AI & SUSTAINABILITY IN VET EDUCATION

Name of the Author/s: Sucessos Criativos, Lda

Title of WP: WP3 – AI-SAVES@ProjectHub

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

Logo:

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: 4
2. Project Category: 4
3. Project Description: 4
4. Keywords: 4
5. Target Audience: 4
6. Project Objectives: 4
7. Project Timeline: 5
8. Resources Provided: 5
9. Tools and Technologies Used: 5
10. Step-by-Step Implementation Plan: 6
11. Evaluation Criteria: 23
12. Languages Available: 23
13. Contributors: 23
14. Feedback and Continuous Improvement: 24

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:

AI-Based Self-Checkout System with Product Recognition

---

2. Project Category:

AI in Robotics

---

3. Project Description:

This project implements a prototype AI-based self-checkout system that uses computer vision to automatically recognize products placed in front of a camera and calculate their total price. The system is designed to simulate a simplified self-service retail experience, where users manually control the scanning process using physical buttons.

When a product is placed in view of the camera, an object detection model identifies supported items such as fruits, vegetables, and packaged food. By pressing an “Add” button, all valid items currently detected in the camera frame are added to a virtual inventory. This process can be repeated multiple times, allowing users to scan products in batches. Once all items are added, a “Checkout” button displays the final purchase summary and total price. A “Reset” button clears the inventory, preparing the system for the next customer. This project demonstrates how AI-powered vision, hardware controls, and user interfaces can be combined to create intelligent retail automation systems.

---

4. Keywords:

Raspberry Pi, Object Detection, Retail Automation, Smart Systems

---

5. Target Audience:

Both Educators and Students

---

6. Project Objectives:

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.

- Detect and recognize retail products using a camera and an object detection model.
- Allow users to manually control item scanning using physical buttons.
- Maintain a virtual inventory with item counts and pricing.
- Calculate and display the total purchase price in real time.
- Provide a clear checkout and reset workflow for multiple users.
- Explore AI-driven automation concepts applied to retail environments.

---

7. Project Timeline:

With the provided resources, this project should take approximately 6-7 hours or less to be done calmly.

---

8. Resources Provided:

- Python code implementing object detection, inventory management, and checkout logic
- Demo video of the result
- Wiring diagram for buttons, LCD Display, and camera

