Autonomous Object-Finding Robot: Mechanical, Electronic, and AI Integration Report

Bryan Javier Torres Bravo
Applied Mathematics, UASLP

Project Start: September 13, 2025
Project End: November 9, 2025

1 Objective

The objective of this project was to apply concepts from two key references: *Neural Networks and Deep Learning* by Michael Nielsen and *Modern Robotics: Mechanics, Planning, and Control* by Kevin M. Lynch and Frank C. Park. The goal was to design and implement a robot capable of helping a user locate objects through teleoperation and computer vision.

2 Project Overview

The project focused on integrating mechanics, electronics, control systems, and artificial intelligence into a functional robotic platform. The user could teleoperate the robot via a computer keyboard, while the system attempted object detection using custom and pre-trained neural networks.

3 Mechanical Design

In a previous robotics project, lack of experience led to neglecting key aspects such as mechanical load and motor torque. This time, motor selection and movement stability were prioritized. Two 12V DC motors with encoders were selected (Garosa 12V DIY Encoder Motorreductor Kit), ensuring full speed control and reliable motion.

The robot chassis also supported:

- A 12V 5200mAh lithium battery,
- A USB webcam (UGREEN 2K),
- A DC–DC buck converter (12–48V to 5V 3A),
- A breadboard for wiring,

- A power bank,
- A Raspberry Pi 5.

Originally, everything would be powered from the same battery. However, voltage drops from motor activity caused the Raspberry Pi to lose connection. The Pi was ultimately powered independently, sharing only a common ground with the rest of the system to allow proper encoder readings.

4 Camera Mount and Servo System

Initially, the camera was mounted on two 9g micro servos to provide full 360° vision and vertical tilt. While the servos functioned correctly on an Arduino R4 Minima, they performed poorly on the Raspberry Pi 5.

