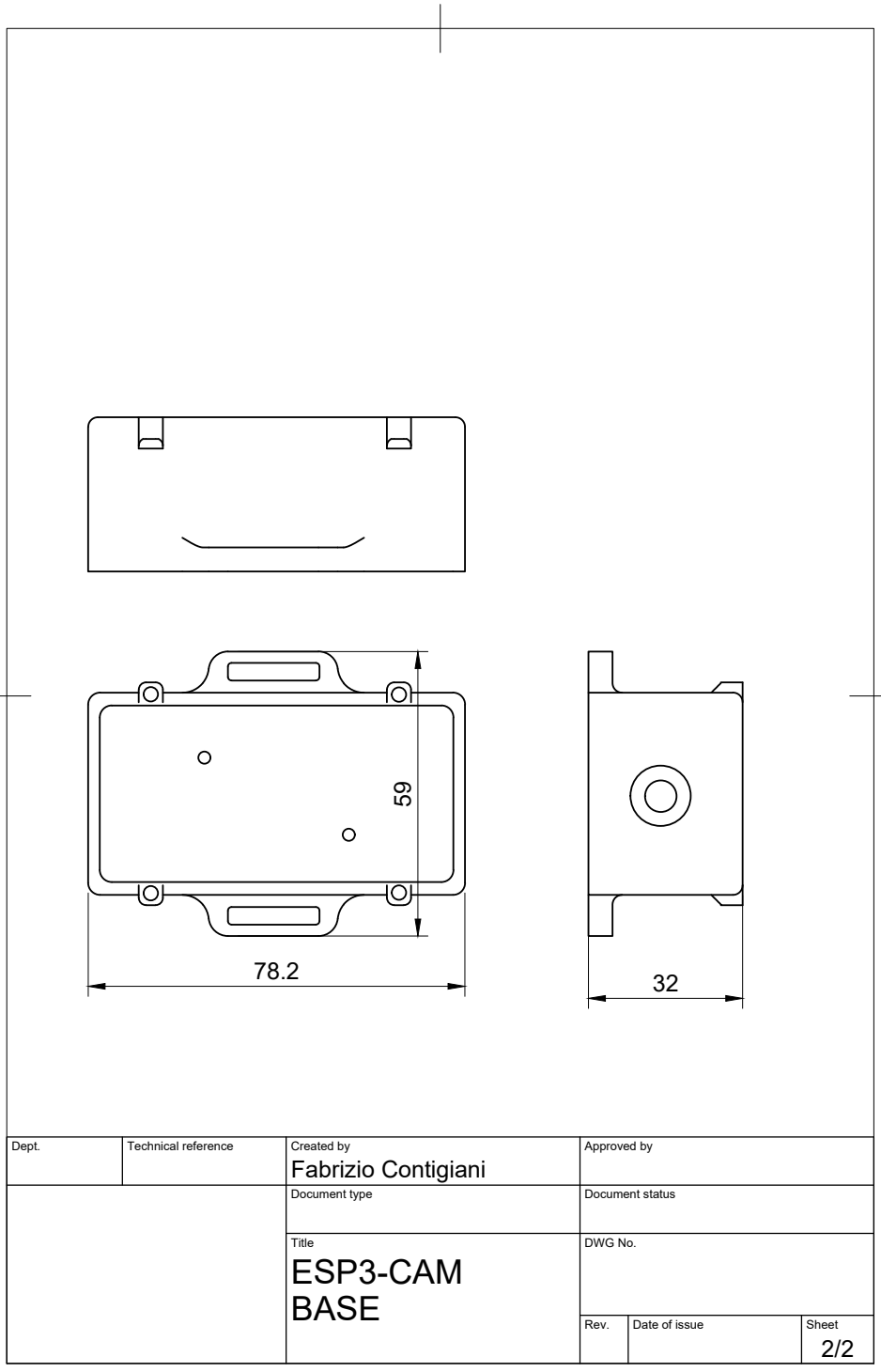

Figura B.2: Plano 2D de la base del nodo de cámara.

Apéndice C

Código fuente relevante

C.1. Firmware del nodo mesh

El archivo principal del firmware del nodo mesh (`mesh_node.c`) contiene la lógica de inicialización, captura de imágenes, detección de movimiento y comunicación con la red mesh. El código C.1 presenta la función principal `app_main()` que inicializa todos los componentes del sistema.

```
634 void app_main()
635 {
636     /* Initialize camera */
637     if (camera_is_supported())
638     {
639         if (camera_init() != ESP_OK)
640         {
641             ESP_LOGE(TAG, "Camera initialization failed, exiting");
642             return;
643         }
644     }
645     else
646     {
647         ESP_LOGW(TAG, "Camera not supported, continuing with SD card only");
648     }
649 
650     /* Initialize SD card for all nodes */
651     ESP_LOGI(TAG, "Initializing SD card...");
652     mdf_err_t sd_ret = sd_init();
653     if (sd_ret != ESP_OK)
654     {
655         ESP_LOGE(TAG, "SD card initialization failed: %s",
               esp_err_to_name(sd_ret));
656         ESP_LOGE(TAG, "Continuing without SD card functionality");
657     }
658     else
659     {
660         ESP_LOGI(TAG, "SD card initialized successfully");
661     }
662 
663     mwifi_init_config_t cfg = MWIFI_INIT_CONFIG_DEFAULT();
664     mwifi_config_t config = {
```

```
        .router_ssid = CONFIG_ROUTER_SSID,
        .router_password = CONFIG_ROUTER_PASSWORD,
        .mesh_id = CONFIG_MESH_ID,
        .mesh_password = CONFIG_MESH_PASSWORD,
        .mesh_type = MWIFI_MESH_NODE,
    };

    /**
     * @brief Set the log level for serial port printing.
     */
    esp_log_level_set("*", ESP_LOG_INFO);
    esp_log_level_set(TAG, ESP_LOG_DEBUG);

    /**
     * @brief Initialize wifi mesh.
     */
    MDF_ERROR_ASSERT(mdf_event_loop_init(event_loop_cb));
    MDF_ERROR_ASSERT(wifi_init());
    MDF_ERROR_ASSERT(mwifi_init(&cfg));
    MDF_ERROR_ASSERT(mwifi_set_config(&config));
    MDF_ERROR_ASSERT(mwifi_start());

    /**
     * @brief select/extend a group memebership here
     *      group id can be a custom address
     */
    const uint8_t group_id_list[2][6] = {{0x01, 0x00, 0x5e, 0xae, 0xae, 0xae},
                                         {0x01, 0x00, 0x5e, 0xae, 0xae, 0xaf}};

    MDF_ERROR_ASSERT(esp_mesh_set_group_id((mesh_addr_t *)group_id_list,
                                           sizeof(group_id_list) / sizeof(group_id_list[0])));

    ESP_LOGI(TAG, "Mesh initialization complete");

    TimerHandle_t timer = xTimerCreate("print_system_info", 10000 / portTICK_PERIOD_MS,
                                       true, NULL, print_system_info_timercb);
    xTimerStart(timer, 0);


    MDF_LOGI("Creating capture photo task...");
    xTaskCreate(capture_photo_task, "capture_photo_task", 4 * 1024, NULL, CONFIG_MDF_TASK_DEFAULT_PRIOTY, NULL);

    // Create motion detection task
    ESP_LOGI(TAG, "Creating motion detection task...");
    BaseType_t ret_task = xTaskCreate(motion_detection_task, "motion_detection", 2048, NULL, 5, &motion_detection_task_handle);
    if (ret_task != pdPASS)
    {
        ESP_LOGE(TAG, "Failed to create motion detection task");
        return;
```

```
714     }
715
716     // Configure GPIO 12 as input with pull-down and interrupt on rising
            edge
717     gpio_config_t io_conf = {
718         .intr_type = GPIO_INTR_POSEDGE,
719         .mode = GPIO_MODE_INPUT,
720         .pin_bit_mask = (1ULL << PIR_SENSOR_PIN),
721         .pull_down_en = 1,
722         .pull_up_en = 0,
723     };
724     gpio_config(&io_conf);
725
726     // Install GPIO ISR service and add handler
727     // gpio_install_isr_service(0); // already installed
728     gpio_isr_handler_add(PIR_SENSOR_PIN, gpio_isr_handler, NULL);
729
730 }
```

