INSTITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA

MATERIA: SISTEMAS EMBEBIDOS BASADOS EN PROCESAMIENTO DIGITAL DE SEÑALES

DOCENTE: DR. JOSE ANGEL NIEVES VAZQUEZ

INVESTIGACION UNIDAD 4: SISTEMAS BASADOS EN PROCESAMIENTO DIGITAL DE SAÑLES

EQUIPO:

CARLOS COBAXIN VILLASEÑOR

NAOMI ROSAS MINQUIZ

ESMERALDA SERRANO VELÁZQUEZ

MARCOS OSIRIS ZAPOT RAMOS

EDUARDO COYOLT ROSENDO

01 DICIEMBRE DE 2025

INTRODUCCIÓN

El procesamiento digital de señales (DSP) es una de las bases tecnológicas más importantes en la ingeniería moderna, pues permite analizar, transformar y optimizar señales provenientes del entorno físico con alta precisión y velocidad. Su presencia es indispensable en sistemas actuales de comunicación, dispositivos electrónicos, aplicaciones biomédicas, automatización y una gran variedad de tecnologías que requieren operar en tiempo real.

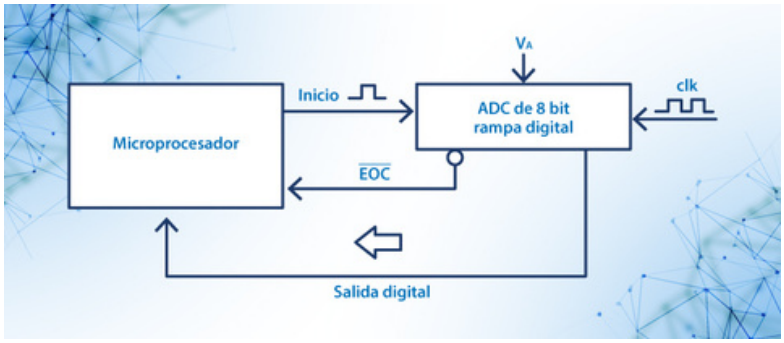
El estudio global del DSP abarca tanto el entendimiento del hardware especializado como la manera en que los algoritmos se implementan y optimizan para funcionar de forma eficiente. Además, las herramientas de programación y los métodos de diseño utilizados permiten transformar modelos matemáticos complejos en soluciones prácticas que pueden integrarse en productos reales. Esta interacción entre teoría, software y hardware ha impulsado la creación de dispositivos más inteligentes, confiables y energéticamente eficientes.

En conjunto, el análisis de estos elementos ofrece una visión clara de cómo el DSP contribuye al desarrollo de tecnologías modernas y por qué representa un componente esencial en sistemas que requieren procesamiento rápido, preciso y continuo. Su estudio no solo fortalece la formación técnica, sino que también abre la puerta a innovaciones que dependen del manejo avanzado de señales digitales.

4.1 Introducción a arquitectura de un sistema DSP

Los sistemas de procesamiento digital de señales (DSP, por sus siglas en inglés) constituyen un componente esencial dentro de la electrónica moderna, especialmente en aplicaciones que requieren manipulación precisa, rápida y eficiente de señales analógicas convertidas a formato digital. La arquitectura de un DSP está diseñada específicamente para realizar operaciones matemáticas complejas de manera optimizada, superando ampliamente a los microcontroladores y microprocesadores de propósito general en este tipo de tareas.

En un sistema DSP, la señal analógica se adquiere mediante un conversor analógico-digital (ADC), se procesa mediante el procesador digital y posteriormente, si es necesario, se devuelve al mundo analógico a través de un conversor digital-analógico (DAC). El corazón del sistema es el procesador DSP, cuya estructura arquitectónica está optimizada para ejecutar operaciones matemáticas repetitivas como multiplicaciones, sumas, convoluciones y transformadas.



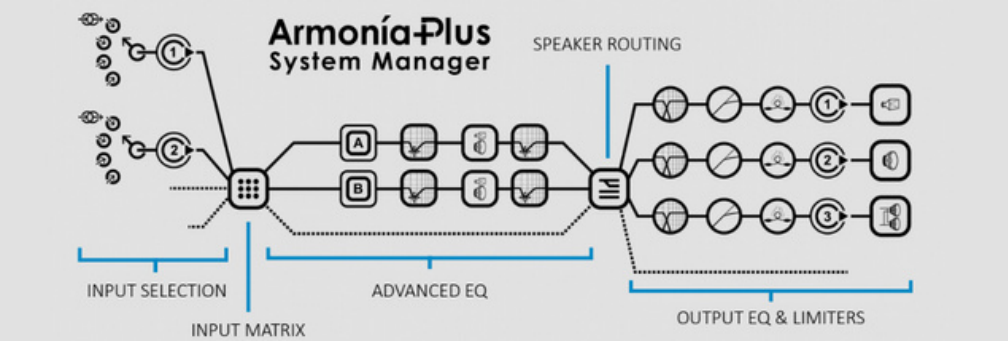
Uno de los elementos más característicos de los DSP es la unidad MAC (Multiply-Accumulate), que permite realizar una multiplicación seguida de una acumulación en un solo ciclo de reloj. Esto representa una ventaja notable en aplicaciones como filtrado digital, transformada rápida de Fourier (FFT) y algoritmos de control. Además, los DSP suelen contar con instrucciones especiales que permiten ejecutar varias operaciones simultáneamente mediante un conjunto de arquitecturas Harvard modificadas, con memorias de instrucción y datos separadas, lo cual permite accesos simultáneos y mayor velocidad.

Los DSP integran además mecanismos de direccionamiento avanzado, como el direccionamiento circular, que facilita la implementación eficiente de buffers o ventanas en filtros FIR y sistemas de adquisición continua. Otro mecanismo común es el direccionamiento bit-reversal, útil para organizar datos en transformadas FFT, evitando operaciones adicionales en software que consumirían tiempo y recursos.