The Raspberry Pi 5 produces jitter on servo PWM signals because its GPIO does not provide true real-time outputs; operating system interrupts destabilize pulse timing. This caused severe camera vibration and degraded image quality. Due to time and technical constraints, the servo system was removed and the camera was mounted in a fixed position. The differential-drive base allowed the robot to rotate and obtain a panoramic view despite the lack of tilt.

```
import RPi.GPIO as GPIO
import time
import socket
import threading

#
# PIN ASSIGNMENTS
#
# Motor A pins
ENA = 10
IN1 = 8
IN2 = 9

# Motor B pins
IN3 = 11
IN4 = 12
ENB = 6

# Servo motor pin
SERVO_PIN = 7
servo = None      # Will be initialized after GPIO setup
servo_pos = 0     # Servo PWM position value

#
# GLOBAL VARIABLES
#
VELOCIDAD = 50    # Default motor speed (duty cycle percentage)
GPIO_LOW = 0
GPIO_HIGH = 1
```

```
#
# GPIO CONFIGURATION
#
GPIO.setmode(GPIO.BCM)

# Motor pin configuration
GPIO.setup([IN1, IN2, IN3, IN4], GPIO.OUT)
GPIO.setup([ENA, ENB], GPIO.OUT)

# Servo pin configuration
GPIO.setup(SERVO_PIN, GPIO.OUT)

# PWM configuration for motor speed control
pwmA = GPIO.PWM(ENA, 1000)   # PWM at 1 kHz
pwmB = GPIO.PWM(ENB, 1000)
pwmA.start(VELOCIDAD)
pwmB.start(VELOCIDAD)

# PWM configuration for servo (50 Hz is standard for hobby servos)
servo = GPIO.PWM(SERVO_PIN, 50)
servo.start(0)

print("Robot ready (Wi-Fi server active).")

#
# MOVEMENT FUNCTIONS
#
def adelante(velocidad):
    """
    Moves the robot forward by setting both motors to forward direction.
    """
    GPIO.output(IN1, GPIO_HIGH)
    GPIO.output(IN2, GPIO_LOW)
    GPIO.output(IN3, GPIO_HIGH)
    GPIO.output(IN4, GPIO_LOW)
    pwmA.ChangeDutyCycle(velocidad)
    pwmB.ChangeDutyCycle(velocidad)

def atras(velocidad):
    """
    Moves the robot backward by reversing both motors.
    """
    GPIO.output(IN1, GPIO_LOW)
    GPIO.output(IN2, GPIO_HIGH)
    GPIO.output(IN3, GPIO_LOW)
    GPIO.output(IN4, GPIO_HIGH)
    pwmA.ChangeDutyCycle(velocidad)
    pwmB.ChangeDutyCycle(velocidad)

def izquierda(velocidad):
    """
    Turns the robot left by reversing Motor A and moving Motor B forward.
    """
    GPIO.output(IN1, GPIO_LOW)
```

```
    GPIO.output(IN2, GPIO_HIGH)
    GPIO.output(IN3, GPIO_HIGH)
    GPIO.output(IN4, GPIO_LOW)
    pwmA.ChangeDutyCycle(velocidad)
    pwmB.ChangeDutyCycle(velocidad)

def derecha(velocidad):
    """
    Turns the robot right by moving Motor A forward and reversing Motor B.
    """
    GPIO.output(IN1, GPIO_HIGH)
    GPIO.output(IN2, GPIO_LOW)
    GPIO.output(IN3, GPIO_LOW)
    GPIO.output(IN4, GPIO_HIGH)
    pwmA.ChangeDutyCycle(velocidad)
    pwmB.ChangeDutyCycle(velocidad)

def detener():
    """
    Stops all motor movement and sets duty cycle to zero.
    """
    GPIO.output([IN1, IN2, IN3, IN4], GPIO_LOW)
    pwmA.ChangeDutyCycle(0)
    pwmB.ChangeDutyCycle(0)

#
# SERVO CONTROL
#
global servo_pos

def servo_izquierda():
    """
    Moves the servo incrementally to the left, respecting limit boundaries
        .
    """
    global servo_pos
    servo_pos = max(-1, servo_pos - 0.1)
    servo.ChangeDutyCycle(servo_pos)

def servo_derecha():
    """
    Moves the servo incrementally to the right, respecting limit
       boundaries.
    """
    global servo_pos
    servo_pos = min(1, servo_pos + 0.1)
    servo.ChangeDutyCycle(servo_pos)

def servo_valor(valor):
    """
    Directly assigns a PWM value to set the servo to a specific angle.
    Expected range: ~2.5 to 12.5 (0   to 180  ).
    """
    global servo_pos
```

```
    servo_pos = valor
    servo.ChangeDutyCycle(servo_pos)

#
# SPEED CONTROL
#
def aumentar_velocidad():
    """
    Increases motor speed by 10%, up to a maximum of 100%.
    """
    global VELOCIDAD
    VELOCIDAD = min(100, VELOCIDAD + 10)
    print(f"Speed: {VELOCIDAD}")

def disminuir_velocidad():
    """
    Decreases motor speed by 10%, down to a minimum of 0%.
    """
    global VELOCIDAD
    VELOCIDAD = max(0, VELOCIDAD - 10)
    print(f"Speed: {VELOCIDAD}")

#
# TCP SERVER FOR REMOTE CONTROL
#
HOST = '0.0.0.0'   # Bind to all network interfaces
PORT = 5000

def servidor():
    """
    Initializes a TCP server that receives control commands from a remote
       PC.
    The robot executes movement and servo instructions based on received
       keys.
    """
    global VELOCIDAD

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(1)
    print("Waiting for connection from PC...")

    try:
        conn, addr = server.accept()
        print(f"Connected by {addr}")

        while True:
            data = conn.recv(1024).decode().strip()
            if not data:
                break

            print(f"Command received: {data}")

            # Execute command based on received character
```

```
            if data == 'w':
                adelante(VELOCIDAD)
            elif data == 's':
                atras(VELOCIDAD)
            elif data == 'a':
                izquierda(VELOCIDAD)
            elif data == 'd':
                derecha(VELOCIDAD)
            elif data == 'x':
                detener()
            elif data == 'e':
                servo_derecha()
            elif data == 'q':
                servo_izquierda()
            elif data == 'h':
                aumentar_velocidad()
            elif data == 'l':
                disminuir_velocidad()

    except:
        pass

    finally:
        print("Closing connection...")
        conn.close()
        server.close()
        detener()
        pwmA.stop()
        pwmB.stop()
        GPIO.cleanup()

#
# PROGRAM ENTRY POINT
#
if __name__ == "__main__":
    servidor()
```

Listing 1: Robot control script (Python/RPi.GPIO)

5 Autopilot Attempt and Sensor Issues

An early goal was fully autonomous navigation, but the project shifted to a hybrid mode: teleoperation with optional sensor-based autopilot. The autopilot system used one HC-SR04 ultrasonic sensor and two Sharp IR sensors.

Despite calibration attempts, the system produced unstable distance measurements and inconsistent detection ranges. This led the robot to interpret false obstacles. As a result, the autopilot issued contradictory commands, causing the robot to spin rapidly in circles.

