AI & SUSTAINABILITY IN VET EDUCATION

Name of the Author/s: Sucessos Criativos, Lda | P. Porto

Title of WP: WP3 – AI-SAVES@ProjectHub

Activity Reference: 3.2 – Project template for AI-SAVES@ProjectHub

European Vocational Training Association

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

1. Project Title: 2

2. Project Category: 2

3. Project Description: 2

5. Target Audience: 2

6. Pre-Requisite knowledge: 2

7. Project Objectives: 3

8. Project Timeline: 3

9. Resources Provided: 3

10. Tools and Technologies Used: 4

11. Step-by-Step Implementation Plan: 6

12. Evaluation Criteria: 30

13. Languages Available: 30

14. Contributors: 30

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1. Project Title:

Smart Waste Classification System with IoT Integration and Computer Vision

---

2. Project Category:

- AI in IoT

---

3. Project Description:

This project develops an intelligent waste management system that combines IoT sensors, computer vision, and artificial intelligence to automatically detect, classify, and manage waste disposal. The system uses a Raspberry Pi equipped with ultrasonic sensors, light detection, LED indicators, and YOLOv11 neural network to create a comprehensive smart waste classification solution. The system features motion detection, automatic lighting in low-light conditions, real-time waste categorization with visual feedback through an OLED display, and efficient resource management. Students will learn to integrate multiple IoT components while building practical AI applications that address real-world environmental challenges.

---

4. Keywords:

Computer Vision, YOLOv11, IoT Sensors, Smart Waste Management, Environmental AI, Raspberry Pi, Motion Detection, Real-time Classification, OLED Display, LED Control, Ultrasonic Sensors, Edge Computing, Sustainable Technology

---

5. Target Audience:

Both Educators and Students

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

6. Pre-Requisite knowledge:

- Intermediate Python programming (classes, functions, threading, exception handling)
- Basic understanding of IoT concepts and sensor integration
- Familiarity with computer vision and machine learning fundamentals
- Experience with Raspberry Pi GPIO programming and hardware interfaces
- Knowledge of I2C, SPI communication protocols
- Understanding of object detection and classification concepts
- Basic electronics knowledge (sensors, LEDs, breadboard wiring)
- Linux command line operations and virtual environment management

---

7. Project Objectives:

- Introduce students to comprehensive IoT system development with multiple sensor integration
- Teach real-time computer vision and AI model deployment on edge devices
- Develop practical skills in multi-threaded programming and resource management
- Demonstrate sustainable technology solutions for environmental challenges
- Build expertise in sensor fusion and intelligent decision-making systems
- Learn professional system architecture with proper error handling and cleanup
- Understand display interfaces and user feedback systems
- Encourage development of smart city and environmental monitoring solutions

---

8. Project Timeline:

---

9. Resources Provided:

- **Raspberry Pi 5** with power supply
- **Pi Camera Module v2** with ribbon cable
- **HC-SR04 Ultrasonic Sensor** for distance measurement
- **LDR (Light Dependent Resistor)**
- **SSD1306 OLED Display** (128x64, I2C interface)

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **WS2812B LED Strip** (12 pixels, NeoPixel compatible)
- **Breadboard and Jumper Wires** (Male-to-Female, Female-to-Female)
- **Pre-trained YOLOv11 Model** (yolov11_garbage.pt) for waste classification
- **Complete Python Implementation** with multi-threading support
- **Configuration Templates** with customizable parameters
- **Comprehensive Setup Guide** with step-by-step instructions
- **Troubleshooting Documentation** with common issues and solutions
- **Technical Documentation** explaining system architecture
- **Code Comments** with detailed explanations
- **Circuit Diagrams** and wiring instructions
- **Extension Ideas** for advanced implementations

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

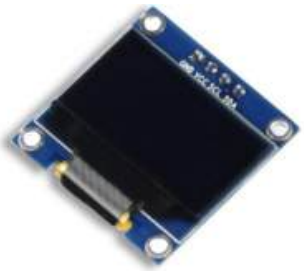
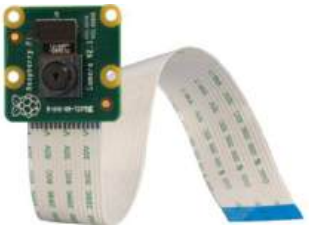
10. Tools and Technologies Used:

10.1 Hardware:

MATERIAL	
Raspberry Pi 5	
27W USB Type-C Power Supply for Raspberry Pi 5	
Flat Jumper Wires F/F	
Breadboard	
SINGLE-FILE WIRE FOR BREADBOARDS	
WS2812 RGB Programmable LED Strip 15x LED	
HC-SR04P - ULTRASONIC SENSOR 3V~5V	