En términos de memoria, los DSP incorporan diversas jerarquías optimizadas para minimizar latencia. Suelen contar con RAM interna de acceso rápido, memoria caché especializada y módulos DMA (Direct Memory Access) que permiten realizar transferencias sin intervención directa del procesador, aumentando la eficiencia global del sistema.

A nivel periférico, los DSP incluyen interfaces diseñadas para aplicaciones de señal: puertos seriales de alta velocidad como I2S, SPI o McBSP, temporizadores avanzados, módulos para procesamiento paralelo y, en algunos casos, coprocesadores adicionales que aceleran operaciones matemáticas o algoritmos específicos.

En sectores como telecomunicaciones, sistemas biomédicos, procesamiento multimedia y control industrial, la arquitectura DSP proporciona rendimiento y precisión indispensables. Su capacidad para operar en tiempo real los convierte en la opción ideal para aplicaciones donde la latencia es un factor crítico.



4.2 Instrucciones de un DSP

Los DSP poseen conjuntos de instrucciones diseñados específicamente para optimizar operaciones matemáticas intensivas, manipulación de datos y acceso eficiente a memoria. Aunque cada fabricante (Texas Instruments, Analog Devices, NXP, etc.) puede implementar variaciones propias, la mayoría de los DSP comparten una serie de instrucciones fundamentales y modos de operación que permiten alcanzar un desempeño de alto nivel en tiempo real.

Una de las instrucciones más importantes es la instrucción MAC (Multiply-Accumulate), que combina multiplicación y suma en una sola operación. Esta instrucción se encuentra en el núcleo del procesamiento digital, pues permite implementar filtros FIR, IIR, convoluciones y cálculos de energía de manera eficiente. Existen variantes como Multiply-Subtract, Multiply-Add con saturación, y operaciones paralelas de MAC múltiple en DSP de gama alta.

Además, muchos DSP incluyen instrucciones SIMD (Single Instruction, Multiple Data), que permiten procesar varios datos en paralelo. Este tipo de instrucción es vital en aplicaciones multimedia y de telecomunicaciones donde se procesan vectores completos en pocos ciclos.

Otro tipo de instrucciones relevantes son las orientadas al control y flujo del programa. Sin embargo, aunque similares a las de un microcontrolador tradicional, los DSP incluyen instrucciones condicionales optimizadas y saltos rápidos para minimizar latencia, especialmente en aplicaciones de control adaptativo o algoritmos con bucles intensivos.

Las instrucciones orientadas a manipulación de memoria son también esenciales. Los DSP incluyen modos de direccionamiento especializados, como:

- **Direccionamiento circular:** Util para buffers continuos.
- **Direccionamiento automático o indexado:** Que incrementa punteros sin necesidad de instrucciones adicionales.
- **Bit-reversal**: Utilizado en FFT para reorganizar datos automáticamente.

Asimismo, los DSP ofrecen instrucciones de saturación, que limitan los resultados para evitar desbordamientos, esenciales en sistemas donde la estabilidad del algoritmo depende de mantener valores dentro de rangos definidos. También se encuentran instrucciones para normalización automática, desplazamientos aritméticos y aritmética en punto fijo o punto flotante.

En los DSP modernos se suelen integrar instrucciones para operaciones complejas como cálculo de raíces cuadradas, magnitud de vectores, multiplicación compleja y transformaciones matemáticas avanzadas. Todo esto permite que los algoritmos se ejecuten en menos tiempo y con menor carga computacional.

4.3 Herramientas de programación

Las herramientas de programación para DSP han evolucionado significativamente, permitiendo a ingenieros y desarrolladores diseñar aplicaciones robustas sin necesidad de programar exclusivamente en ensamblador. Entre las herramientas más utilizadas se encuentran compiladores optimizados, entornos de desarrollo integrados (IDE), bibliotecas matemáticas y simuladores que permiten probar el funcionamiento de los algoritmos antes de implementarlos en hardware.

Texas Instruments ofrece entornos como Code Composer Studio (CCS), ampliamente utilizado en DSP de las familias C6000 y C5000. Este software incluye compiladores avanzados, herramientas de depuración, perfiles de rendimiento y simulación. También incorpora bibliotecas optimizadas como DSPLib y MathLib, que contienen implementaciones eficientes de filtros, transformadas y operaciones numéricas.

Analog Devices, por su parte, ofrece CrossCore Embedded Studio, que permite trabajar con DSP SHARC o Blackfin. Estas plataformas incluyen herramientas de optimización, generadores automáticos de código y perfiles que analizan el uso del procesador para garantizar que el algoritmo cumple con los requisitos de tiempo real.

Otras herramientas de programación comunes incluyen MATLAB y Simulink, ampliamente utilizados en etapas iniciales de diseño. MATLAB permite desarrollar algoritmos digitales, simularlos y posteriormente generar código C optimizado mediante Embedded Coder. En algunos casos es posible generar código directamente compatible con plataformas DSP específicas, lo cual reduce drásticamente tiempos de desarrollo.

Asimismo, existen lenguajes específicos y bibliotecas de alto rendimiento como FFTW, CMSIS-DSP (para procesadores ARM) y librerías especializadas para filtros complejos, comunicaciones digitales o procesamiento de audio.

La programación de DSP también requiere herramientas de verificación, como analizadores lógicos, osciloscopios y perfiles de ejecución que permiten medir ciclos de reloj, latencia y uso de memoria. Los simuladores integrados en los IDE suelen ofrecer capacidad de emulación ciclo a ciclo, permitiendo verificar el comportamiento del algoritmo sin necesidad de cargarlo en la tarjeta física.

4.4 Desarrollo de aplicaciones

El desarrollo de aplicaciones basadas en DSP implica un proceso estructurado que abarca desde el diseño teórico del algoritmo hasta su implementación en hardware. El primer paso consiste en analizar la naturaleza de la señal que se desea procesar y definir los requerimientos: ancho de banda, frecuencia de muestreo, precisión necesaria, latencia máxima permitida y tipo de operación requerida (filtrado, compresión, reconocimiento, etc.).