I was not able to resolve these issues during this stage of the project.

```
import RPi.GPIO as GPIO
import time
```

```
import busio  # SPI communication interface
import board  # Access to Raspberry Pi pin definitions
from adafruit_mcp3xxx.mcp3008 import MCP3008  # ADC driver
from adafruit_mcp3xxx.analog_in import AnalogIn  # Analog input interface

#
# GPIO AND SPI CONFIGURATION
#
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# HC-SR04 ultrasonic sensor pins
TRIG = 24
ECHO = 23

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# MCP3008 ADC configuration using SPI
spi = busio.SPI(clock=board.D18, MISO=board.MISO, MOSI=board.MOSI)
cs = digitalio.DigitalInOut(board.D8)  # Chip Select (GPIO8)
mcp = MCP3008(spi, cs)  # Initialize ADC

# Sharp IR sensor channels (analog     ADC)
chan_ir_izq = AnalogIn(mcp, board.D0)  # ADC Channel 0
chan_ir_der = AnalogIn(mcp, board.D1)  # ADC Channel 1

#
# SENSOR READING FUNCTIONS
#
def leer_ultrasonico():
    """Measures distance using the HC-SR04 ultrasonic sensor."""
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    pulso_inicio = time.time()
    pulso_fin = time.time()

    while GPIO.input(ECHO) == 0:
        pulso_inicio = time.time()

    while GPIO.input(ECHO) == 1:
        pulso_fin = time.time()

    duracion = pulso_fin - pulso_inicio
    distancia = (duracion * 34300) / 2  # cm
    return distancia

def voltaje_a_distancia_sharp(v):
    """Converts Sharp GP2Y0A21YK0F voltage to distance in cm."""
    if v <= 0.1:
        return 80  # Out of range
```

```
    distancia = 27.86 / (v - 0.42)  # Empirical model

    if distancia > 80:
        return 80
    elif distancia < 10:
        return 10
    return distancia

#
# MAIN LOOP
#
try:
    print("Starting sensor test... (Ctrl+C to exit)\n")
    time.sleep(1)

    while True:
        dist_ultra = leer_ultrasonico()

        volt_izq = chan_ir_izq.voltage
        volt_der = chan_ir_der.voltage

        dist_izq = voltaje_a_distancia_sharp(volt_izq)
        dist_der = voltaje_a_distancia_sharp(volt_der)

        print(f"Front: {dist_ultra:5.1f} cm | Left (V:{volt_izq:.2f} | D:{
            dist_izq:5.1f} cm) | Right (V:{volt_der:.2f} | D:{dist_der:5.1f
            } cm)")

        time.sleep(0.5)

except KeyboardInterrupt:
    print("\n[N] Ending sensor test...")
finally:
    GPIO.cleanup()
```

Listing 2: Sensor acquisition module using HC-SR04 and Sharp IR sensors via MCP3008

6 Neural Network and Vision System

The goal was to train a custom neural network without relying on pretrained models. Two datasets were collected: 1) images without the target object, and 2) images with the object present.

However, the custom network consistently failed to generalize, oscillating between always detecting an object or never detecting one. Due to limited experience and the complexity of tuning such a model, development stalled.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
```

```
#
# DATA LOADING AND PREPROCESSING
#

# Path containing transformed image data
path = "C:/Users/bryan/Downloads/Transformed_data"

# Transform: converts images to tensors (scales to [0,1])
transform = transforms.ToTensor()

# Load dataset from folders: each subfolder = class label
dataset = datasets.ImageFolder(root=path, transform=transform)

# Split ratios: 80% train, 10% validation, 10% test
train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size

# Random split into subsets
train_dataset, val_dataset, test_dataset = random_split(dataset,
                                                        [train_size,
                                                           val_size,
                                                           test_size])

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset,   batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset,  batch_size=32, shuffle=False)

# Debug: Check batch shape
images, labels = next(iter(train_loader))
print("Batch size:", images.shape)

#
# MODEL DEFINITION (Convolutional Neural Network)
#
class CNNet(nn.Module):
    def __init__(self):
        super().__init__()

        # Feature extraction: 3 convolutional blocks
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
```

```
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )

        # Classifier: fully connected layers
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128*8*8, 256),  # Adjust if image size differs
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 2)          # Binary classification
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

#
# TRAINING CONFIGURATION
#

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNNet().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.00001)
epochs = 20

#
# TRAINING LOOP
#
for epoch in range(epochs):
    model.train()
    train_loss = 0.0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * images.size(0)

    # Validation phase
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
```

```
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * images.size(0)