Código C.1: Función principal `app_main()` del nodo mesh.

El código C.2 muestra la función `capture_and_save_photo()`, responsable de capturar una imagen, guardarla en la tarjeta SD y transmitirla a través de la red mesh.

```
419 static esp_err_t capture_and_save_photo(int photo_counter)
420 {
421     if (!camera_is_supported())
422     {
423         ESP_LOGW(TAG, "Camera not supported on this platform");
424         return ESP_ERR_NOT_SUPPORTED;
425     }
426
427     camera_fb_t *frame_buffer = camera_capture_photo();
428     if (frame_buffer == NULL)
429     {
430         ESP_LOGE(TAG, "Failed to capture photo");
431         return ESP_FAIL;
432     }
433
434     /* Create unique filename with counter */
435     char photo_path[64];
436     snprintf(photo_path, sizeof(photo_path), "/sdcard/pic_%d.jpg",
            photo_counter);
437
438     ESP_LOGI(TAG, "Attempting to save photo to: %s", photo_path);
439
440     /* Save photo to SD card */
441     esp_err_t write_ret = s_example_write_file(photo_path,
            frame_buffer->buf, frame_buffer->len);
442     if (write_ret != ESP_OK)
443     {
444         ESP_LOGE(TAG, "Failed to write photo to SD card: %s",
                esp_err_to_name(write_ret));
445         camera_return_frame_buffer(frame_buffer);
446         return write_ret;
447     }
448
```

```
449     ESP_LOGI(TAG, "Photo saved to: %s (size: %d bytes)", photo_path,
            frame_buffer->len);
450
451     /* Send image over mesh (only if connected) */
452     if (mwifi_is_connected())
453     {
454         ESP_LOGI(TAG, "Sending image over mesh, size: %d bytes",
                frame_buffer->len);
455         mwifi_data_type_t data_type = {0x0};
456
457         if (esp_mesh_is_root())
458         {
459             ESP_LOGI(TAG, "Node is root, skipping mesh transmission");
460             camera_return_frame_buffer(frame_buffer);
461             return ESP_OK;
462         }
463
464         mdf_err_t mesh_ret = mwifi_write(NULL, &data_type,
                frame_buffer->buf, frame_buffer->len, true);
465         if (mesh_ret != MDF_OK)
466         {
467             ESP_LOGE(TAG, "Failed to send image over mesh: %s",
                    mdf_err_to_name(mesh_ret));
468         }
469         else
470         {
471             ESP_LOGI(TAG, "Image sent over mesh successfully!");
472         }
473     }
474     else
475     {
476         ESP_LOGW(TAG, "Mesh not connected, skipping mesh transmission");
477     }
478
479     /* Return the frame buffer */
480     camera_return_frame_buffer(frame_buffer);
481
482     return ESP_OK;
483 }
```

Código C.2: Función de captura y envío de imágenes.

C.2. Firmware del nodo raíz

El nodo raíz actúa como puerta de enlace entre la red mesh y el servidor remoto. Su firmware (`root_node.c`) implementa la recepción de imágenes desde los nodos de cámara y su transmisión al servidor mediante HTTP POST.

El código C.3 muestra la función `app_main()` del nodo raíz, que configura la red mesh en modo raíz fijo e inicializa las tareas de recepción y transmisión de datos.

```
457 void app_main() {
458   mwifi_init_config_t cfg = MWIFI_INIT_CONFIG_DEFAULT();
459   mwifi_config_t config = {
460       .router_ssid = CONFIG_ROUTER_SSID,
```

```
    .router_password = CONFIG_ROUTER_PASSWORD,
    .mesh_id = CONFIG_MESH_ID,
    .mesh_password = CONFIG_MESH_PASSWORD,
    .mesh_type = MWIFI_MESH_ROOT,
};

/**
 * @brief Set the log level for serial port printing.
 */
esp_log_level_set("*", ESP_LOG_INFO);
esp_log_level_set(TAG, ESP_LOG_DEBUG);

/**
 * @brief Initialize wifi mesh.
 */
MDF_ERROR_ASSERT(mdf_event_loop_init(event_loop_cb));
MDF_ERROR_ASSERT(wifi_init());
MDF_ERROR_ASSERT(mwifi_init(&cfg));
MDF_ERROR_ASSERT(mwifi_set_config(&config));
MDF_ERROR_ASSERT(esp_mesh_fix_root(true));
MDF_ERROR_ASSERT(mwifi_start());

/**
 * @brief select/extend a group memebership here
 *      group id can be a custom address
 */
const uint8_t group_id_list[2][6] = {{0x01, 0x00, 0x5e, 0xae, 0xae, 0xae},
                                     {0x01, 0x00, 0x5e, 0xae, 0xae,
                                        0xaf}};

MDF_ERROR_ASSERT(
    esp_mesh_set_group_id((mesh_addr_t *)group_id_list,
                          sizeof(group_id_list) /
                             sizeof(group_id_list[0])));

ESP_LOGI(TAG, "Mesh initialization complete. Waiting for root to get
   IP...");

TimerHandle_t timer =
    xTimerCreate("print_system_info", 10000 / portTICK_PERIOD_MS, true,
       NULL,
                 print_system_info_timercb);
xTimerStart(timer, 0);

// Create HTTP queue and task
ESP_LOGI(TAG, "Creating HTTP queue and task...");
http_queue = xQueueCreate(HTTP_QUEUE_SIZE, sizeof(received_data_t));
if (http_queue == NULL) {
  ESP_LOGE(TAG, "Failed to create HTTP queue");
  return;
}

BaseType_t http_task = xTaskCreate(http_post_task, "http_post_task",
   12288,
                                   NULL, 4, &http_task_handle);
if (http_task != pdPASS) {
```

```
512     ESP_LOGE(TAG, "Failed to create HTTP POST task");
513     vQueueDelete(http_queue);
514     return;
515   }
516   ESP_LOGI(TAG, "HTTP queue and task created successfully");
517
518   MDF_LOGI("Creating root task...");
519   xTaskCreate(root_task, "root_task", 4 * 1024, NULL,
520               CONFIG_MDF_TASK_DEFAULT_PRIOTY, NULL);
521 }
```

Código C.3: Función principal `app_main()` del nodo raíz.