---

9. Tools and Technologies Used:

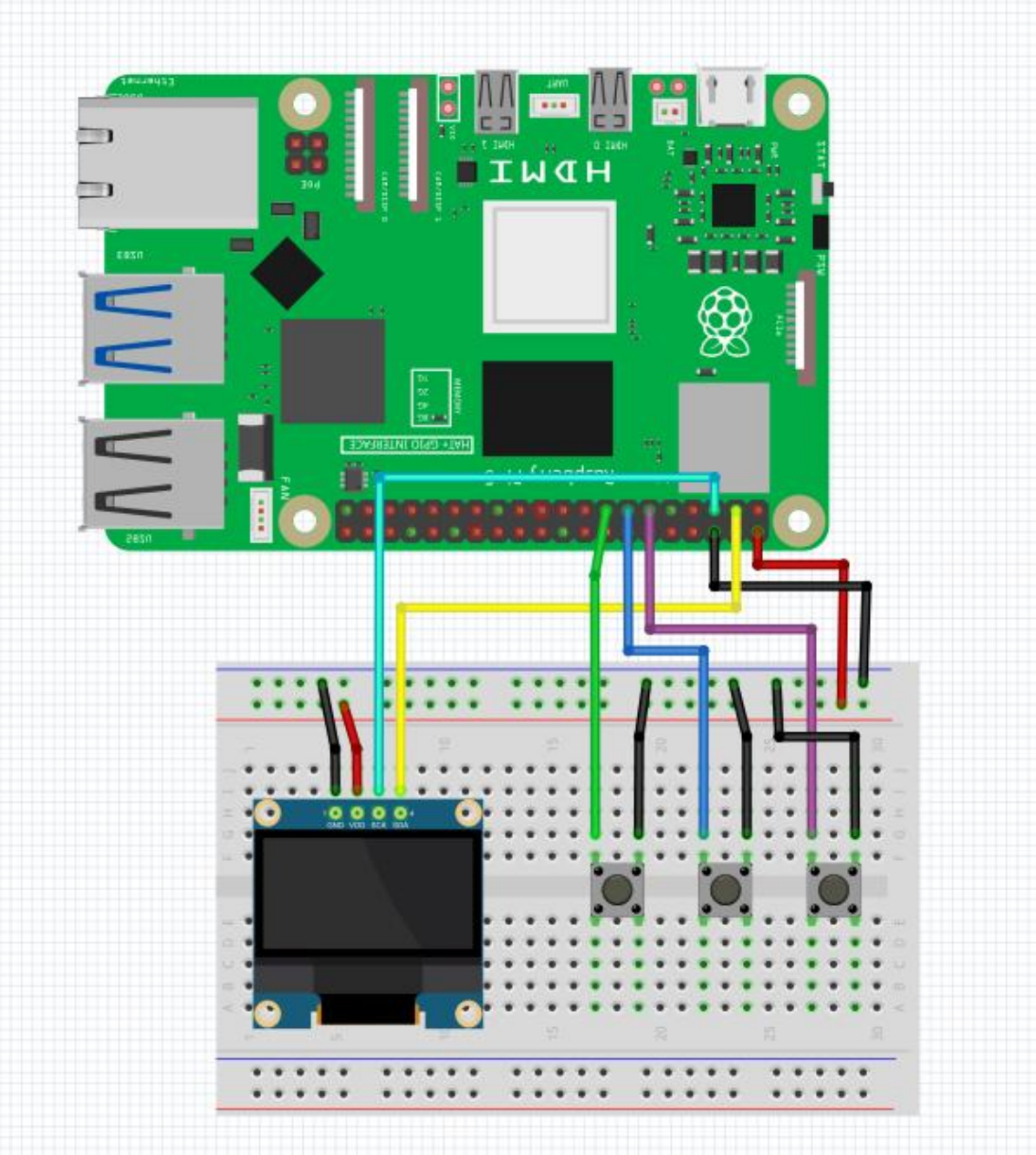
- Raspberry Pi 5
- Raspberry Pi Camera Module 4 v2
- Python programming language
- OpenCV for camera input and image processing
- Pre-trained Object Detection Model (YOLO11-Nano)
- OLED display (SSD1306) for user feedback
- Physical push buttons for user interaction
- GPIO libraries for hardware input control

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. Step-by-Step Implementation Plan:

1. **Hardware Preparation:** Connect Raspberry Pi, the camera module, buttons and LCD Display according to the wiring diagram.



2. **Software Setup:** Install the necessary software on Raspberry Pi, including Raspbian OS and Python libraries including OpenCV, Picamera2, adafruit_ssd1306, Ultralytics YOLO and GPIO libraries.

3. **Execute the Code:** Copy the provided code bellow and save it on your 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.

```
import cv2

import time

import threading

from picamera2 import Picamera2

from ultralytics import YOLO

import RPi.GPIO as GPIO

import board

import adafruit_ssd1306

from PIL import Image, ImageDraw, ImageFont


# ===== PRICE CONFIGURATION (EDIT HERE) =====

PRICES = {

    "apple": 0.50,

    "banana": 0.30,

    "orange": 0.40,

    "sandwich": 2.50,

    "broccoli": 1.20,

    "carrot": 0.60

}


# List of detectable objects (only these will show boxes)

DETECTABLE_OBJECTS = ["apple", "banana", "orange", "sandwich", "broccoli",
"carrot"]


# ===== PIN CONFIGURATION =====

BUTTON_ADD = 17      # Button to add items

BUTTON_FINISH = 27   # Button to finish/checkout

BUTTON_RESET = 22    # Button to reset inventory
```