            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    train_loss /= len(train_loader.dataset)
    val_loss /= len(val_loader.dataset)
    accuracy = correct / total

    print(f"Epoch: {epoch+1}, Train Loss: {train_loss:.4f}, "
          f"Val Loss: {val_loss:.4f}, Accuracy: {accuracy:.4f}")

#
# TESTING PHASE
#
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

print(f"Test Accuracy: {100 * correct / total:.2f}%")

# Save trained model
torch.save(model.state_dict(), "modelo_entrenado.pth")
```

Listing 3: PyTorch CNN training pipeline with dataset splitting and evaluation

So then i tried with YOLO, but results were so general, i did not need that kind of resuelts

```
import socket
import keyboard
import cv2
import pickle
import struct
import threading
import time
import torch
import torch.nn as nn
import numpy as np
import os
```

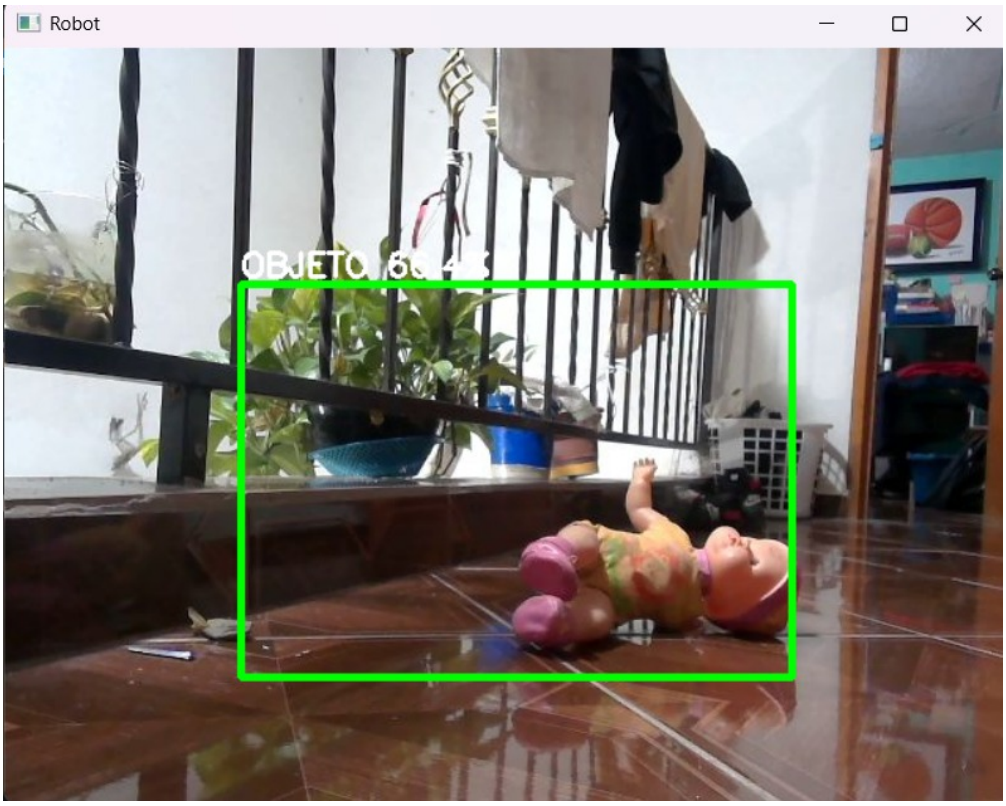

Figure 1: Output of the custom-trained Convolutional Neural Network (CNN).

```
# CONFIGURATION
HOST = "192.168.100.35"  # Raspberry Pi IP
PORT_CMD = 8000          # Command port
PORT_CAM = 8001          # Camera stream port
SAVE_PATH = "C:/Users/bryan/Desktop/codigos robot/prototipo2"

# Create directory for saved pictures
if not os.path.exists(SAVE_PATH):
    os.makedirs(SAVE_PATH)

foto_count = 0
cam_actual = (False, None)