El código C.4 presenta la función `root_task()`, que implementa el bucle principal de recepción de datos desde la red mesh. Esta tarea espera hasta que el nodo se establezca como raíz, luego recibe datos de los nodos hijos mediante `mwifi_root_read()` y los encola para su transmisión HTTP.

```
215 static void root_task(void *arg) {
216   mdf_err_t ret = MDF_OK;
217   // Use 64KB buffer to handle larger image chunks or accumulated data
218   size_t buffer_capacity = DATA_BUFFER_SIZE;
219   uint8_t *data = MDF_CALLOC(1, buffer_capacity);
220   if (data == NULL) {
221     ESP_LOGW(
222         TAG,
223         "Failed to allocate %d-byte data buffer, trying MWIFI payload size",
224         DATA_BUFFER_SIZE);
225     buffer_capacity = MWIFI_PAYLOAD_LEN;
226     data = MDF_CALLOC(1, buffer_capacity);
227     if (data == NULL) {
228       ESP_LOGE(TAG, "Failed to allocate data buffer (%zu bytes)",
229                buffer_capacity);
230       ESP_LOGE(TAG, "Free heap: %u bytes, largest free block: %u bytes",
231                esp_get_free_heap_size(),
232                heap_caps_get_largest_free_block(MALLOC_CAP_8BIT));
233       vTaskDelete(NULL);
234       return;
235     }
236   }
237   size_t size = buffer_capacity;
238
239   uint8_t src_addr[MWIFI_ADDR_LEN] = {0x0};
240   mwifi_data_type_t data_type = {0};
241
242   MDF_LOGI("Root task is running");
243
244   // Wait until node is actually root before attempting to read
245   while (!esp_mesh_is_root()) {
246     ESP_LOGI(TAG, "Waiting to become root node...");
247     vTaskDelay(1000 / portTICK_PERIOD_MS);
248
249     if (!mwifi_is_started()) {
250       ESP_LOGW(TAG, "Mesh not started, waiting...");
251       vTaskDelay(1000 / portTICK_PERIOD_MS);
252       continue;
```

```
    }
  }

  ESP_LOGI(TAG, "Node is now root, starting to read data");

  for (;;) {
    if (!mwifi_is_started()) {
      vTaskDelay(500 / portTICK_PERIOD_MS);
      continue;
    }

    // Double-check we're still root before attempting read
    if (!esp_mesh_is_root()) {
      ESP_LOGW(TAG, "Node is no longer root, waiting to become root
          again...");
      while (!esp_mesh_is_root()) {
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        if (!mwifi_is_started()) {
          break;
        }
      }
      ESP_LOGI(TAG, "Node is root again, resuming data reading");
      continue;
    }

    size = buffer_capacity;
    memset(data, 0, buffer_capacity);
    ret = mwifi_root_read(src_addr, &data_type, data, &size, portMAX_DELAY);
    MDF_ERROR_CONTINUE(ret != MDF_OK, "<%s> mwifi_root_read",
                       mdf_err_to_name(ret));
    MDF_LOGI("Root receive, addr: " MACSTR ", size: %d", MAC2STR(src_addr),
             size);

    // Validate received data size
    if (size == 0) {
      ESP_LOGW(TAG, "Received empty data, skipping save");
      continue;
    }

    // Prepare data for storage task
    received_data_t received_item;
    memcpy(received_item.src_addr, src_addr, MWIFI_ADDR_LEN);
    received_item.data_type = data_type;
    received_item.size = size;

    // Allocate memory for the data copy in internal memory (for large image
    // data)
    received_item.data = MDF_MALLOC(size);
    if (received_item.data == NULL) {
      ESP_LOGE(TAG, "Failed to allocate memory for received data (%zu
          bytes)",
               size);
      ESP_LOGE(TAG, "Free heap: %u bytes, largest free block: %u bytes",
               esp_get_free_heap_size(),
               heap_caps_get_largest_free_block(MALLOC_CAP_8BIT));
      continue;
```

```
307     }
308     memcpy(received_item.data, data, size);
309
310     // Queue the data for HTTP POST task
311     if (http_queue != NULL) {
312       if (xQueueSend(http_queue, &received_item, pdMS_TO_TICKS(1000)) !=
313           pdTRUE) {
314         ESP_LOGE(TAG,
315                  "Failed to queue received data for HTTP POST (queue
                     full?)");
316         MDF_FREE(received_item.data); // Free memory if queueing failed
317       } else {
318         ESP_LOGI(TAG, "Queued received data for HTTP POST");
319       }
320     } else {
321       ESP_LOGE(TAG, "HTTP queue not initialized");
322       MDF_FREE(received_item.data);
323     }
324
325     // size = sprintf(data, "(%d) Hello node!", i);
326     // ret = mwifi_root_write(src_addr, 1, &data_type, data, size, true);
327     // MDF_ERROR_CONTINUE(ret != MDF_OK, "mwifi_root_recv, ret: %x", ret);
328     // MDF_LOGI("Root send, addr: " MACSTR ", size: %d, data: %s",
329     // MAC2STR(src_addr), size, data);
330   }
331
332   MDF_LOGW("Root is exit");
333
334   MDF_FREE(data);
335   vTaskDelete(NULL);
336 }
```

Código C.4: Tarea de recepción de datos de la red mesh.