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.

```
# ===== GPIO SETUP =====
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON_ADD, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(BUTTON_FINISH, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(BUTTON_RESET, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Debounce tracking
last_button_time = {"add": 0, "finish": 0, "reset": 0}
DEBOUNCE_TIME = 0.5  # seconds

# ===== OLED DISPLAY SETUP =====
WIDTH = 128
HEIGHT = 64

# Initialize I2C and OLED display
i2c = board.I2C()
oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3C)

# Clear display
oled.fill(0)
oled.show()

# Load font
try:
    font                                                    =
ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 10)
    font_small                                              =
ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 8)
```

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.

```
    font_large                                                    =
ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf",
12)

except:

    font = ImageFont.load_default()

    font_small = ImageFont.load_default()

    font_large = ImageFont.load_default()


# ===== CAMERA + YOLO SETUP =====

picam2 = Picamera2()

picam2.preview_configuration.main.size = (640, 400)

picam2.preview_configuration.main.format = "RGB888"

picam2.preview_configuration.align()

picam2.configure("preview")

picam2.start()


model = YOLO("yolo11n.pt")


# ===== INVENTORY SYSTEM =====

inventory = {}  # {item_name: count}

system_state = "READY"  # States: READY, SCANNING, CHECKOUT

scroll_offset = 0

scroll_direction = 1

camera_running = True

latest_frame = None

latest_results = None


# ===== DISPLAY FUNCTIONS =====

def getfontsize(font, text):
```

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.

```
    """Calculate text size in pixels"""
    left, top, right, bottom = font.getbbox(text)
    return right - left, bottom - top



def update_display():
    """Update OLED display based on current state"""
    global scroll_offset, scroll_direction

    # Create a blank image for drawing
    image = Image.new("1", (oled.width, oled.height))
    draw = ImageDraw.Draw(image)

    if system_state == "READY":
        # Title
        title = "SMART SELLER"
        (tw, th) = getfontsize(font_large, title)
        draw.text((oled.width // 2 - tw // 2, 2), title, font=font_large,
fill=255)

        # Instructions
        draw.text((2, 20), "Press ADD to scan", font=font_small, fill=255)
        draw.text((2, 32), "items on table", font=font_small, fill=255)

        # Line separator
        draw.line((0, 44, 128, 44), fill=255)

        # Ready message
```

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.

```
        draw.text((10, 50), "Ready to start!", font=font_small, fill=255)


    elif system_state == "SCANNING":
        # Title
        draw.text((5, 0), "SCANNING MODE", font=font_small, fill=255)
        draw.line((0, 12, 128, 12), fill=255)

        # Inventory list with scrolling
        if not inventory:
            draw.text((5, 18), "No items yet", font=font_small, fill=255)
            draw.text((5, 30), "Place item and", font=font_small, fill=255)
            draw.text((5, 40), "press ADD", font=font_small, fill=255)
        else:
            # Calculate total content height
            total_items = len(inventory)
            item_height = 10
            total_height = total_items * item_height

            # Draw items with scroll offset
            y_start = 15 - scroll_offset
            for idx, (item, count) in enumerate(inventory.items()):
                y_pos = y_start + (idx * item_height)

                # Only draw if visible on screen
                if -10 <= y_pos <= 64:
                    price = PRICES.get(item, 0.0)
                    item_total = price * count
```