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

LDR Sensor Module with Potentiometer	
0.95inch OLED i2c Display Blue/Yellow SSD1306	
Raspberry Pi Camera Board V2 8MP 1080p (IM8)	
Official FPC camera cable for Raspberry Pi 5 200mm (IM11)	

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

10.2 Software:

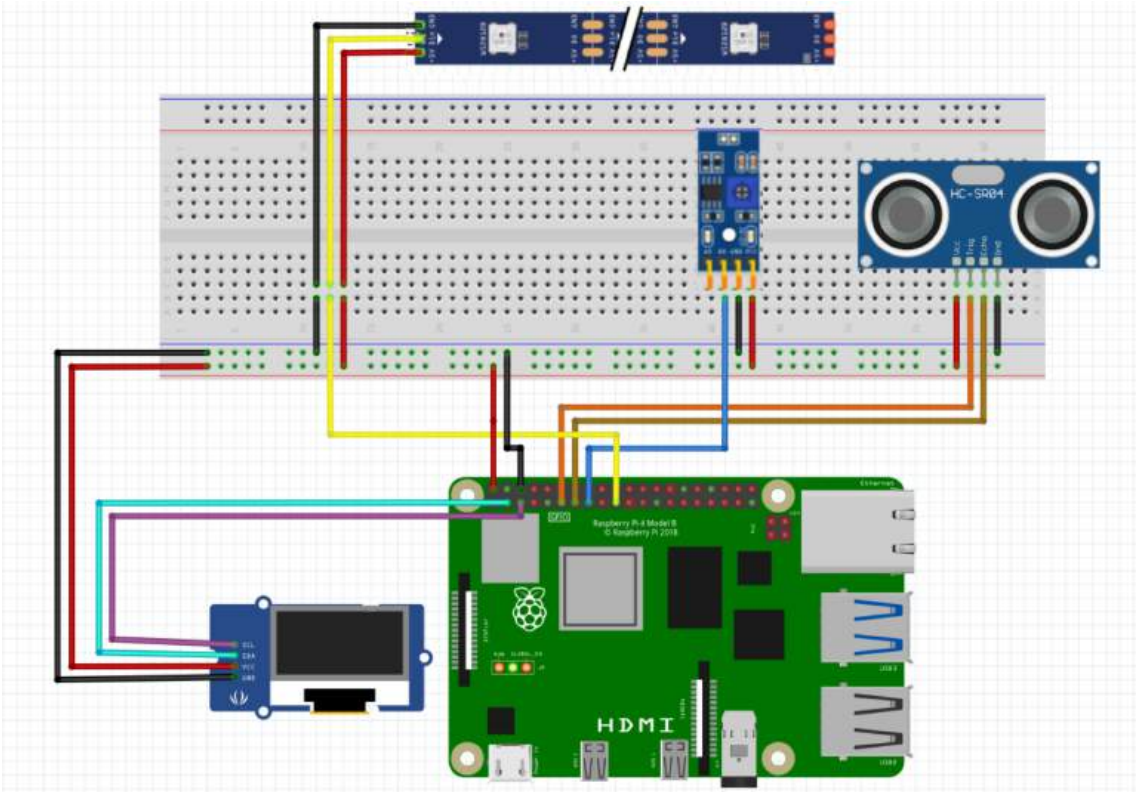
- **Operating System:** Raspberry Pi OS
- **Programming Language:** Python 3.8+
- **Development Environment:** Thonny IDE or VS Code
- **Model Training Platform:** Google Colab with GPU acceleration
- **AI Framework:** YOLOv11 (You Only Look Once v11)
- **Model Format:** PyTorch (.pt)

Libraries: time, cv2, YOLO, PiCamera2, os, kagglehub, ultralytics.YOLO, shutil, yaml, pathlib.Path, cv2, numpy, sklearn, glob, matplotlib.pyplot, seaborn, PIL.Image, PIL.ImageDraw, PIL.ImageFont, pandas, json, datetime, torch, tqdm

---

11. Step-by-Step Implementation Plan:

11.1 Hardware Setup



Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Circuit Connections

HC-SR04 Ultrasonic Sensor:

- VCC ⟶ 5V (Pin 4)
- GND ⟶ Ground (Pin 6)
- TRIGGER ⟶ GPIO 17 (Pin 11)
- ECHO ⟶ GPIO 27 (Pin 13)

SSD1306 OLED Display (I2C):

- VCC ⟶ 5V (Pin 4)
- GND ⟶ Ground (Pin 6)
- SDA ⟶ GPIO 2/SDA (Pin 3)
- SCL ⟶ GPIO 3/SCL (Pin 5)

LDR Light Sensor:

- One leg ⟶ GPIO 22 (Pin 15)
- Other leg ⟶ Ground (Pin 6)
- Built-in pull-up resistor configuration in software