El código C.5 muestra la tarea `http_post_task()`, responsable de enviar las imágenes recibidas al servidor Django mediante solicitudes HTTP POST con formato multipart. La tarea construye el cuerpo de la solicitud incluyendo la dirección MAC del nodo origen y los datos de la imagen.

```
41 /**
42  * @brief Task to handle HTTP POST of received data
43  */
44 static void http_post_task(void *arg) {
45   ESP_LOGI(TAG, "HTTP POST task started");
46   received_data_t received_item;
47
48   for (;;) {
49     // Wait for data to be queued
50     if (xQueueReceive(http_queue, &received_item, portMAX_DELAY) == pdTRUE)
         {
51       ESP_LOGI(TAG, "Processing received data from: " MACSTR " (%d bytes)",
52                MAC2STR(received_item.src_addr), received_item.size);
53
54       // Validate received data
55       if (received_item.data == NULL || received_item.size == 0) {
56         ESP_LOGW(TAG, "Invalid received data, skipping HTTP POST");
```

```
        if (received_item.data != NULL) {
          MDF_FREE(received_item.data);
        }
        continue;
      }

      // Generate filename for the upload
      char filename[64];
      snprintf(filename, sizeof(filename),
               "received_%02x%02x%02x%02x%02x%02x_%lu.jpg",
               received_item.src_addr[0], received_item.src_addr[1],
               received_item.src_addr[2], received_item.src_addr[3],
               received_item.src_addr[4], received_item.src_addr[5],
               (unsigned long)(esp_timer_get_time() / 1000));

      ESP_LOGI(TAG, "Sending received data via HTTP POST: %s (%d bytes)",
               filename, received_item.size);
      ESP_LOGI(TAG, "HTTP Server URL: %s", HTTP_SERVER_URL);

      // Get MAC address of this device (root node)
      uint8_t root_mac[6];
      esp_wifi_get_mac(ESP_IF_WIFI_STA, root_mac);

      // Try using esp_http_client_perform with pre-built data
      // Create the complete multipart body in memory first
      char boundary[] = "----WebKitFormBoundary7MA4YWxkTrZu0gW";

      char form_start[512];
      char form_end[128];

      snprintf(
          form_start, sizeof(form_start),
          "--%s\r\n"
          "Content-Disposition: form-data; name=\"mac_address\"\r\n\r\n"
          "%02x:%02x:%02x:%02x:%02x:%02x\r\n"
          "--%s\r\n"
          "Content-Disposition: form-data; name=\"image\";
             filename=\"%s\"\r\n"
          "Content-Type: image/jpeg\r\n\r\n",
          boundary, root_mac[0], root_mac[1], root_mac[2], root_mac[3],
          root_mac[4], root_mac[5], boundary, filename);
      snprintf(form_end, sizeof(form_end), "\r\n--%s--\r\n", boundary);

      int start_len = strlen(form_start);
      int end_len = strlen(form_end);
      int total_body_len = start_len + received_item.size + end_len;

      // Allocate memory for complete body in heap
      uint8_t *complete_body = MDF_MALLOC(total_body_len);
      if (complete_body == NULL) {
        ESP_LOGE(TAG, "Failed to allocate memory for HTTP body (%d bytes)",
                 total_body_len);
        ESP_LOGE(TAG, "Free heap: %u bytes, largest free block: %u bytes",
                 esp_get_free_heap_size(),
                 heap_caps_get_largest_free_block(MALLOC_CAP_8BIT));
        MDF_FREE(received_item.data);
```

```
112         continue;
113       }
114
115       // Build complete multipart body
116       memcpy(complete_body, form_start, start_len);
117       memcpy(complete_body + start_len, received_item.data,
              received_item.size);
118       memcpy(complete_body + start_len + received_item.size, form_end,
              end_len);
119
120       ESP_LOGI(TAG, "Built complete multipart body: %d bytes",
              total_body_len);
121       ESP_LOGI(TAG, "Form structure preview: %.100s", (char
              *)complete_body);
122
123       // Configure HTTP client with complete body
124       esp_http_client_config_t config = {
125           .url = HTTP_SERVER_URL,
126           .method = HTTP_METHOD_POST,
127           .timeout_ms = HTTP_TIMEOUT_MS,
128           .max_redirection_count = 5,
129           .disable_auto_redirect = false,
130       };
131
132       esp_http_client_handle_t client = esp_http_client_init(&config);
133       if (client == NULL) {
134         ESP_LOGE(TAG, "Failed to initialize HTTP client");
135         MDF_FREE(received_item.data);
136         MDF_FREE(complete_body);
137         continue;
138       }
139
140       // Set content type header
141       char content_type[128];
142       snprintf(content_type, sizeof(content_type),
143                "multipart/form-data; boundary=%s", boundary);
144       esp_http_client_set_header(client, "Content-Type", content_type);
145
146       // Set the complete body
147       esp_http_client_set_post_field(client, (char *)complete_body,
148                                      total_body_len);
149
150       // Perform the request
151       esp_err_t err = esp_http_client_perform(client);
152
153       if (err == ESP_OK) {
154         ESP_LOGI(TAG, "HTTP request completed successfully");
155       } else {
156         ESP_LOGE(TAG, "HTTP request failed: %s", esp_err_to_name(err));
157       }
158
159       // Get response details
160       int status_code = esp_http_client_get_status_code(client);
161       int content_length = esp_http_client_get_content_length(client);
162
163       ESP_LOGI(TAG, "HTTP POST completed - Status: %d, Content-Length: %d",
```

```
164                 status_code, content_length);
165
166       if (status_code >= 200 && status_code < 300) {
167         ESP_LOGI(TAG, "Successfully uploaded image: %s", filename);
168       } else if (status_code >= 300 && status_code < 400) {
169         // Handle redirect responses
170         ESP_LOGW(TAG, "HTTP redirect response: %d", status_code);
171
172         // Get the Location header for debugging
173         // char location_buffer[256];
174         // int location_len = esp_http_client_get_header(client, "Location",
175         // location_buffer, sizeof(location_buffer) - 1); if (location_len
                > 0)
176         // {
177         //     location_buffer[location_len] = '\0';
178         //     ESP_LOGW(TAG, "Redirect location: %s", location_buffer);
179         // }
180
181         ESP_LOGE(TAG, "HTTP upload failed due to redirect: %d",
                status_code);
182       } else {
183         ESP_LOGE(TAG, "HTTP upload failed with status: %d", status_code);
184
185         // Read error response if available
186         if (content_length > 0 && content_length < 1024) {
187           char response_buffer[1024];
188           int read_len = esp_http_client_read_response(
189               client, response_buffer, sizeof(response_buffer) - 1);
190           if (read_len > 0) {
191             response_buffer[read_len] = '\0';
192             ESP_LOGE(TAG, "Server response: %s", response_buffer);
193           }
194         }
195       }
196
197       // Cleanup
198       esp_http_client_cleanup(client);
199       MDF_FREE(complete_body);
200       MDF_FREE(received_item.data);
201     }
202   }
203
204   vTaskDelete(NULL);
205 }
```

Código C.5: Tarea de envío HTTP al servidor.

C.3. Servidor de detección

El servidor de detección implementa una versión extendida del API de SpeciesNet que, además de realizar la inferencia, dibuja los cuadros delimitadores sobre las detecciones y guarda las imágenes anotadas. El código está contenido en `main.py` del repositorio `wildlife-detection-capstone`.

El código C.6 presenta la función `save_annotated_image()`, que utiliza la biblioteca PIL

para dibujar los cuadros delimitadores sobre la imagen original y guardarla con el sufijo `_annotated`.

```python
def save_annotated_image(filepath: str, detections: list[dict]) -> Optional[str]:
    """Draw bounding boxes and save annotated image alongside original.

    Args:
        filepath: Path to the original image.
        detections: List of detection dictionaries with bbox info.

    Returns:
        Path to the saved annotated image, or None if no detections.
    """
    if not detections:
        return None

    try:
        img = Image.open(filepath).convert("RGB")
        annotated_img = draw_bboxes(img, detections)
        # Convert RGBA back to RGB for JPEG saving
        annotated_img = annotated_img.convert("RGB")

        # Generate annotated filename
        path = Path(filepath)
        annotated_path = path.parent / f"{path.stem}_annotated{path.suffix}"

        # Save with high quality
        annotated_img.save(str(annotated_path), quality=90)

        return str(annotated_path)
    except Exception as e:
        print(f"Error saving annotated image for {filepath}: {e}")
        return None
```

Código C.6: Función para guardar imágenes anotadas con detecciones.