Una vez definidos los requerimientos, los ingenieros desarrollan el algoritmo matemático en un entorno de simulación, generalmente MATLAB o Python. En esta etapa se evalúa el desempeño del algoritmo, se optimizan parámetros y se implementan pruebas de estabilidad, sensibilidad y eficiencia numérica. Posteriormente, el algoritmo se traduce a código C o lenguaje compatible con el DSP seleccionado.

El desarrollo continua con la optimización del código, pues incluso algoritmos correctos pueden fallar en tiempo real si no cumplen con los requisitos de latencia o consumo de procesamiento. Las herramientas del IDE permiten identificar cuellos de botella, optimizar bucles y aprovechar instrucciones específicas del DSP, como instrucciones MAC o SIMD.

Las aplicaciones de DSP abarcan una amplia gama de industrias. En telecomunicaciones se emplean para modulación, demodulación, ecualización y codificación de canal. En audio, se utilizan para cancelación de ruido, ecualización, mezcla digital y efectos. En biomédica, los DSP procesan señales ECG, EEG y ultrasonido, detectando patrones críticos en tiempo real. En automoción se aplican en procesamiento radar, LIDAR y sistemas de asistencia avanzada al conductor.

Una vez implementado el algoritmo en hardware, se llevan a cabo pruebas en condiciones reales. Esto incluye validar la respuesta con señales reales, medir latencia, estabilidad térmica, consumo energético y capacidad de operación continua.

Finalmente, se integran interfaces de comunicación, conversiones analógicas, drivers, controladores y protocolos que permitan la incorporación del sistema DSP dentro de un producto final.

CONCLUSIÓN

El estudio del procesamiento digital de señales permite comprender cómo la tecnología actual depende de sistemas capaces de analizar y transformar información de manera rápida y precisa. Dentro de este campo convergen conocimientos de hardware especializado, algoritmos matemáticos y herramientas de programación que, en conjunto, hacen posible el funcionamiento de dispositivos y aplicaciones que utilizamos diariamente.

A través de esta revisión se aprecia que el DSP no solo es un componente técnico, sino una disciplina integral que impulsa innovaciones en comunicaciones, electrónica, salud, automatización y muchas otras áreas. Su capacidad para operar en tiempo real, optimizar recursos y garantizar resultados confiables lo convierte en un pilar fundamental en el diseño de sistemas modernos.

Asimismo, la importancia del DSP radica en su enfoque multidisciplinario: requiere habilidades analíticas, dominio de estrategias de optimización y comprensión de las limitaciones físicas del hardware. Esta combinación permite que las soluciones desarrolladas sean eficientes y adecuadas para implementarse en entornos reales.

En síntesis, el procesamiento digital de señales continúa siendo un motor tecnológico clave, proporcionando las herramientas necesarias para enfrentar desafíos actuales y futuros, y demostrando su papel central en el avance de sistemas inteligentes y de alto rendimiento.

REFERENCIAS BIBLIOGRÁFICAS

- Smith, S. W. (1999). The Scientist and Engineer's Guide to Digital Signal Processing (2nd ed.). California Technical Publishing. Recuperado de https://www.dspguide.com/

- Ingle, V. K., & Proakis, J. G. (2017). Digital Signal Processing: Principles, Algorithms and System Design. Academic Press.

- Lyons, R. G. (2011). Understanding Digital Signal Processing (3ª ed.). Prentice Hall.

- Oppenheim, A. V., Schafer, R. W., & Buck, J. R. (2019). Discrete-Time Signal Processing (3rd ed.). Pearson. Recuperado de https://www.pearson.com/en-us/subject-catalog/p/discrete-time-signal-processing/P200000006847/

LISTA DE COTEJO INVESTIGACION

SISTEMAS EMBEBIDOS BASADOS EN PROCESAMIENTO DIGITAL DE SEÑALES.

Nombre del estudiante: COBAXIN VILLASEÑOR CARLOS.

Tema: SEÑALES Y SISTEMAS EN TIEMPO DISCRETO.

Portada	2 %	2 %
Introducción	5 %	5 %
Desarrollo	10 %	10 %
Conclusiones	5 %	5%
Referencias	3 %	3 %
Entrega en tiempo y forma	5 %	5 %
Examen (Asistencias)	10 %	4 %
Total	40 %	34%

LISTA DE COTEJO DE PRÁCTICAS

SISTEMAS EMBEBIDOS BASADOS EN PROCESAMIENTO DIGITAL DE SEÑALES.

Nombre del estudiante: COBAXIN VILLASEÑOR CARLOS.

Tema: REPORTE DE PROYECTO FINAL: SISTEMA DE DETECCIÓN DE FAUNA INTRUSA EN HUERTOS.

Portada	5 %	0 %
Introducción	10 %	10 %
Desarrollo	30 %	17 %
Conclusiones	5 %	5 %
Referencias	2 %	0 %
Entrega en tiempo y forma	8 %	8 %
Total	60 %	40 %

Nota: La portada no coincide con la materia, el desarrollo no es adecuado y no tienen referencias.

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

División de Ingeniería Mecatrónica

Materia: Microcontroladores.

Docente: Dr. José Ángel Nieves Vázquez.

Periodo: Agosto-Enero 2026.

Reporte de Proyecto Final: Sistema de detección de fauna intrusa en huertos

Integrantes:

- Rosas Minquiz Naomi
- Cobaxin Villaseñor Carlos
- Serrano Velázquez Esmeralda
- Coyolt Rosendo Eduardo
- Zapot Ramos Marcos Osiris

INTRODUCCIÓN

La presencia de fauna intrusa en huertos representa un desafío constante para los productores agrícolas, ya que puede ocasionar daños significativos en los cultivos, reducir la calidad de la cosecha y afectar la eficiencia de los procesos productivos. Frente a esta problemática, surge la necesidad de desarrollar soluciones tecnológicas capaces de identificar y disuadir oportunamente la presencia de animales no deseados, garantizando así la protección del huerto sin comprometer el equilibrio ecológico.