# COMMAND CONNECTION
sock_cmd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    sock_cmd.connect((HOST, PORT_CMD))
except ConnectionRefusedError:
    print(f"[WARNING] Could not connect to robot command server at {HOST
        }:{PORT_CMD}.")
except Exception as e:
    print(f"[WARNING] Command connection error: {e}")

def send_command(cmd):
```

```
    """Send a UTF-8 encoded command to the robot."""
    try:
        sock_cmd.send(cmd.encode("utf-8"))
    except Exception:
        pass

movement_commands = {'w': 'W', 's': 'S', 'a': 'A', 'd': 'D'}
autopilot_enabled = False

def toggle_autopilot():
    """Enable or disable autopilot mode."""
    global autopilot_enabled
    autopilot_enabled = not autopilot_enabled
    state = "ENABLED" if autopilot_enabled else "DISABLED"
    print(f"[AUTO] Autopilot {state}")
    send_command("P")

def capture_photo():
    """Save current camera frame to disk."""
    global foto_count, cam_actual
    ret, frame = cam_actual
    if ret and frame is not None:
        foto_count += 1
        filename = os.path.join(SAVE_PATH, f"photo_{foto_count:03d}.jpg")
        cv2.imwrite(filename, frame)
        print(f"[PHOTO] Saved: {filename}")
        send_command(f"F{foto_count}")
    else:
        print("[PHOTO] No available frame to capture")

def robot_event_control():
    """
    Handles keyboard commands in a background thread.
    Allows manual control, autopilot switching, photo capture,
    and speed adjustment.
    """
    print("Connected to the robot. Use W/A/S/D, P, Q, +, -, ESC to exit.")
    print("Press 'P' to toggle MANUAL/AUTOMATIC mode.")
    print("Press 'Q' to take a photo.")

    keyboard.on_press_key('p', lambda e: toggle_autopilot(), suppress=True
        )
    keyboard.on_press_key('q', lambda e: capture_photo(), suppress=True)

    def manual_move(e, cmd):
        if not autopilot_enabled:
            send_command(cmd)

    def manual_stop(e):
        if not autopilot_enabled:
            send_command("X")

    # Register movement keys
    for key, command in movement_commands.items():
```

```
        keyboard.on_press_key(key, lambda e, c=command: manual_move(e, c),
            suppress=True)
        keyboard.on_release_key(key, manual_stop, suppress=True)

    # Speed control
    keyboard.on_press_key('+', lambda e: send_command('+'), suppress=True)
    keyboard.on_press_key('-', lambda e: send_command('-'), suppress=True)

    # Keep thread alive indefinitely
    threading.Event().wait()


# FRAME PROCESSING
def process_frame(frame):
    """Resize frame, send through neural network, and return label +
       probability."""
    img = cv2.resize(frame, (image_height, image_width))
    img = img.astype(np.float32) / 255.0
    img_tensor = torch.tensor(img).permute(2,0,1).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(img_tensor)
        prob = torch.sigmoid(output).item()

    label = "OBJECT" if prob > 0.5 else "NO OBJECT"
    return label, prob

# VIDEO STREAM RECEIVER
def receive_video_with_capture():
    """Receives video stream from robot, processes each frame, overlays
       labels and boxes."""
    global cam_actual
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        client_socket.connect((HOST, PORT_CAM))
    except ConnectionRefusedError:
        print(f"[ERROR] Could not connect to camera stream at {HOST}:{
           PORT_CAM}.")
        return

    data = b""
    payload_size = struct.calcsize("Q")

    cv2.namedWindow("Robot View", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Robot View", 640, 480)

    while True:
        try:
            # === Receive frame size ===
            while len(data) < payload_size:
                packet = client_socket.recv(4096)
                if not packet:
                    print("[INFO] Video connection closed by robot.")
```

```
                return
            data += packet

        packed_size = data[:payload_size]
        data = data[payload_size:]
        msg_size = struct.unpack("Q", packed_size)[0]

        # === Receive frame data ===
        while len(data) < msg_size:
            data += client_socket.recv(4096)

        frame_data = data[:msg_size]
        data = data[msg_size:]

        frame = cv2.imdecode(np.frombuffer(frame_data, np.uint8), cv2.
            IMREAD_COLOR)
        if frame is None:
            continue

        cam_actual = (True, frame.copy())

        # === Process frame with ML ===
        label, prob = process_frame(frame)
        percentage = prob * 100

        color = (0,255,0) if label == "OBJECT" else (0,0,255)
        text = f"{label} Conf: {percentage:.1f}%"

        if label == "OBJECT":
            x_min, y_min = 150, 150
            x_max, y_max = 500, 400

            cv2.rectangle(frame, (x_min, y_min), (x_max, y_max), color
                , 3)

            (w, h), _ = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX
                , 0.7, 2)
            cv2.rectangle(frame, (x_min, y_min - h - 5), (x_min + w,
                y_min), color, -1)
            cv2.putText(frame, text, (x_min, y_min - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255),
                            2)
        else:
            cv2.putText(frame, text, (20, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

        cv2.imshow("Robot View", frame)