El código C.7 muestra la clase `AnnotatingSpeciesNetAPI`, que extiende el servidor estándar de SpeciesNet para incluir la funcionalidad de anotación de imágenes. El método `predict()` procesa las imágenes, ejecuta la inferencia y opcionalmente guarda las versiones anotadas.

```python
class AnnotatingSpeciesNetAPI(ls.LitAPI):
    """Extended SpeciesNet API that saves annotated images.

    This class extends the standard SpeciesNet server to also draw bounding
    boxes on detected objects and save the annotated images alongside the
    original files.
    """

    def __init__(
        self,
        model_name: str,
        geofence: bool = True,
        extra_fields: Optional[list[str]] = None,
        save_annotated: bool = True,
        api_path: str = "/predict",
    ) -> None:
```

```
        """Initializes the annotating SpeciesNet API server.

        Args:
            model_name:
                String value identifying the model to be loaded.
            geofence:
                Whether to enable geofencing or not. Defaults to `True`.
            extra_fields:
                Comma-separated list of extra fields to propagate.
            save_annotated:
                Whether to save annotated images. Defaults to `True`.
            api_path:
                URL path for the server endpoint. Defaults to `/predict`.
        """
        super().__init__()
        self.api_path = api_path
        self.model_name = model_name
        self.geofence = geofence
        self.extra_fields = extra_fields or []
        self.save_annotated = save_annotated

    def setup(self, device):
        del device  # Unused.
        self.model = SpeciesNet(self.model_name, geofence=self.geofence)

    def decode_request(self, request, context):
        del context  # Unused.
        for instance in request["instances"]:
            filepath = instance["filepath"]
            if not file_exists(filepath):
                raise HTTPException(400, f"Cannot access filepath:
                    `{filepath}`")
        return request

    def _propagate_extra_fields(
        self, instances_dict: dict, predictions_dict: dict
    ) -> dict:
        predictions = predictions_dict["predictions"]
        new_predictions = {p["filepath"]: p for p in predictions}
        for instance in instances_dict["instances"]:
            for field in self.extra_fields:
                if field in instance:
                    new_predictions[instance["filepath"]][field] =
                        instance[field]
        return {"predictions": list(new_predictions.values())}

    def predict(self, instances_dict, context):
        del context  # Unused.
        predictions_dict = self.model.predict(instances_dict=instances_dict)
        assert predictions_dict is not None

        result = self._propagate_extra_fields(instances_dict,
            predictions_dict)

        # Draw and save annotated images if enabled
        if self.save_annotated:
```

```
            for prediction in result["predictions"]:
                filepath = prediction["filepath"]
                detections = prediction.get("detections", [])
                annotated_path = save_annotated_image(filepath, detections)
                prediction["annotated_filepath"] = annotated_path

        return result

    def encode_response(self, output, context):
        del context  # Unused.
        return output
```

Código C.7: Clase del servidor de inferencia con anotación de imágenes.

C.4. Aplicación Django

La aplicación Django coordina el procesamiento de imágenes y el envío de notificaciones. El módulo `service.py` contiene la lógica de orquestación entre la recepción de imágenes, el servicio de inferencia y el sistema de alertas.

El código C.8 muestra la función `process_image()`, que envía la imagen al servidor SpeciesNet, procesa los resultados de la inferencia, guarda los metadatos de clasificación y dispara el envío de notificaciones a través de Telegram.

```
def process_image(instance: MyImage) -> None:
    """Process an image through SpeciesNet for detection and classification.

    The external SpeciesNet service draws bounding boxes and saves the annotated
    image alongside the original. This function reads that annotated image path
    from the prediction response.
    """
    prediction = run_inference(instance.image.path)

    if "error" in prediction:
        logger.error(
            f"Inference failed for {instance.image.path}: {prediction['error']}"
        )
        instance.metadata = {"error": prediction["error"]}
        instance.save(update_fields=["metadata"])
        return

    # Extract classification results
    top_classifications = format_classifications(prediction)

    # Get the annotated image path from the prediction response
    annotated_filepath = prediction.get("annotated_filepath")

    if annotated_filepath and os.path.exists(annotated_filepath):
        # Copy the annotated image to the processed_images directory
        with open(annotated_filepath, "rb") as f:
            filename = f"processed_{Path(instance.image.name).name}"
```

```
            instance.processed_image.save(filename, File(f), save=False)
        # Remove the duplicate from the original location
        os.remove(annotated_filepath)

    # Save full prediction metadata including classifications
    instance.metadata = {
        "predictions": prediction,
        "top_classifications": top_classifications,
        "detections_count": len(prediction.get("detections", [])),
    }
    instance.save(update_fields=["processed_image", "metadata"])

    logger.info(
        f"Processed image {instance.image.path}:
           {len(prediction.get('detections', []))} detections"
    )

    # Send Telegram notification
    send_telegram_notification(instance)
```

Código C.8: Función de procesamiento de imágenes.

El código C.9 presenta la función `send_telegram_notification()`, que lee los datos de la imagen procesada y los transmite a todos los usuarios de Telegram registrados en el sistema. La función envía tanto la imagen original como la versión anotada con las detecciones, junto con un reporte de texto formateado con los resultados de la clasificación.

```
def send_telegram_notification(instance):
    """
    Lee los datos de la imagen y los transmite a todos los usuarios de
       Telegram registrados.
    """
    try:
        # 1. Obtener destinatarios (usuarios que han iniciado el bot)
        chat_ids = list(TelegramUser.objects.values_list("chat_id",
           flat=True))
        if not chat_ids:
            return

        logger.warning(
            f"Preparing to send notification to {len(chat_ids)} users:
               {chat_ids}"
        )

        # 2. Preparar datos (leer archivos en memoria para evitar problemas
           de I/O en async)
        img_path = instance.image.path
        if not os.path.exists(img_path):
            logger.warning(f"Image file not found: {img_path}")
            return

        with open(img_path, "rb") as f:
            original_bytes = f.read()

        processed_bytes = None
        if instance.processed_image and
```

```
            os.path.exists(instance.processed_image.path):
78              with open(instance.processed_image.path, "rb") as f:
79                  processed_bytes = f.read()
80
81          # 3. Formatear texto usando la utilidad existente
82          metadata = instance.metadata or {}
83          text_report = format_classification_results(metadata)
84
85          # 4. Execute async send from sync context
86          async_to_sync(_broadcast_results)(
87              chat_ids, original_bytes, processed_bytes, text_report,
                    instance.created_at
88          )
89          logger.info(f"Sent Telegram notification to {len(chat_ids)} users.")
90
91      except Exception as e:
92          logger.error(f"Error in send_telegram_notification: {e}")
```

Código C.9: Función de envío de notificaciones por Telegram.