WS2812B LED Strip (NeoPixel):

- VCC ⟶ 5V (Pin 4)
- GND ⟶ Ground (Pin 6)
- DATA ⟶ SPI0 MOSI (Pin 19)
- Uses SPI interface for communication

Pi Camera Module v3:

- Connected via CSI ribbon cable to camera port
- No GPIO pins required - dedicated CSI interface

Common Connections:

- **Primary Ground:** Pin 6 (Ground) - shared by all components
- **5V Power:** Pin 4 - for ultrasonic sensor, LED strip and OLED display
- **I2C Bus:** Pins 3 (SDA) and 5 (SCL) - for OLED communication

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Wiring Steps

1. **Enable Required Interfaces**
   - Camera interface via `raspi-config`
   - I2C interface for OLED display
   - SPI interface for LED strip
2. **Connect Power Distribution**
   - Establish common ground connections
   - Connect 3.3V and 5V power rails
3. **Install Ultrasonic Sensor**
   - Mount HC-SR04 sensor for motion detection
   - Connect trigger and echo pins to GPIO 17 and 27
4. **Setup OLED Display**
   - I2C connections for status display
   - Verify with `i2cdetect -y 1` (should show 0x3C)
5. **Install Light Sensor**
   - LDR configuration with pull-up resistor on GPIO 22
   - For automatic lighting control
6. **Connect LED Strip**
   - WS2812B strip via SPI interface
   - 12 pixels for visual feedback and lighting
7. **Camera Module Installation**
   - Connect Pi Camera v3 via CSI cable
   - Test with `libcamera-hello --timeout 5000`

⚠️ Important: Before powering on the system, carefully verify all connections. Incorrect wiring can damage components or cause system instability.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

11.2 Model Training (Google Colab)

This section outlines the training process for a YOLOv11-based garbage classification model using Google Colab with GPU acceleration. The goal is to develop a lightweight model capable of classifying waste into multiple categories in real time on a Raspberry Pi.

Environment Setup

- Open Google Colab.
- Enable GPU:
  Runtime → Change runtime type → Hardware accelerator: GPU.
- Upload and open the notebook `train_model.ipynb`.
- Install required libraries and configure training parameters.

Dataset Configuration

- Dataset consists of labeled images of waste items in the following categories:
  - **Plastic**
  - **Paper**
  - **Cardboard**
  - **Glass**
  - **Trash**

- Images are sourced from open datasets (e.g., Kaggle) and organized into class-specific folders.
- Converted to YOLOv11-compatible classification format with `.txt` label files per image.

Training Process

- **Preprocessing**: Image resizing, normalization, and optional augmentation (e.g., rotation, flipping, brightness).
- **Model**: YOLOv11n (classification variant optimized for edge deployment).
- **Training execution**
- **Evaluation**: Performed automatically at each epoch, tracking classification accuracy, loss, and confusion matrix.
- **Export**: Final model saved in `.pt` format for deployment on Raspberry Pi

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Output

The resulting `yolov11_garbage.pt` model is optimized for inference on Raspberry Pi.
It can classify waste images into the defined categories in real time.
Output visualizations and training metrics can be reviewed directly within Colab.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

11.3 Software Setup

Now that the hardware is ready, it's time to implement the light tracking control system.

1. Create Virtual Environment

```
# Create project directory
mkdir smart_garbage_classifier
cd smart_garbage_classifier

# Create virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate
```

2. Install Dependencies

```
# Install requirements
pip install -r requirements.txt
```

3. Add the Code

- Copy the smart garbage classifier Python code to `main.py`
- Make sure the file is in your project directory

4. Add the model

- Copy the smart garbage classifier system model to `model/yolov11_garbage.pt`
- Make sure the file is in your project directory

5. Run the System

```
# Make sure virtual environment is activated
source venv/bin/activate

# Run the smart garbage classifier
python main.py
```

6. Stop the System

- Press Ctrl+C to stop the program safely

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

System Behavior

Startup Sequence

1. **System Initialization:** OLED display shows "SMART WASTE CLASSIFIER" startup message
2. **Hardware Verification:** All sensors and components perform self-check
3. **AI Model Loading:** YOLOv11 waste classification model initialization
4. **Monitoring Mode:** System enters continuous monitoring state
5. **Status Display:** Startup complete message shown for 3 seconds

Operational States

Standby Mode (Default):

- **Continuous Distance Monitoring:** Ultrasonic sensor measures every 0.2 seconds
- **OLED Status Display:** Shows "MONITORING..." with current distance reading
- **LED Strip Off:** Energy conservation mode
- **Camera Inactive:** Resource management - camera remains off
- **Light Level Monitoring:** LDR continuously checks ambient light conditions

Motion Detection State:

- **Trigger Condition:** Object detected within 50cm threshold
- **Immediate Response:** System activates within 0.5 seconds
- **Status Update:** OLED displays "MOTION DETECTED - ACTIVATING"
- **Automatic Lighting:** LEDs activate if dark environment detected
- **Camera Preparation:** Pi Camera initializes for image capture

Active Classification Mode:

- **Camera Operation:** Continuous frame capture for 15 seconds maximum
- **Real-time Processing:** YOLOv11 analyzes frames every 1 second
- **Classification Display:** OLED shows detected waste type and confidence
- **Visual Feedback:** LED strip provides color-coded indication
- **Resource Management:** Smart power usage with automatic timeouts

Classification Results Display

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Successful Detection:

- **Primary Display:** Waste category (PLASTIC, PAPER, CARDBOARD, etc.)
- **Confidence Score:** Percentage accuracy of classification
- **Visual Indicator:** Color-coded LED feedback (Green = High confidence)
- **Duration:** Results displayed for 5 seconds before returning to monitoring

No Detection/Low Confidence:

- **Display Message:** "UNIDENTIFIED WASTE" or "LOW CONFIDENCE"
- **LED Indication:** Yellow/Orange warning colors
- **Retry Option:** System continues monitoring for better classification
- **Timeout Protection:** Returns to standby after maximum active period

Smart Lighting System

- **Dark Environment Detection:** LDR triggers LED activation
- **Motion-Based Activation:** LEDs turn on only when object detected
- **Energy Efficiency:** 10-second timeout after motion stops
- **Brightness Adaptation:** Adjustable LED intensity based on ambient light
- **Manual Override:** System maintains lighting during active classification

Error Handling and Recovery

Sensor Failure Management:

- **Ultrasonic Sensor Error:** OLED displays "DISTANCE SENSOR ERROR"
- **Camera Malfunction:** Shows "CAMERA ERROR - CHECK CONNECTION"
- **I2C Communication Issues:** "DISPLAY ERROR" with automatic retry

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **LED Strip Problems:** Continues operation without visual feedback
- **Graceful Degradation:** System maintains core functionality

Automatic Recovery Features:

- **Sensor Reconnection:** Automatic retry every 5 seconds
- **Resource Cleanup:** Proper GPIO and camera resource management
- **Memory Management:** Efficient handling of image processing
- **Thread Safety:** Multi-threaded operation with proper synchronization