En este proyecto de Diseño e Implementación de un Modelo de Inteligencia Artificial para la Identificación y Disuasión de Fauna Intrusa en un huerto, se propone el desarrollo de una solución basada en microcontroladores que integre alarmas y algoritmos de detección. El sistema tiene como objetivo reconocer la presencia de fauna mediante tecnologías mediante un microcontrolador ESP CAM32 y procesar la información mediante una red neuronal en YOLOV8 en tiempo real .

Este proyecto, enmarcado en la materia de Microcontroladores, permite aplicar conocimientos sobre electrónica digital, procesamiento de señales, programación embebida y comunicación entre dispositivos, consolidando un enfoque práctico orientado a la resolución de problemas reales. Además, fomenta el diseño de sistemas autónomos e inteligentes que contribuyen al fortalecimiento de prácticas agrícolas más eficientes y sostenibles.

JUSTIFICACIÓN

La agricultura moderna requiere soluciones tecnológicas que incrementen la productividad y reduzcan pérdidas causadas por factores externos. Entre ellos, el ingreso de fauna intrusa constituye uno de los más comunes y costosos. Aunque existen métodos tradicionales como cercas, espantapájaros o repelentes químicos, muchos resultan poco eficaces, costosos o dañinos para el medio ambiente.

Este proyecto se justifica por las siguientes razones:

Relevancia agrícola

- Reduce daños ocasionados por fauna sin necesidad de métodos agresivos.
- Permite la detección temprana y respuesta inmediata.

Impacto tecnológico

- Promueve el uso de microcontroladores como plataforma versátil para el desarrollo de sistemas inteligentes.
- Integra algoritmos, en áreas clave de la electrónica moderna.

Sustentabilidad

- El sistema emplea métodos de disuasión no invasivos para proteger tanto a los animales como a los cultivos.
- Contribuye a prácticas agrícolas responsables y alineadas con la conservación del entorno.

Viabilidad

- Sus componentes son de bajo costo, fáciles de integrar y mantener.
- Puede expandirse a diferentes tipos de cultivos y configurarse según la fauna local.

OBJETIVOS

Objetivo general

Diseñar e implementar un sistema de Inteligencia Artificial basado en ESPCAM32 que permita identificar la presencia de fauna intrusa en huertos.

Objetivos específicos

1. Integrar la cámara de EspCAM32 para la detección de movimiento o presencia animal.
2. Desarrollar un algoritmo capaz de analizar la información en tiempo real.
3. Construir el bot de Telegram que recibirá la alerta y foto del animal.
4. Evaluar el rendimiento del sistema en diferentes condiciones de iluminación, clima y distancia.
5. Optimizar el consumo energético para garantizar su funcionamiento continuo.

DIAGRAMA CONCEPTUAL DEL SISTEMA (DESCRIPCIÓN)

Entrada (Detección)
→ Cámara

Procesamiento
→ Microcontrolador (ESP32-CAM)
→ Algoritmo de detección
→ Filtrado de datos

Salida (Disuasión)
→ Alarma en celular
→ Registro de eventos

Retroalimentación
→ El sistema se reinicia y queda en espera para una nueva detección.

CODIGOS

```python
import cv2
import numpy as np
import requests
from ultralytics import YOLO
import telebot
import time

# ========================
# CONFIGURACIÓN
# ========================
ESP32_URL = "http://10.164.156.38/foto"
MODEL_PATH = r"C:/Users/HP/intrusos_dataset/runs/detect/train6/weights/best.pt"

BOT_TOKEN = "8534838126:AAF2VbVGzTsRKpxUeC74526fTSjsymWF8aA"
CHAT_ID = "8433061754"

bot = telebot.TeleBot(BOT_TOKEN)


# ========================
# CARGAR MODELO YOLO
# ========================
print("Cargando modelo YOLO...")
model = YOLO(MODEL_PATH)
print("🚀 Sistema iniciado. Vigilando animales...")


# ========================
# FUNCIONES
# ========================
def obtener_imagen():
    """
    Captura una imagen del ESP32-CAM.
    Con tiempo de espera alto para evitar fallos por saturación.
    """
    try:
        r = requests.get(ESP32_URL, timeout=10)
```

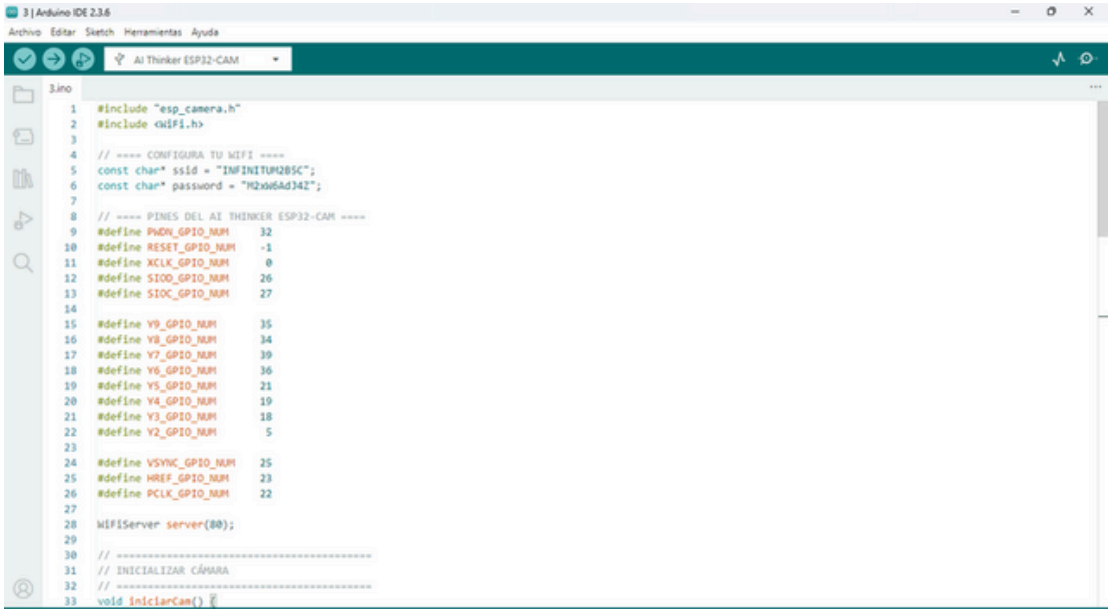
```python
        if r.status_code != 200:
            print("⚠️ ESP32 devolvió código:", r.status_code)
            return None

        img_bytes = np.frombuffer(r.content, dtype=np.uint8)
        img = cv2.imdecode(img_bytes, cv2.IMREAD_COLOR)

        if img is None:
            print("⚠️ Imagen inválida (None)")
            return None

        return img

    except requests.exceptions.Timeout:
        print("⏳ Timeout: ESP32 tardó demasiado en responder.")
        return None

    except requests.exceptions.ConnectionError:
        print("⚠️ Error de conexión con ESP32 (posible saturación)")
        return None

    except Exception as e:
        print("⚠️ Error inesperado ESP32:", e)
        return None


def enviar_alerta(img, animal, conf):
    """
    Envía alerta a Telegram con manejo de errores robusto
    para evitar error 409 o saturación del bot.
    """
    try:
        mensaje = f"🚨 *Animal detectado*\n🐾 {animal}"
        bot.send_message(CHAT_ID, mensaje, parse_mode="Markdown")

        cv2.imwrite("alerta.jpg", img)
```