Apéndice D

Manual de instalación y uso

Este capítulo presenta las instrucciones detalladas para la configuración e instalación del sistema completo, incluyendo el firmware de los nodos, el servidor de aplicación y el bot de Telegram.

D.1. Configuración del firmware

La configuración del firmware para los nodos mesh y el nodo raíz se realiza mediante el menú de configuración de ESP-IDF. Los parámetros principales se encuentran bajo el submenú "Example Configuration" accesible mediante el comando `idf.py menuconfig`.

D.1.1. Parámetros de la red Wi-Fi

Los siguientes parámetros deben configurarse para conectar la red mesh a un router externo que proporcione acceso a Internet:

Router SSID: Nombre de la red Wi-Fi del router.

Router Password: Contraseña de la red Wi-Fi.

Router Channel: Canal de operación del router. Si se desconoce, configurar como 0 para detección automática.

D.1.2. Parámetros de la red ESP-MESH

La red mesh se configura mediante los siguientes parámetros:

Mesh ID: Identificador único de la red mesh (6 bytes).

Mesh Password: Contraseña de la red mesh (entre 8 y 64 caracteres). Si se deja en blanco, la red no estará encriptada.

Max Connections: Número máximo de conexiones que puede aceptar cada nodo (por defecto, 6).

D.1.3. Parámetros del servidor TCP

Para que el nodo raíz pueda conectarse al servidor de aplicación, se deben configurar:

Server IP: Dirección IP del servidor donde se ejecuta la aplicación Django.

Server Port: Puerto TCP del servidor (por defecto, 8001).

D.1.4. Compilación y grabación

Una vez configurados los parámetros, el firmware se compila y graba utilizando los siguientes comandos:

```
# Borrar flash y grabar firmware
idf.py erase_flash flash monitor -b \num{921600} -p /dev/ttyUSB0
```

Código D.1: Comandos para compilar y grabar el firmware.

Es recomendable grabar primero el nodo raíz y verificar su conexión al router antes de grabar los nodos de cámara.

D.2. Despliegue del servidor

El servidor de aplicación se despliega utilizando Docker y Docker Compose, lo que simplifica la gestión de dependencias y la configuración del entorno.

D.2.1. Prerrequisitos

- Docker y Docker Compose instalados en el servidor.
- Token del bot de Telegram (obtenido de @BotFather).
- Opcional: GPU NVIDIA con nvidia-container-toolkit para acelerar la inferencia.

D.2.2. Configuración del entorno

1. Clonar el repositorio del servidor.
2. Copiar el archivo de configuración de ejemplo: `cp .env.sample .env`
3. Editar el archivo `.env` con los valores apropiados.

Las variables de entorno principales son:

DJANGO_SECRET_KEY: Clave secreta de Django para producción.

POSTGRES_DB, POSTGRES_USER, POSTGRES_PASSWORD: Credenciales de la base de datos PostgreSQL.

TELEGRAM_BOT_TOKEN: Token del bot de Telegram (requerido).

GUNICORN_WORKERS: Número de workers de Gunicorn (por defecto, 2).

D.2.3. Inicio de los servicios

El sistema completo se inicia con un único comando:

```
docker compose up --build
```

Código D.2: Inicio de los servicios con Docker Compose.

Este comando inicia cuatro servicios:

web: Servidor de aplicación Django (puerto 8000).

bot: Bot de Telegram para notificaciones.

speciesnet: Servicio de inferencia de SpeciesNet (puerto 8002).

db: Base de datos PostgreSQL.

Nota: La primera ejecución puede tardar entre 5 y 10 minutos mientras se descarga el modelo de SpeciesNet ($\sim 2\,\text{GB}$). Las ejecuciones posteriores son significativamente más rápidas gracias al caché del modelo en un volumen de Docker.

D.2.4. Aceleración por GPU

Para habilitar la aceleración por GPU en el servicio de SpeciesNet:

1. Instalar nvidia-container-toolkit en el servidor.
2. Descomentar la sección `deploy` del servicio `speciesnet` en `docker-compose.yml`.
3. Reiniciar los servicios.

Con GPU, el tiempo de inferencia se reduce de 1 s a 5 s por imagen (CPU) a 0,5 s a 1 s (GPU).

D.3. Configuración del bot de Telegram

El bot de Telegram permite recibir notificaciones en tiempo real y consultar las últimas imágenes procesadas desde cualquier dispositivo móvil.

D.3.1. Creación del bot

1. Abrir Telegram y buscar @BotFather.
2. Enviar el comando `/newbot` y seguir las instrucciones.
3. Copiar el token proporcionado por BotFather.
4. Configurar el token en la variable `TELEGRAM_BOT_TOKEN` del archivo `.env`.

D.3.2. Registro de usuarios

Para recibir notificaciones, los usuarios deben registrarse enviando el comando `/start` al bot. Esto crea un registro en la base de datos que asocia el ID de Telegram del usuario con el sistema.

D.3.3. Comandos disponibles

El bot soporta los siguientes comandos:

/start: Registra al usuario y muestra el mensaje de bienvenida.

/last: Recupera la última imagen procesada, mostrando:

- Fotografía original.
- Fotografía procesada con cuadros delimitadores.
- Lista de detecciones con niveles de confianza.
- Top 5 de clasificaciones de especies.

D.4. Uso del sistema

Una vez desplegado el sistema, el flujo de operación es completamente automático. A continuación se describen las distintas formas de interactuar con el sistema.

D.4.1. Interfaz web

La interfaz web está disponible en `http://<servidor>:8000` y permite:

1. Visualizar las imágenes recibidas de los nodos.
2. Ver los resultados de la detección y clasificación.
3. Acceder a las imágenes anotadas con cuadros delimitadores.

D.4.2. Notificaciones automáticas

Cuando un nodo de cámara detecta movimiento y captura una imagen:

1. La imagen se transmite a través de la red mesh hasta el nodo raíz.
2. El nodo raíz reenvía la imagen al servidor vía TCP.
3. El servidor procesa la imagen con SpeciesNet.
4. Si se detectan animales, personas o vehículos, se envía una notificación a todos los usuarios registrados en Telegram.

D.4.3. Consultas bajo demanda

Los usuarios pueden consultar el estado del sistema en cualquier momento utilizando el comando `/last` del bot de Telegram, que devuelve la última imagen procesada junto con su análisis.

D.4.4. Acceso directo a la API

Para integraciones avanzadas, es posible realizar consultas directas al servicio de SpeciesNet:

```
curl -X POST http://localhost:\num{8002}/predict \
  -H "Content-Type: application/json" \
  -d '{"instances": [{"filepath": "/app/media/images/foto.jpg"}]}'
```

Código D.3: Ejemplo de consulta directa a la API de SpeciesNet.

La respuesta incluye:

detections: Lista de objetos detectados con etiquetas (animal/humano/vehículo) y niveles de confianza.

classifications: Predicciones de especies con niveles de confianza.

annotated_filepath: Ruta a la imagen anotada con cuadros delimitadores.

D.4.5. Solución de problemas comunes

SpeciesNet no responde: Verificar el estado del contenedor con `docker compose logs speciesnet`. La descarga del modelo puede tardar varios minutos en la primera ejecución.