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CODE

```
import cv2
import time
import board
import lgpio
import neopixel_spi as neopixel
import adafruit_ssd1306
from picamera2 import Picamera2
from ultralytics import YOLO
from PIL import Image, ImageDraw, ImageFont
import threading
import numpy as np

# ==================== PIN CONFIGURATION ====================
# Ultrasonic sensor
TRIGGER_PIN = 17
ECHO_PIN = 27

# LDR sensor
LDR_PIN = 22

# OLED Display (I2C)
OLED_WIDTH = 128
OLED_HEIGHT = 64

# ==================== LED STRIP CONFIGURATION ====================
NUM_PIXELS = 12
PIXEL_ORDER = neopixel.GRB
LED_COLOR = 0xFFFFFF
LED_BRIGHTNESS = 0.5

# ==================== DETECTION CONFIGURATION ====================
MOTION_THRESHOLD = 50.0  # cm
LIGHT_ON_DURATION = 10.0  # seconds
MEASUREMENT_INTERVAL = 0.2
CAMERA_TIMEOUT = 15.0  # Maximum time camera stays active
CLASSIFICATION_DELAY = 1.0  # Delay between classifications

# ==================== CLASSIFICATION CONFIGURATION ====================
# Mapping from YOLO model labels to our display categories
LABEL_MAPPING = {
    'plastic': 'PLASTIC',
    'paper': 'PAPER',
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
    'cardboard': 'CARDBOARD',
    'organic': 'ORGANIC',
    'green-glass': 'GLASS'
}

class WasteClassificationSystem:
    def __init__(self):
        # Initialize GPIO
        self.h = lgpio.gpiochip_open(0)
        lgpio.gpio_claim_output(self.h, TRIGGER_PIN)
        lgpio.gpio_claim_input(self.h, ECHO_PIN)
        lgpio.gpio_claim_input(self.h, LDR_PIN)

        # Initialize LED strip
        spi = board.SPI()
        self.pixels = neopixel.NeoPixel_SPI(
            spi,
            NUM_PIXELS,
            brightness=LED_BRIGHTNESS,
            pixel_order=PIXEL_ORDER,
            auto_write=False
        )

        # Initialize OLED display
        i2c = board.I2C()
         self.oled = adafruit_ssd1306.SSD1306_I2C(OLED_WIDTH, OLED_HEIGHT, i2c,
addr=0x3C)
        self.oled.fill(0)
        self.oled.show()

        # Initialize camera and model once at startup
        self.initialize_camera_and_model()

        # State variables
        self.lights_on_until = 0
        self.camera_active_until = 0
        self.previous_distance = None
        self.last_classification = "WAITING..."
        self.classification_confidence = 0
        self.is_running = True
        self.last_classification_time = 0
        self.camera_lock = threading.Lock()
        self.camera_window_open = False
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
        # Show startup message
        self.display_startup_message()
        self.turn_off_lights()

    def display_startup_message(self):
        """Display startup message on screen"""
        image = Image.new("1", (OLED_WIDTH, OLED_HEIGHT))
        draw = ImageDraw.Draw(image)
        font = ImageFont.load_default()

        self.draw_centered_text(draw, "WASTE", font, -15)
        self.draw_centered_text(draw, "CLASSIFIER", font, 0)
        self.draw_centered_text(draw, "Starting...", font, 15)

        self.oled.image(image)
        self.oled.show()
        time.sleep(3)

    def draw_centered_text(self, draw, text, font, y_offset=0):
        """Draw horizontally centered text"""
        try:
            left, top, right, bottom = font.getbbox(text)
            font_width = right - left
            font_height = bottom - top
        except:
            # Fallback for older PIL versions
            font_width, font_height = font.getsize(text)

        x = (OLED_WIDTH - font_width) // 2
        y = (OLED_HEIGHT - font_height) // 2 + y_offset
        draw.text((x, y), text, font=font, fill=255)

    def measure_distance(self):
        """Measure distance using ultrasonic sensor"""
        try:
            lgpio.gpio_write(self.h, TRIGGER_PIN, 1)
            time.sleep(0.00001)
            lgpio.gpio_write(self.h, TRIGGER_PIN, 0)

            start_time = time.time()
            timeout = start_time + 0.1
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
            while lgpio.gpio_read(self.h, ECHO_PIN) == 0:
                start_time = time.time()
                if start_time > timeout:
                    return None

            end_time = time.time()
            while lgpio.gpio_read(self.h, ECHO_PIN) == 1:
                end_time = time.time()