        # Check ESC key (must run in main thread)
        if cv2.waitKey(1) & 0xFF == 27:
            break

    except Exception as e:
        print(f"[STREAM ERROR] {e}")
```

```
            break

    client_socket.close()
    cv2.destroyAllWindows()


# MAIN THREAD (CORRECTED LOGIC)
if __name__ == "__main__":

    # Start keyboard control on secondary thread
    control_thread = threading.Thread(target=robot_event_control)
    control_thread.daemon = True
    control_thread.start()

    # Video stream must run on the main thread
    receive_video_with_capture()

    print("Exiting main program.")
```

Listing 4: Client Control + Video Stream Script (Translated and Documented)

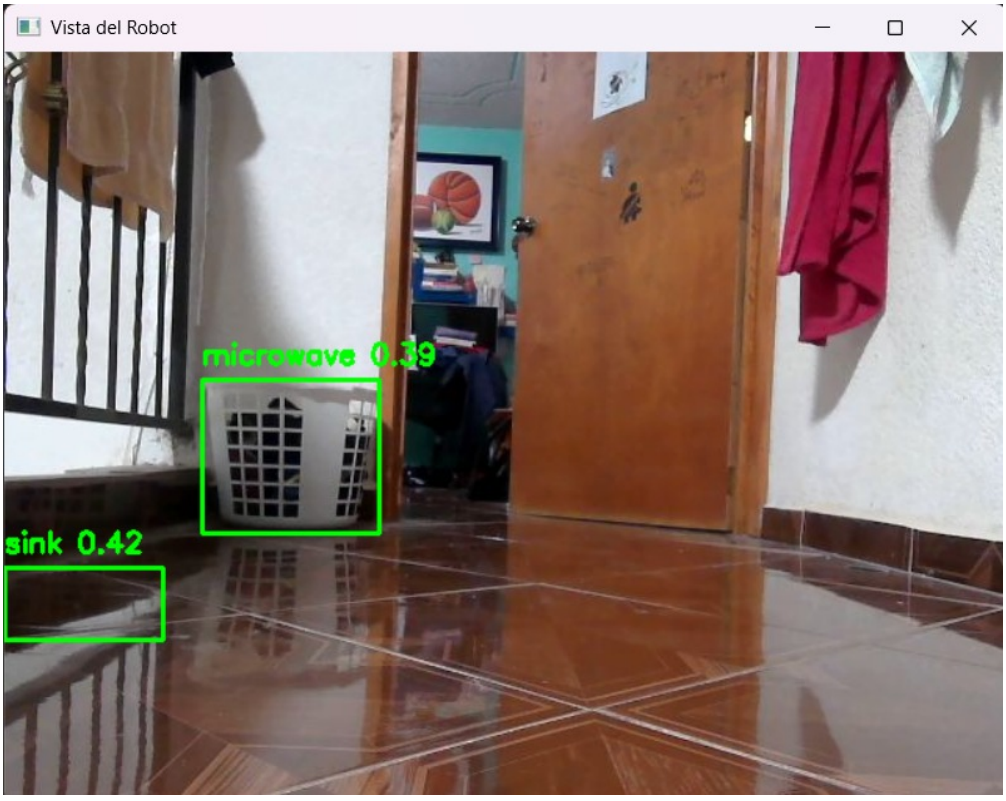

Figure 2: Result of the general-purpose YOLO detection attempt.

Given these limitations, a pretrained model was adopted. Using OpenAI's vision API, the system was adapted to classify whether an object was in view and highlight it with a green bounding box, or return "no object" if uncertain.

```python
import torch
import clip
from PIL import Image
import cv2
import numpy as np

# Initial Configuration
# Select the available device (GPU preferred)
device = "cuda" if torch.cuda.is_available() else "cpu"

# Load CLIP model (ViT-B/32) and preprocessing routine
model, preprocess = clip.load("ViT-B/32", device=device)
print("CLIP loaded successfully")

# Text labels to classify against
text = clip.tokenize(["a bottle", "nothing"]).to(device)

# Open default camera
cap = cv2.VideoCapture(0)

# Main Loop: Real-Time Inference
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convert frame to PIL image (required by CLIP transform)
    img_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))