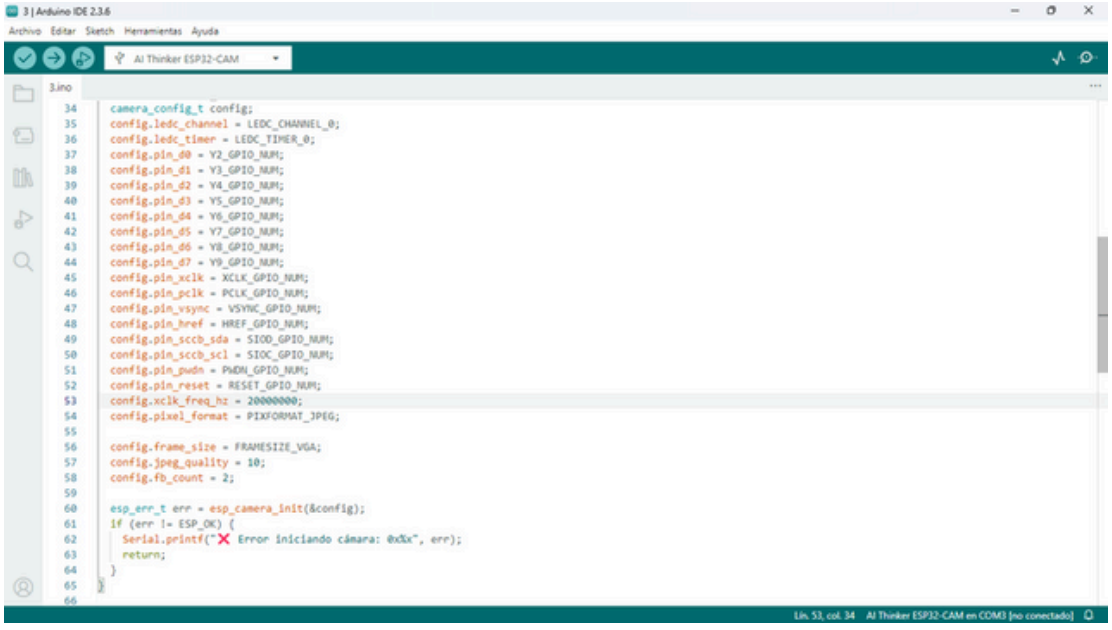
CODIGOS

```
    # 3) Procesar con YOLO
    try:
        results = model(img, verbose=False)[0]

        for box in results.boxes:
            cls = int(box.cls[0])
            conf = float(box.conf[0])
            animal = model.names[cls]

            if conf > 0.70:
                if time.time() - ultimo_envio > COOLDOWN:
                    enviar_alerta(img, animal, conf)
                    ultimo_envio = time.time()

    except Exception as e:
        print("⚠️ Error procesando YOLO:", e)

    # 4) Evitar saturación de la ESP32-CAM
    time.sleep(FPS_DELAY)

cv2.destroyAllWindows()

conda: base (Python 3.12.7)  Completions: conda(base)  ✓ LSP:
```

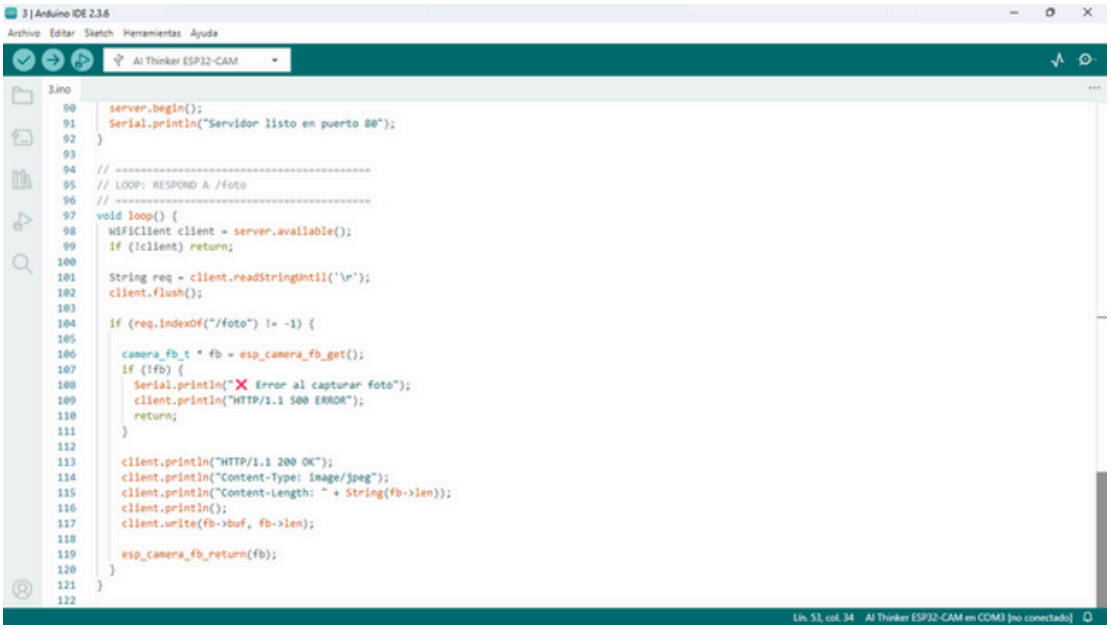
```
        with open("alerta.jpg", "rb") as f:
            bot.send_photo(CHAT_ID, f)

        print("📤 Alerta enviada a Telegram")

    except telebot.apihelper.ApiTelegramException as e:
        print(f"⚠️ Error Telegram: {e}")

    except Exception as e:
        print(f"⚠️ Error inesperado enviando alerta: {e}")


# ========================
# LOOP PRINCIPAL
# ========================
ultimo_envio = 0
COOLDOWN = 10   # segundos entre alertas
FPS_DELAY = 0.25  # delay para no saturar la ESP32 (IMPORTANTE)

while True:

    # 1) Obtener imagen estable
    img = obtener_imagen()

    if img is None:
        time.sleep(1)  # esperar para no saturar
        continue

    # 2) Mostrar imagen
    cv2.imshow("ESP32-CAM", img)
    if cv2.waitKey(1) == 27:
        break

    # 3) Procesar con YOLO
    try:
        results = model(img, verbose=False)[0]