                if end_time > timeout:
                    return None

            pulse_duration = end_time - start_time
            distance = (pulse_duration * 34300) / 2
            return distance

        except Exception as e:
            print(f"Error measuring distance: {e}")
            return None

    def is_dark(self):
        """Check if environment is dark"""
        try:
            return lgpio.gpio_read(self.h, LDR_PIN) == 1
        except Exception as e:
            print(f"Error reading LDR sensor: {e}")
            return False

    def detect_motion(self, current_distance):
        """Detect motion based on distance change"""
        if self.previous_distance is None:
            self.previous_distance = current_distance
            return False

        distance_change = abs(current_distance - self.previous_distance)
        self.previous_distance = current_distance

        return distance_change > 5.0

    def turn_on_lights(self):
        """Turn on LEDs"""
        try:
            for i in range(NUM_PIXELS):
                self.pixels[i] = LED_COLOR
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
            self.pixels.show()
            self.lights_on_until = time.time() + LIGHT_ON_DURATION
            print("Lights ON - Dark environment detected")
        except Exception as e:
            print(f"Error turning on lights: {e}")

    def turn_off_lights(self):
        """Turn off LEDs"""
        try:
            self.pixels.fill(0x000000)
            self.pixels.show()
        except Exception as e:
            print(f"Error turning off lights: {e}")

    def should_lights_be_on(self):
        """Check if lights should be on"""
        return time.time() < self.lights_on_until

    def initialize_camera_and_model(self):
        """Initialize camera and YOLO model once at startup"""
        try:
            print("Initializing camera...")
            self.picam2 = Picamera2()

            # Configure camera with proper settings
            config = self.picam2.create_preview_configuration(
                main={"size": (640, 480), "format": "RGB888"}
            )
            self.picam2.configure(config)
            self.picam2.start()

            # Wait for camera to stabilize
            time.sleep(2)

            # Load YOLO model with correct path
            print("Loading YOLO model...")
            self.model = YOLO("model/yolov11_garbage.pt")
            print("Camera and YOLO model initialized successfully")

        except Exception as e:
            print(f"Error initializing camera/model: {e}")
            self.picam2 = None
            self.model = None
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
    def classify_waste(self):
        """Classify waste using camera"""
        if self.picam2 is None or self.model is None:
            return

        # Limit classification frequency
        current_time = time.time()
        if current_time - self.last_classification_time < CLASSIFICATION_DELAY:
            return

        with self.camera_lock:
            try:
                # Capture frame
                frame = self.picam2.capture_array()

                if frame is None or frame.size == 0:
                    return

                # Show camera window when object is detected
                if not self.camera_window_open:
                                    cv2.namedWindow('Waste  Classifier  Camera',
cv2.WINDOW_AUTOSIZE)
                    self.camera_window_open = True

                # Run YOLO inference on the frame
                results = self.model(frame, verbose=False)

                # Create a copy of the frame for annotation
                annotated_frame = frame.copy()

                # Initialize variables to track the best detection
                best_label_text = ""
                best_category = ""
                highest_conf = 0

                # Process detection results
                for result in results:
                    if result.boxes is not None:
                        # Iterate through all detected boxes
                        for box in result.boxes:
                            # Get confidence score as percentage
                            confidence = float(box.conf[0]) * 100
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
                            # Only consider detections with confidence > 60%