Bot de Telegram sin respuesta: Verificar que el token esté configurado correctamente en `.env` y revisar los logs con `docker compose logs bot`.

Imágenes no procesadas: Verificar que todos los servicios estén en ejecución con `docker compose ps` y revisar los logs del servicio web.

Apéndice E

Análisis de viabilidad económica

El presente anexo desarrolla un análisis de viabilidad económica del sistema propuesto, evaluando su potencial como emprendimiento comercial para la provisión de servicios de monitoreo de fauna silvestre y detección de intrusos.

E.1. Modelo de negocio

El modelo de negocio propuesto se basa en dos fuentes de ingreso:

Cobro por instalación: Un pago inicial por parte del cliente que cubre el costo del hardware (nodos de cámara y nodo raíz), la instalación física del sistema en el sitio del cliente, y la configuración inicial del servicio.

Suscripción mensual: Un cobro recurrente que incluye el acceso al servicio de procesamiento de imágenes con IA, el almacenamiento de imágenes en la nube, las notificaciones vía bot de Telegram, y el soporte técnico.

Este modelo permite recuperar los costos de hardware en el momento de la venta, mientras que la suscripción genera ingresos recurrentes que aumentan con cada nuevo cliente, creando un flujo de caja predecible y creciente.

E.2. Segmentación de mercado

El mercado objetivo se compone de dos segmentos principales en la provincia de Misiones, Argentina, como se detalla en la Tabla E.1.

La proyección considera alcanzar 28 nuevos clientes por año durante los primeros 5 años de operación, llegando a 140 clientes activos al final del período analizado.

E.3. Costos fijos

Los costos fijos mensuales contemplan los gastos operativos necesarios para mantener la operación del emprendimiento, detallados en la Tabla E.2.

Tabla E.1: Segmentación del mercado objetivo.

Segmento	**Cantidad**
Reservas naturales en Misiones	116
EAP con bosques y montes	10,802
Total mercado potencial	10,918
Mercado alcanzado (objetivo)	150
Porcentaje del mercado	1.37 %

Tabla E.2: Estructura de costos fijos.

Concepto	**Mensual (USD)**	**Anual (USD)**
Alquiler	150.00	1,800.00
Electricidad	100.00	1,200.00
Agua	30.00	360.00
Dominio	3.00	36.00
Hosting	5.00	60.00
Internet	22.00	264.00
Sueldos	400.00	4,800.00
Teléfono	8.00	96.00
Seguros	15.00	180.00
Total	**733.00**	**8,796.00**

E.4. Costos variables

Los costos variables se estructuran en tres categorías según el tipo de producto o servicio:

E.4.1. Suscripción mensual

La Tabla E.3 detalla el costo de provisión del servicio por cada susciptor activo.

Tabla E.3: Costos variables por suscripción.

Componente	**Costo (USD)**
API de inferencia	2.00
Hosting base de datos	1.00
Costo total	**3.00**
Precio venta	**50.00**
Margen bruto	**1,567 %**

E.4.2. Nodo de cámara

La Tabla E.4 presenta el desglose de costos de fabricación de cada nodo de cámara.

Tabla E.4: Costos variables por nodo de cámara.

Componente	**Costo (USD)**
ESP32-CAM	9.00
Baterías 18650	18.00
PCB	1.00
Sensor PIR	1.00
Cables	1.00
Estaño	1.00
Conectores	3.00
Filamento PLA	3.00
Abrazaderas	5.00
Insertos	1.00
Tornillos	1.00
Convertidor buck	1.00
Costo total	**45.00**
Precio venta	**75.00**
Margen bruto	**67 %**

E.4.3. Nodo raíz

La Tabla E.5 muestra los costos asociados a cada nodo raíz.

Tabla E.5: Costos variables por nodo raíz.

Componente	**Costo (USD)**
ESP32 DevKit	3.00
Cables	1.00
Estaño	1.00
Filamento PLA	3.00
Insertos	1.00
Tornillos	1.00
Costo total	**10.00**
Precio venta	**100.00**
Margen bruto	**900 %**

E.5. Bienes de capital

La inversión inicial contempla la adquisición de equipamiento para fabricación, logística y operaciones, según se detalla en la Tabla E.6.

Tabla E.6: Inversión en bienes de capital.

Concepto	**Precio (USD)**	**Amortización (años)**
Camioneta	15,000.00	5
Impresora 3D	450.00	3
Ploteo	100.00	3
PC de Escritorio	200.00	3
Insumos de Oficina	1,000.00	3
Estación de Soldado	150.00	3
Multímetro	30.00	3
Total equipos	16 930,00	
Mano de obra inicial	3600,00	(240 hs × \$15/h)
Inversión total	20 530,00	

E.6. Flujo de caja operativo

El flujo de caja operativo proyecta los ingresos y egresos sin considerar financiamiento externo, como se presenta en la Tabla E.7.

Tabla E.7: Flujo de caja operativo proyectado (USD).

Concepto	**Año 0**	**Año 1**	**Año 2**	**Año 3**	**Año 4**	**Año 5**
Suscripciones acum.		28	56	84	112	140
Cámaras vendidas		280	280	280	280	280
Nodos raíz vendidos		28	28	28	28	28
Ingreso total		25,200	26,600	28,000	29,400	30,800
Costo total		-21,760	-21,844	-21,928	-22,012	-22,096
Amortización		-3,643	-3,643	-3,643	-3,000	-3,000
Resultado antes imp.		-203	1,113	2,429	4,388	5,704
Impuesto ganancias		0	-389	-850	-1,536	-1,996
Resultado neto		-203	723	1,579	2,852	3,708
Inversión inicial	-20,530					
Recupero IVA		3,150				
Valor residual						4,233
Flujo de caja	-20,530	6,590	4,367	5,222	5,852	6,708
Acumulado	**-20,530**	**-13,940**	**-9,573**	**-4,351**	**1,501**	**8,208**

El período de recupero de la inversión se alcanza durante el **Año 4**.

E.7. Flujo de caja del inversionista

El análisis considera un financiamiento parcial mediante préstamo bancario bajo el sistema francés de amortización:

Tabla E.8: Condiciones del financiamiento.

Parámetro	**Valor**
Monto del préstamo	15,000.00 USD
Plazo	5 años
Tasa de interés	5 % anual
Cuota anual	3,464.62 USD
Sistema de amortización	Francés
Inversión propia	**5,530.00 USD**

La Tabla E.8 resume las condiciones del préstamo y la Tabla E.9 presenta el flujo de caja resultante para el inversionista.

Tabla E.9: Flujo de caja del inversionista (USD).

Concepto	Año 0	Año 1	Año 2	Año 3	Año 4	Año 5
Ingreso total		25,200	26,600	28,000	29,400	30,800
Costo total		-21,760	-21,844	-21,928	-22,012	-22,096
Cuota préstamo		-3,465	-3,465	-3,465	-3,465	-3,465
Utilidad neta		-25	1,291	2,607	3,600	4,456
Inversión propia	-5,530					
Préstamo recibido	15,000					
Valor residual						4,233
Flujo de caja	**-5,530**	**-25**	**1,291**	**2,607**	**3,923**	**5,239**
Acumulado	**-5,530**	**-5,555**	**-4,263**	**-1,656**	**2,268**	**7,507**

E.7.1. Indicadores financieros

La Tabla E.10 resume los principales indicadores de rentabilidad del proyecto.