```

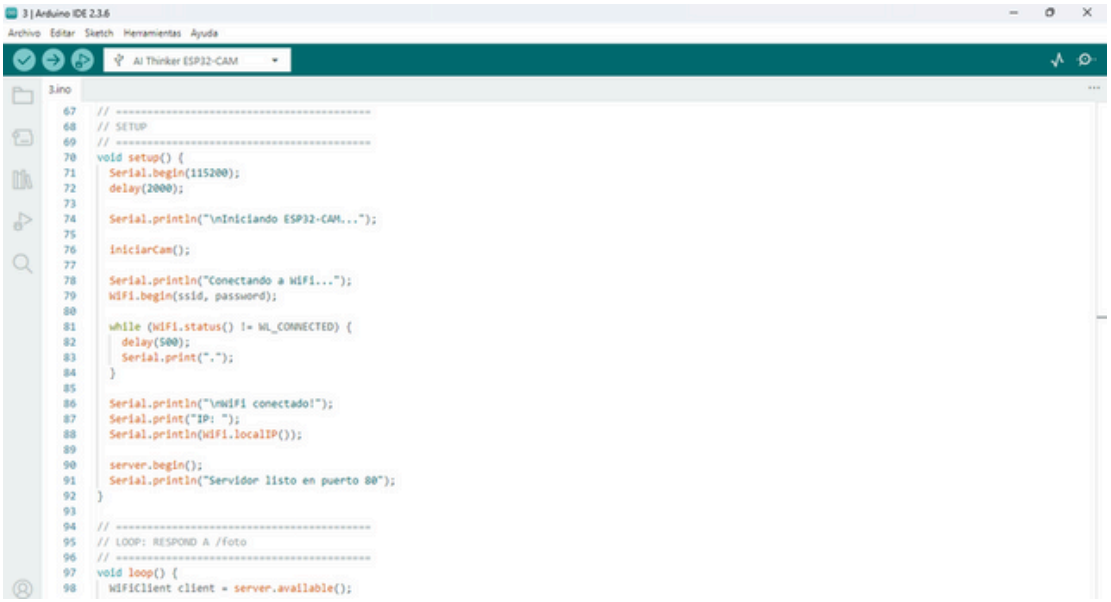
CODIGOS


3 | Arduino IDE 2.3.6
Archivo Editar Sketch Herramientas Ayuda
AI Thinker ESP32-CAM
3.ino
#include "esp_camera.h"
#include <WiFi.h>
// ==== CONFIGURA TU WIFI ====
const char* ssid = "INFINITUM2B5C";
const char* password = "M2aW6Ad34Z";
// ==== PINES DEL AI THINKER ESP32-CAM ====
#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27
#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22
WiFiServer server(80);
// ==
// INICIALIZAR CÁMARA
// ==
void iniciarCam() {

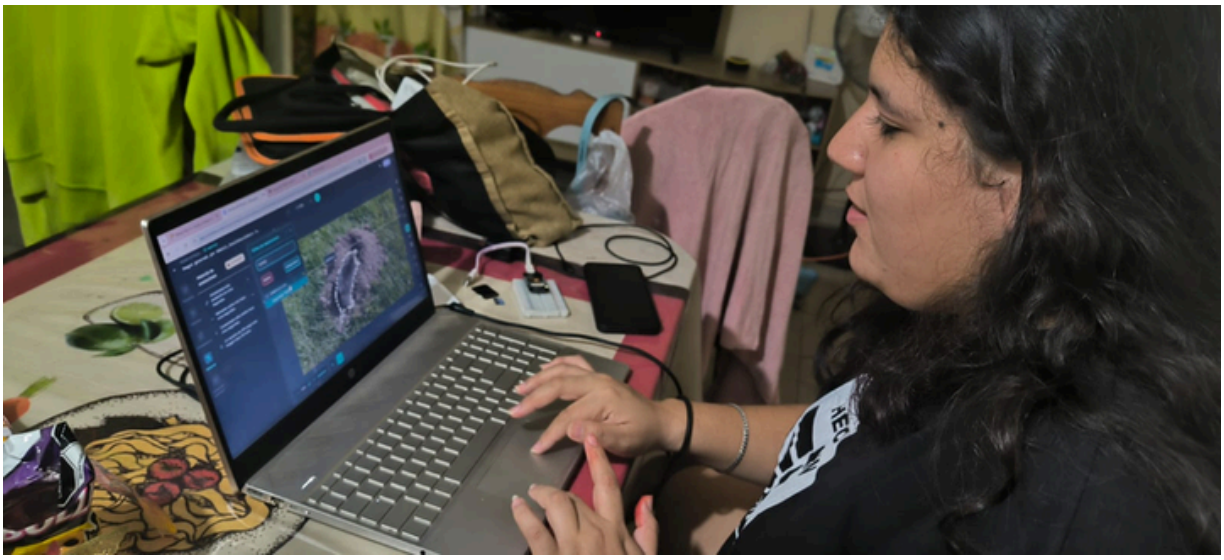

3 | Arduino IDE 2.3.6
Archivo Editar Sketch Herramientas Ayuda
AI Thinker ESP32-CAM
3.ino
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sccb_sda = SIOD_GPIO_NUM;
config.pin_sccb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size = FRAMESIZE_VGA;
config.jpeg_quality = 10;
config.fb_count = 2;
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
Serial.printf("❌ Error iniciando cámara: 0x%x", err);
return;
}
}
Lín. 53, col. 34 AI Thinker ESP32-CAM en COM3 [no conectado]