and higher than current best
                        if confidence > 60 and confidence > highest_conf:
                            cls_id = int(box.cls[0])  # Get class ID
                                original_label = result.names[cls_id]  # Get
original class name from model

                                    # Map the original label to our display
category
                            if original_label in LABEL_MAPPING:
                                                        best_category  =
LABEL_MAPPING[original_label]
                                highest_conf = confidence  # Update highest
confidence
                                            print(f"Detected: {original_label} ->
Mapped to: {best_category}")  # Debug mapping

            # Format the best detection text if found
            if best_category:
                best_label_text = f"{best_category} {highest_conf:.2f}%"
                print(best_label_text)  # Print like your example

                self.last_classification = best_category
                self.classification_confidence = highest_conf
            else:
                self.last_classification = "UNIDENTIFIED"
                self.classification_confidence = 0

            # Display the detection text on the frame
            if best_label_text:
                cv2.putText(
                    annotated_frame,
                    best_label_text,
                    (30, 60),  # Position (x, y)
                    cv2.FONT_HERSHEY_SIMPLEX,  # Font type
                    1.5,  # Font scale
                    (0, 0, 255),  # Color (BGR format - red)
                    3,  # Thickness
                    cv2.LINE_AA  # Anti-aliasing
                )

            # Display the frame in a window
            cv2.imshow('Waste Classifier Camera', annotated_frame)
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
                cv2.waitKey(1)  # Refresh window

                self.last_classification_time = current_time

            except Exception as e:
                print(f"Error: {e}")
                self.last_classification = "CAMERA ERROR"
                self.classification_confidence = 0

    def update_display(self):
        """Update OLED display with current information"""
        try:
            image = Image.new("1", (OLED_WIDTH, OLED_HEIGHT))
            draw = ImageDraw.Draw(image)
            font = ImageFont.load_default()

            # Title
            draw.text((30, 2), "CLASSIFIER", font=font, fill=255)

            # Current classification
            if self.classification_confidence > 0:
                        draw.text((5, 18), f"Type: {self.last_classification}",
font=font, fill=255)
                                                draw.text((5,   32),   f"Conf:
{self.classification_confidence:.1f}%", font=font, fill=255)
            else:
                  self.draw_centered_text(draw, self.last_classification, font,
5)

            # System status
            current_time = time.time()
            if current_time < self.camera_active_until:
                draw.text((5, 46), "CAM: ACTIVE", font=font, fill=255)
            else:
                draw.text((5, 46), "CAM: IDLE", font=font, fill=255)

            # Light indicator
            if self.should_lights_be_on():
                draw.text((85, 46), "LED: ON", font=font, fill=255)
            else:
                draw.text((85, 46), "LED: OFF", font=font, fill=255)

            self.oled.image(image)
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
            self.oled.show()

        except Exception as e:
            print(f"Display update error: {e}")

    def sensor_monitoring_loop(self):
        """Main sensor monitoring loop"""
        print("Waste classification system started")
        print("Motion detection activated")

        while self.is_running:
            try:
                # Measure distance
                distance = self.measure_distance()

                if distance is not None:
                    object_detected = distance <= MOTION_THRESHOLD
                    dark_environment = self.is_dark()

                                    print(f"Distance:  {distance:.1f}cm  |  Object:
{object_detected}  |  Dark:  {dark_environment}")

                    # Activate camera if motion detected