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.

```
                    text = f"{item.capitalize()}: {count}x €{item_total:.2f}"
                    draw.text((2, y_pos), text, font=font_small, fill=255)

            # Draw total at bottom (always visible)
            draw.rectangle((0, 54, 128, 64), fill=0, outline=255)
            total = sum(PRICES.get(item, 0) * count for item, count in inventory.items())
            draw.text((2, 55), f"Total: €{total:.2f}", font=font_small, fill=255)

            # Update scroll for continuous loop effect
            if total_height > 40:  # Only scroll if content is larger than display area
                scroll_offset += scroll_direction
                if scroll_offset > total_height:
                    scroll_offset = 0
                elif scroll_offset < 0:
                    scroll_offset = total_height

    elif system_state == "CHECKOUT":
        # Title
        title = "CHECKOUT"
        (tw, th) = getfontsize(font_large, title)
        draw.text((oled.width // 2 - tw // 2, 2), title, font=font_large, fill=255)
        draw.line((0, 14, 128, 14), fill=255)

        if not inventory:
            draw.text((25, 30), "No items!", font=font, fill=255)
```

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.

```
        else:

            # Show summary

            total_items = sum(inventory.values())

            total_price = sum(PRICES.get(item, 0) * count for item, count in
inventory.items())



            draw.text((5,  18),  f"Items:  {total_items}",  font=font_small,
fill=255)

            draw.line((0, 30, 128, 30), fill=255)

            draw.text((5,      35),      f"TOTAL:      €{total_price:.2f}",
font=font_large, fill=255)

            draw.line((0, 48, 128, 48), fill=255)

            draw.text((15, 52), "Press RESET", font=font_small, fill=255)



    # Display the image on OLED

    oled.image(image)

    oled.show()




def display_thread():

    """Continuous display update thread"""

    while True:

        try:

            update_display()

            time.sleep(0.1)    #  Update  10  times  per  second  for  smooth
scrolling

        except Exception as e:

            print(f"Display error: {e}")

            time.sleep(0.5)
```

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 camera_preview_thread():
    """Continuous camera preview with YOLO detection"""
    global latest_frame, latest_results, camera_running

    while camera_running:
        try:
            # Capture frame
            frame = picam2.capture_array()

            # Run YOLO detection
            results = model(frame, verbose=False)

            # Store results for use by add_items function
            latest_frame = frame.copy()
            latest_results = results

            # Create annotated frame for display
            annotated_frame = frame.copy()

            # Draw detection boxes
            for result in results:
                for box in result.boxes:
                    cls_id = int(box.cls[0])
                    label = result.names[cls_id].lower()
                    confidence = box.conf[0] * 100

                    # Only show boxes for detectable objects
```

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 label in DETECTABLE_OBJECTS:
                        x1, y1, x2, y2 = map(int, box.xyxy[0])

                        # Choose color based on confidence threshold
                        if confidence > 50:
                            # Green for items that will be added
                            box_color = (0, 255, 0)
                            text_color = (0, 255, 0)
                            status = ""
                        else:
                            # Yellow/Orange for low confidence (won't be added)
                            box_color = (0, 165, 255)  # Orange
                            text_color = (0, 165, 255)
                            status = " (Too Low)"

                        # Draw bounding box
                        cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), box_color, 2)

                        # Create label text (item name, confidence, price, and status)
                        price = PRICES.get(label, 0.0)
                        label_text = f"{label.capitalize()} {confidence:.1f}% - {price:.2f}eur{status}"

                        # Draw label
                        cv2.putText(
                            annotated_frame,
                            label_text,
```


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.

```
                        (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        0.5,
                        text_color,
                        2
                    )

            # Add info text at top (no background)
            # Status
            status_text = f"State: {system_state}"
            cv2.putText(
                annotated_frame,
                status_text,
                (10, 25),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.7,
                (0, 255, 255),
                2
            )

            # Tip text
            cv2.putText(
                annotated_frame,
                "Tip: >50% confidence required to add to cart",
                (10, 50),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.45,
                (200, 200, 200),  # Light gray
```

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

            )


            # Add inventory count

            if inventory:

                total_items = sum(inventory.values())

                total_price = sum(PRICES.get(item, 0) * count for item, count
in inventory.items())

                inventory_text  =  f"Cart:  {total_items}  items  |  Total:
{total_price:.2f}eur"

                cv2.putText(

                    annotated_frame,

                    inventory_text,

                    (10, 80),

                    cv2.FONT_HERSHEY_SIMPLEX,

                    0.6,

                    (255, 255, 0),

                    2

                )


            # Display frame

            cv2.imshow("Smart Seller - Live Preview", annotated_frame)


            # Small delay

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

                camera_running = False

                break


        except Exception as e:
```

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.