Tabla E.10: Indicadores de rentabilidad del proyecto.

Indicador	Valor
TREMA (Tasa de Rendimiento Mínima Aceptable)	15 %
TIR (Tasa Interna de Retorno)	25 %
VAN (Valor Actual Neto)	7517,55 USD
Período de recupero	Año 4

Con un VAN positivo de \$7,517.55 y una TIR del 25 % —superior a la TREMA del 15 %—, el proyecto resulta **económicamente viable**.

E.8. Análisis de sensibilidad

Se analizó la sensibilidad del proyecto ante variaciones en las dos variables más críticas: cantidad de suscripciones anuales y precio de la suscripción mensual.

E.8.1. Sensibilidad a cantidad de suscripciones

La Tabla E.11 presenta los valores de TIR y VAN para diferentes cantidades de suscripciones anuales, manteniendo constantes las demás variables del modelo.

La Figura E.1 muestra la relación entre la cantidad de suscripciones anuales y los indicadores de rentabilidad. Se observa una relación lineal entre las variables, donde el punto de equilibrio (VAN = 0) se alcanza con aproximadamente **24 suscripciones anuales**. El proyecto supera la TREMA del 15 % a partir de aproximadamente 26 suscripciones anuales.

Tabla E.11: Sensibilidad a cantidad de suscripciones anuales.

Suscripciones/año	TIR	VAN (USD)
20	-80 %	-6,373.04
22	-35 %	-2,900.39
24	-11 %	572.26
26	8 %	4,044.90
28 (base)	**25 %**	**7,517.55**
30	41 %	10,990.20

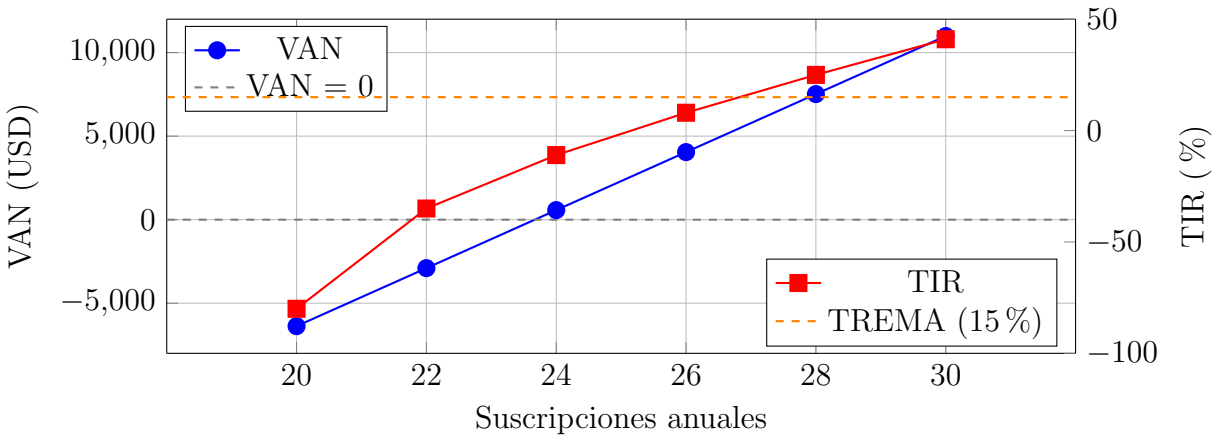

Figura E.1: Sensibilidad del VAN y TIR a la cantidad de suscripciones anuales.

E.8.2. Sensibilidad a precio de suscripción

La Tabla E.12 muestra el impacto de variaciones en el precio mensual de suscripción sobre los indicadores de rentabilidad del proyecto.

Tabla E.12: Sensibilidad al precio mensual de suscripción.

Precio mensual (USD)	TIR	VAN (USD)
30	-4 %	2,406.26
35	5 %	3,684.09
40	12 %	4,961.91
45	19 %	6,239.73
50 (base)	**25 %**	**7,517.55**
55	30 %	8,795.37

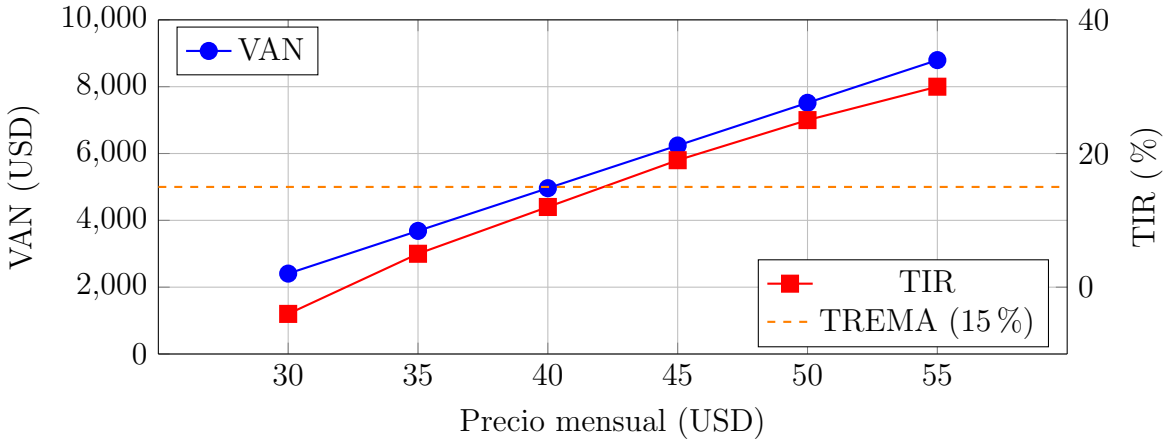

Figura E.2: Sensibilidad del VAN y TIR al precio mensual de suscripción.

La Figura E.2 ilustra cómo varían el VAN y la TIR en función del precio mensual de suscripción. Se observa que el proyecto es viable (TIR > TREMA) a partir de aproximadamente \$43 USD mensuales. El precio mínimo que mantiene el VAN positivo es cercano a **\$30 USD mensuales**, aunque la TIR en ese punto sería negativa.

E.8.3. Conclusiones del análisis económico

El análisis demuestra que el emprendimiento es económicamente viable bajo los supuestos considerados:

- El proyecto genera un VAN positivo de \$7,517.55 USD con una TIR del 25 %, superior a la TREMA del 15 %.
- El período de recupero de la inversión es de 4 años.

- El proyecto tiene margen de seguridad, tolerando una reducción del 14 % en la cantidad de suscripciones (de 28 a 24) antes de volverse inviable.
- El precio de suscripción puede reducirse hasta $40 USD mensuales manteniendo la viabilidad del proyecto.