                    if object_detected:
                        self.camera_active_until = time.time() + CAMERA_TIMEOUT

                    # Turn on lights if dark and motion detected
                    if object_detected and dark_environment:
                        self.turn_on_lights()

                # Classify waste if camera should be active
                current_time = time.time()
                  if current_time < self.camera_active_until and self.picam2 is
not None:
                    self.classify_waste()
                elif current_time >= self.camera_active_until:
                    # Reset classification when camera becomes idle
                    if self.last_classification != "WAITING...":
                        self.last_classification = "WAITING..."
                        self.classification_confidence = 0
                        print("Camera idle - waiting for motion...")

                    # Close camera window when idle
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
                if self.camera_window_open:
                    cv2.destroyAllWindows()
                    self.camera_window_open = False

            # Manage lights
            if not self.should_lights_be_on() and self.lights_on_until > 0:
                self.turn_off_lights()
                self.lights_on_until = 0
                print("Lights OFF")

            # Update display
            self.update_display()

            time.sleep(MEASUREMENT_INTERVAL)

        except Exception as e:
            print(f"Error in main loop: {e}")
            time.sleep(1)

def run(self):
    """Run the complete system"""
    try:
        self.sensor_monitoring_loop()
    except KeyboardInterrupt:
        print("\nStopping system...")
    finally:
        self.cleanup()

def cleanup(self):
    """Clean up resources"""
    print("Cleaning up resources...")
    self.is_running = False

    try:
        # Clear display
        self.oled.fill(0)
        self.oled.show()
    except:
        pass

    try:
        # Close camera window
        if self.camera_window_open:
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
                cv2.destroyAllWindows()
        except:
            pass

        try:
            # Turn off lights
            self.turn_off_lights()
        except:
            pass

        try:
            # Stop camera
            if self.picam2 is not None:
                self.picam2.stop()
                self.picam2.close()
                self.picam2 = None
        except:
            pass

        try:
            # Close GPIO
            lgpio.gpiochip_close(self.h)
        except:
            pass

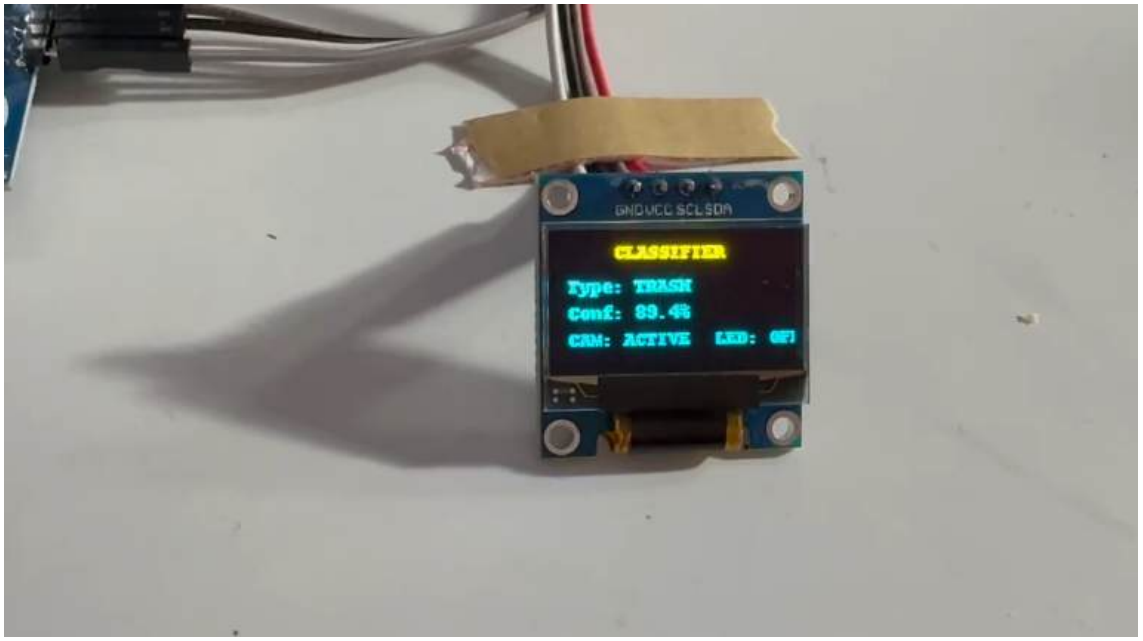
        print("System completely stopped")

# ==================== MAIN EXECUTION ====================
if __name__ == "__main__":
    system = WasteClassificationSystem()
    system.run()
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

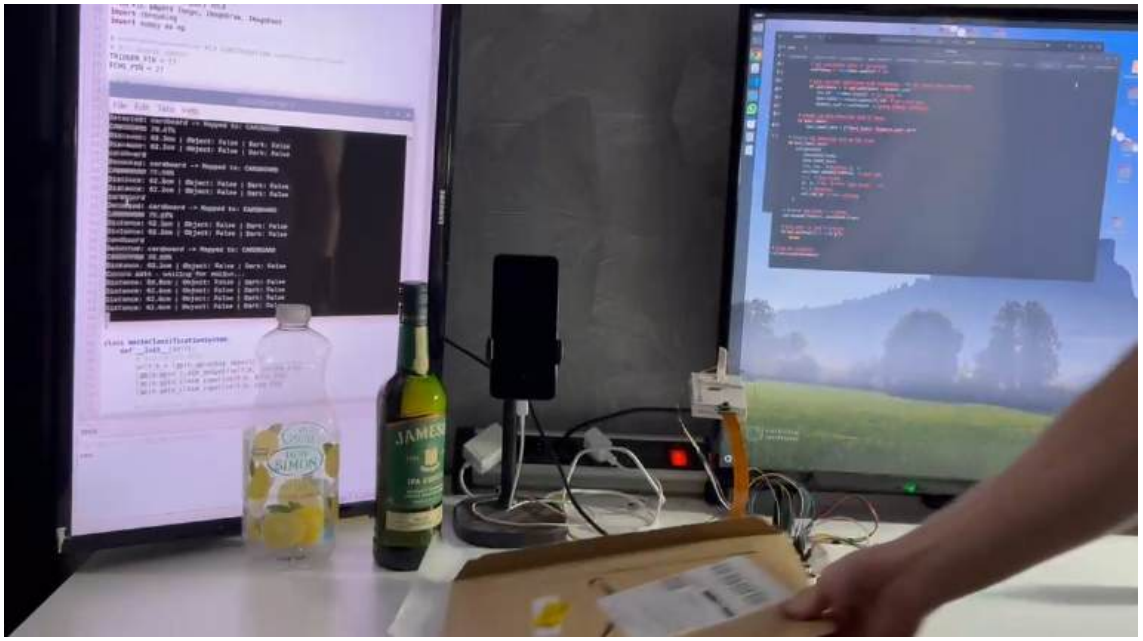
Now, you can run the program and see the result:





Co-funded by the European Union

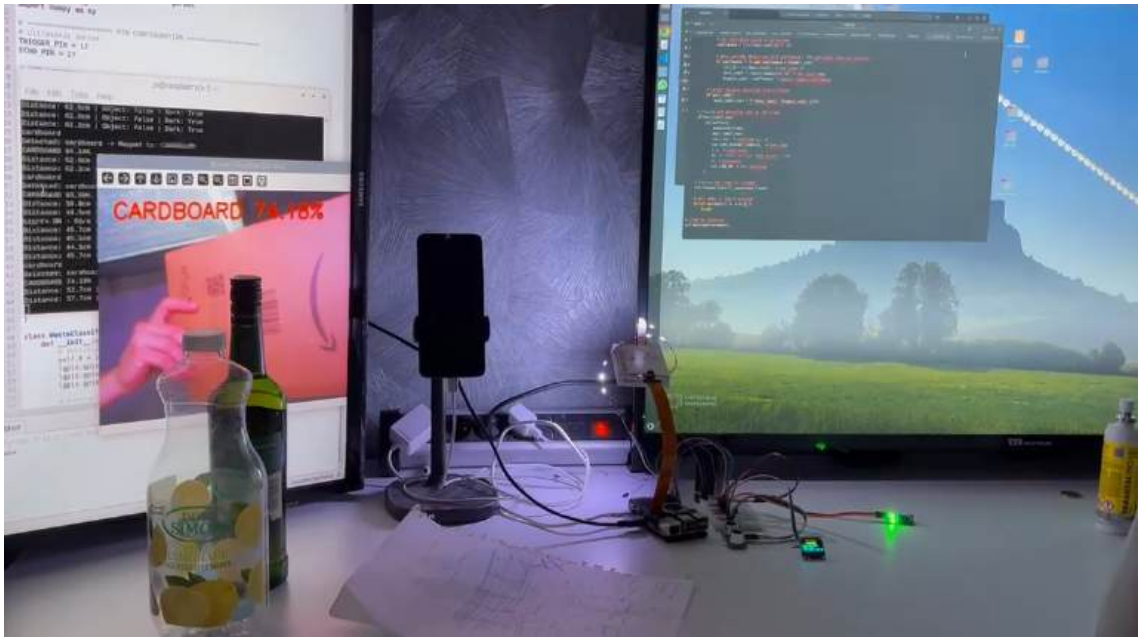
This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.





Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

12. Evaluation Criteria:

- **Hardware Setup and Integration (30%):** Correct assembly and configuration of all hardware components (ultrasonic sensor, LDR, OLED, LED strip, Pi camera). Includes enabling required interfaces, accurate wiring, and successful hardware testing.
- **Model Training Execution (30%):** Proper execution of the YOLOv11 training notebook using the provided dataset. Includes dataset loading, configuration of training parameters, error-free execution, convergence monitoring, and generation of a deployable model.
- **Software Implementation (30%):** Reliable execution of the smart waste classifier system. Includes real-time sensor monitoring, motion-triggered AI inference, OLED updates, LED feedback, and proper multi-threaded control flow.
- **Documentation and System Understanding (10%):** Clear and complete documentation of system setup, code structure, and usage instructions. Demonstrates understanding of the project's architecture and ability to troubleshoot basic issues.

---

13. Languages Available:

---

14. Contributors:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.