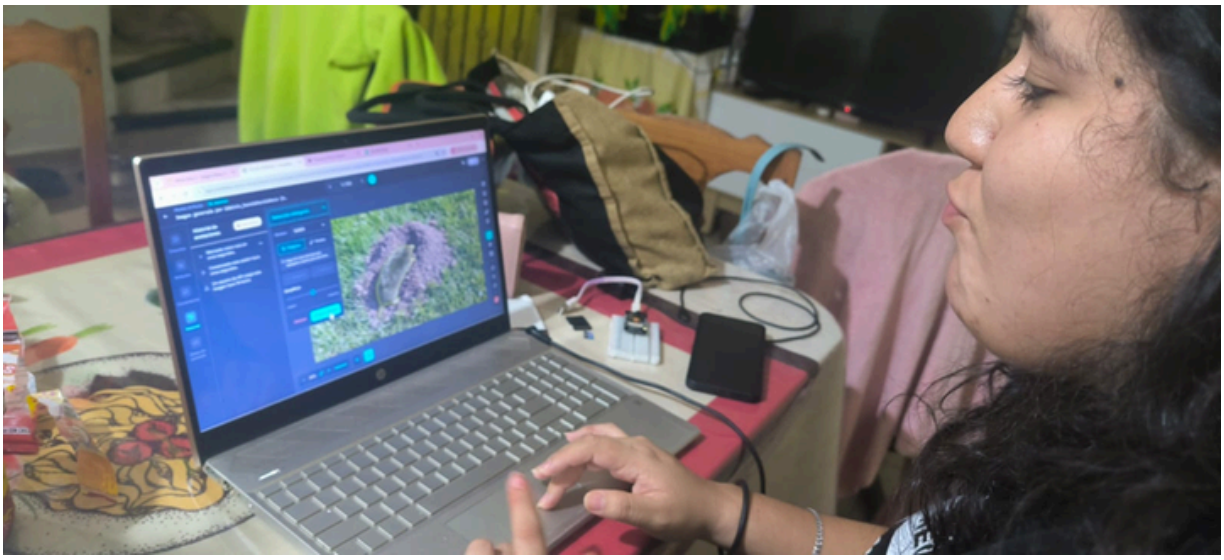
CODIGOS


3 | Arduino IDE 2.3.6
Archivo Editar Sketch Herramientas Ayuda
AI Thinker ESP32-CAM
3.ino
server.begin();
Serial.println("Servidor listo en puerto 80");
}
// ==
// LOOP: RESPOND A /foto
// ==
void loop() {
WiFiClient client = server.available();
if (!client) return;
String req = client.readStringUntil('\r');
client.flush();
if (req.indexOf("/foto") != -1) {
camera_fb_t * fb = esp_camera_fb_get();
if (!fb) {
Serial.println("❌ Error al capturar foto");
client.println("HTTP/1.1 500 ERROR");
return;
}
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: image/jpeg");
client.println("Content-Length: " + String(fb->len));
client.println();
client.write(fb->buf, fb->len);
esp_camera_fb_return(fb);
}
}
Lín. 53, col. 34 AI Thinker ESP32-CAM en COM3 [no conectado]