    # Apply CLIP preprocessing
    image = preprocess(img_pil).unsqueeze(0).to(device)

    # CLIP Inference
    with torch.no_grad():
        logits_per_image, _ = model(image, text)
        probs = logits_per_image.softmax(dim=-1).cpu().numpy()[0]

    prob_bottle, prob_none = probs

    # Determine Detection Result
    if prob_bottle > prob_none:
        label = "OBJECT DETECTED"
        color = (0, 255, 0)  # Green
        # Draw a bounding rectangle around the entire frame
        cv2.rectangle(frame, (5, 5),
                      (frame.shape[1] - 5, frame.shape[0] - 5),
                      color, 4)
    else:
        label = "NO OBJECT"
        color = (0, 0, 255)  # Red

    # Display label and confidence
    cv2.putText(frame, f"{label} ({prob_bottle*100:.1f}%)",
```

```
                (30, 50), cv2.FONT_HERSHEY_SIMPLEX,
                1, color, 2)

    # Show processed frame
    cv2.imshow("CLIP Detector", frame)

    # Exit on key 'o'
    if cv2.waitKey(1) & 0xFF == ord('o'):
        break

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

Listing 5: Real-time Object Detection using CLIP

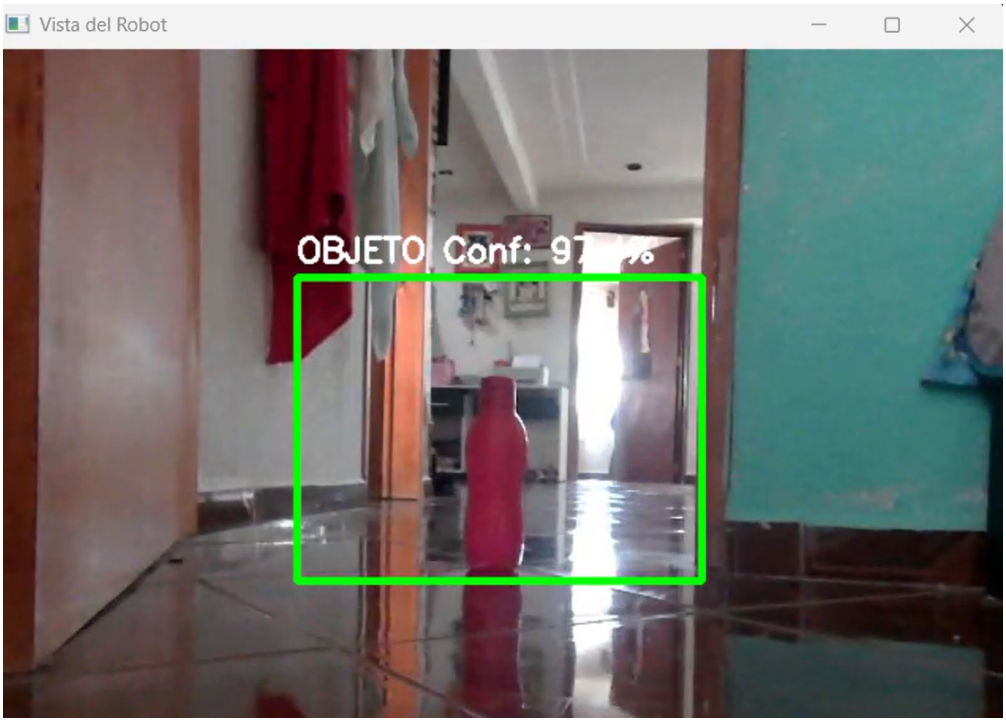

Figure 3: **Figure 1:** First result with CLIP

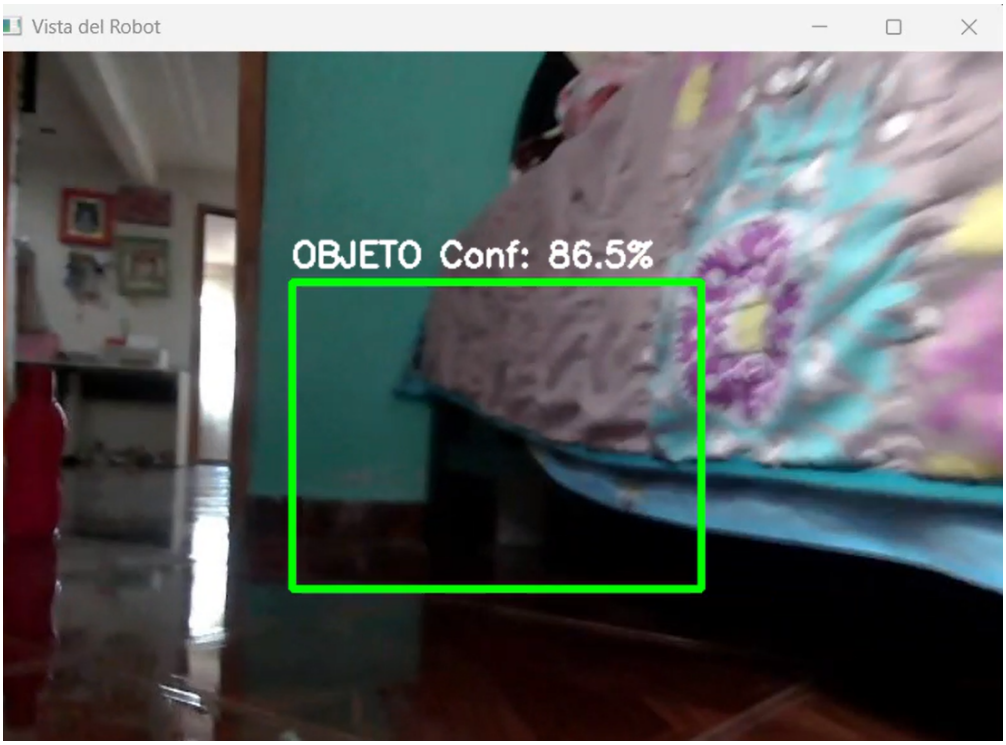

Figure 4: **Figure 2:** We can see, some error frame because it's not aiming to the object