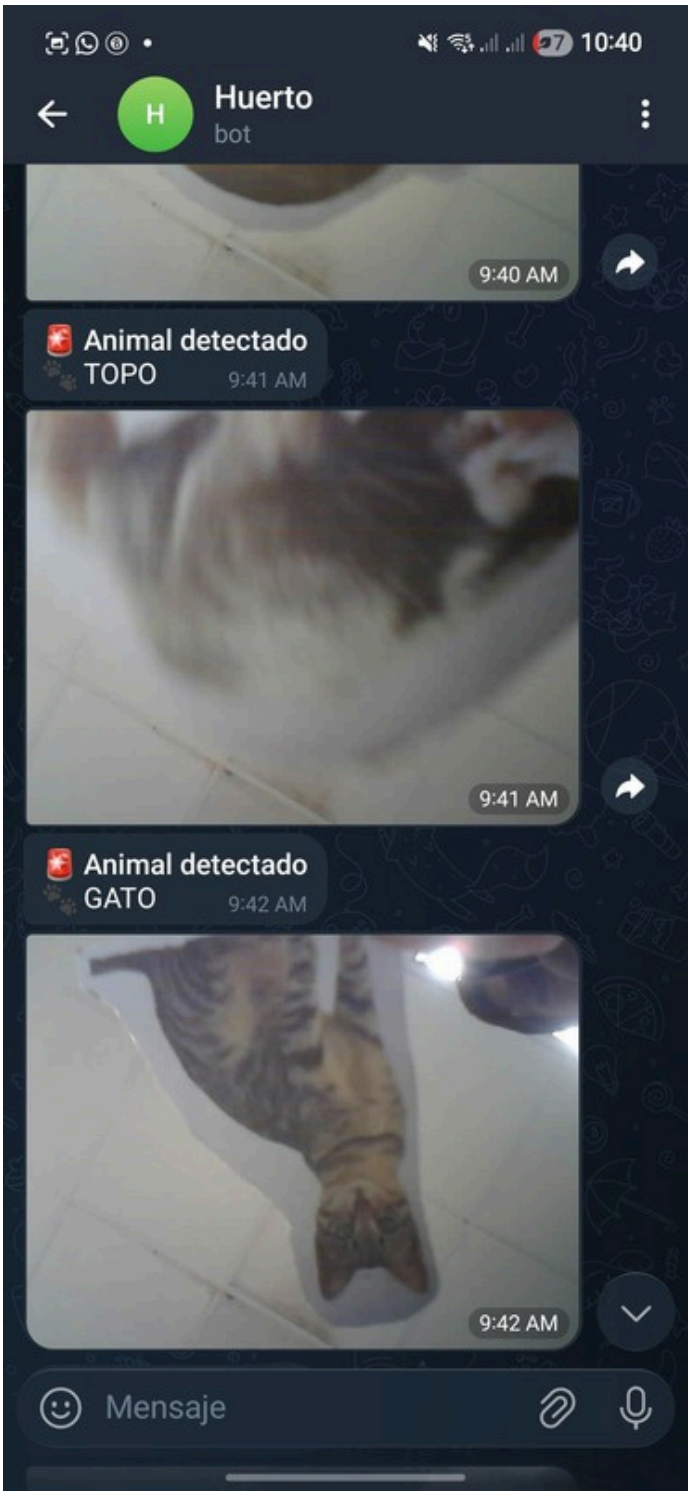

3 | Arduino IDE 2.3.6
Archivo Editar Sketch Herramientas Ayuda
AI Thinker ESP32-CAM
3.ino
// ==
// SETUP
// ==
void setup() {
Serial.begin(115200);
delay(2000);
Serial.println("\nIniciando ESP32-CAM...");
iniciarCam();
Serial.println("Conectando a WiFi...");
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("\nWiFi conectado!");
Serial.print("IP: ");
Serial.println(WiFi.localIP());
server.begin();
Serial.println("Servidor listo en puerto 80");
}
// ==
// LOOP: RESPOND A /foto
// ==
void loop() {
WiFiClient client = server.available();

DESARROLLO DEL PROYECTO

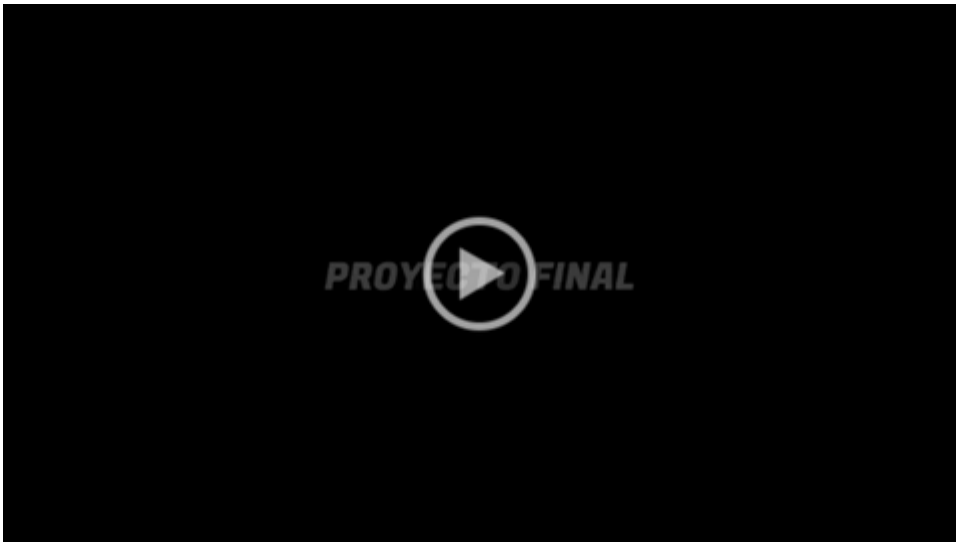




MENSAJE ALERTA EN TELEGRAM



LINK DEL VIDEO



CONCLUCIÓN

En el presente proyecto se desarrolló un sistema de monitoreo inteligente para un huerto casero empleando el microcontrolador ESP32-CAM, el cual integra capacidades de procesamiento, comunicación inalámbrica y visión artificial en una sola plataforma. El sistema fue diseñado como un sistema embebido, ya que cumple una función específica, opera de manera autónoma, interactúa directamente con su entorno mediante sensores y actuadores, y trabaja bajo restricciones de recursos como memoria, potencia de cómputo y consumo energético.

La implementación del sistema permitió aplicar de manera práctica los conceptos fundamentales abordados en la materia de microcontroladores, tales como la configuración y uso de periféricos, el manejo de entradas y salidas digitales, la comunicación serial e inalámbrica, la gestión de interrupciones y temporización, así como la programación a bajo nivel orientada al control del hardware. Asimismo, el uso de la ESP32-CAM evidenció la importancia de la correcta administración de recursos del microcontrolador, especialmente al integrar módulos demandantes como la cámara y los algoritmos de inteligencia artificial.

Desde la perspectiva de los sistemas embebidos, el proyecto demostró la viabilidad de integrar técnicas de inteligencia artificial en dispositivos de bajo costo y recursos limitados, mediante el uso de modelos de aprendizaje automático optimizados. La capacidad del sistema para capturar imágenes, procesarlas localmente, tomar decisiones en tiempo real y generar acciones de respuesta, como la activación de una alarma y el envío de notificaciones remotas, refleja el comportamiento típico de un sistema embebido moderno orientado a aplicaciones del Internet de las Cosas (IoT).

Los resultados obtenidos muestran que el sistema puede contribuir de manera efectiva a la protección de cultivos al detectar la presencia de intrusos animales y alertar oportunamente al usuario, reduciendo pérdidas y mejorando la supervisión del huerto. No obstante, se identifican áreas de mejora relacionadas con la precisión del modelo de detección ante variaciones ambientales, la reducción de falsos positivos y la optimización del consumo energético para su operación continua.

En conclusión, este proyecto integra de forma sólida los conocimientos de microcontroladores y sistemas embebidos, demostrando su aplicación en un problema real mediante el desarrollo de una solución funcional, escalable y adaptable.