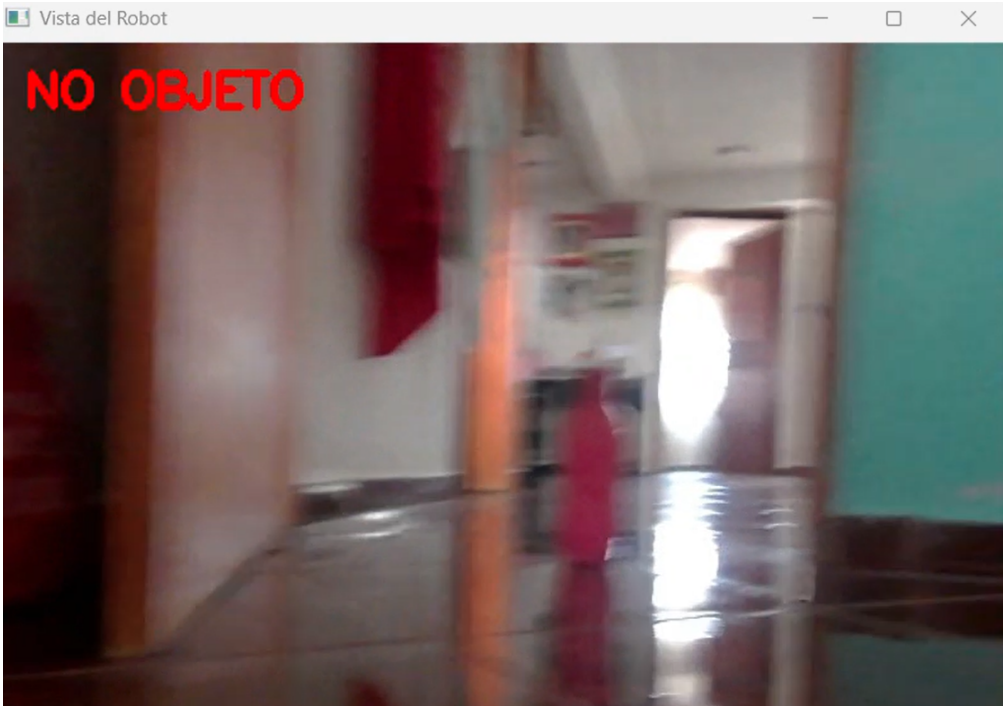

Figure 5: **Figure 3:** Just when we move the robot, in a way that was not able to see something in concrete it changes to "No object"

7 Conclusion

This project marked one of my first comprehensive attempts to combine mechanical design, electronics, embedded control, and AI into a single robot. Several components did not work as expected, including the servo system on the Raspberry Pi, the autonomous navigation logic, and the initial neural network. However, each difficulty highlighted areas in which I need to strengthen my skills.

These challenges were learning milestones: they taught me the importance of mechanical design, stable power distribution, real-time signal processing, and correct neural network training. Despite the obstacles, the final robot was significantly more robust than my previous attempt, and I successfully integrated a pretrained model for real-time object detection.

This project is not an endpoint but a foundation. In future work, I plan to improve motor control, servo integration, sensor fusion, neural network architecture, and real-time processing on the Raspberry Pi. Each new project will reflect the natural progress of an autodidactic learning process that is just beginning to take shape.