```
            print(f"Camera error: {e}")
            time.sleep(0.1)




# ===== BUTTON HANDLERS =====
def button_pressed(button_name):
    """Check if button was pressed with debouncing"""
    current_time = time.time()
    if current_time - last_button_time[button_name] < DEBOUNCE_TIME:
        return False
    last_button_time[button_name] = current_time
    return True



def add_items():
    """Scan and add items to inventory"""
    global system_state, scroll_offset, latest_frame, latest_results

    print("\n> SCANNING for items...")
    system_state = "SCANNING"
    scroll_offset = 0

    # Use the latest frame and results from camera preview
    if latest_frame is None or latest_results is None:
        print("X Camera not ready, please wait...")
        return
```

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.

```
    items_found = []


    # Process all detections from cached results
    for result in latest_results:
        for box in result.boxes:
            cls_id = int(box.cls[0])
            label = result.names[cls_id].lower()


            # Check if item is in our detectable objects list
            if label in DETECTABLE_OBJECTS:
                confidence = box.conf[0] * 100


                # Only count if confidence is reasonable
                if confidence > 50:
                    items_found.append(label)


    # Update inventory
    if items_found:
        for item in items_found:
            inventory[item] = inventory.get(item, 0) + 1


        print(f"✓ Added: {', '.join(items_found)}")
        print(f"Current inventory: {inventory}")
    else:
        print("✗ No valid items detected")
```

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 finish_checkout():

    """Finish and show final checkout"""

    global system_state



    print("\n> CHECKOUT")

    system_state = "CHECKOUT"



    if inventory:

        total_items = sum(inventory.values())

        total_price = sum(PRICES.get(item, 0) * count for item, count in
inventory.items())



        print("="*40)

        print("FINAL RECEIPT")

        print("="*40)

        for item, count in inventory.items():

            price = PRICES.get(item, 0.0)

            item_total = price * count

            print(f"{item.capitalize():15}  {count}x     €{price:.2f}     =
€{item_total:.2f}")

        print("-"*40)

        print(f"{'TOTAL':15}          €{total_price:.2f}")

        print("="*40)

    else:

        print("No items in cart!")




def reset_system():

    """Reset inventory and return to ready state"""
```

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.

```
    global inventory, system_state, scroll_offset


    print("\n> SYSTEM RESET")
    inventory.clear()
    system_state = "READY"
    scroll_offset = 0
    print("Inventory cleared. Ready for new customer.")



# ===== MAIN PROGRAM =====
print("="*50)
print("       SMART SELLER SYSTEM")
print("="*50)
print("\nPrice List:")
for item, price in PRICES.items():
    print(f"  {item.capitalize():15} €{price:.2f}")
print("\n" + "="*50)
print("CONTROLS:")
print(f"  GPIO {BUTTON_ADD:2d} - Add items to cart")
print(f"  GPIO {BUTTON_FINISH:2d} - Finish & checkout")
print(f"  GPIO {BUTTON_RESET:2d} - Reset system")
print("="*50)
print("\nPress Ctrl+C to exit\n")


# Start display update thread
display_updater = threading.Thread(target=display_thread, daemon=True)
display_updater.start()
```

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.

```
# Start camera preview thread

camera_previewer      =      threading.Thread(target=camera_preview_thread,
daemon=True)

camera_previewer.start()



print("> Camera preview started. Press 'q' in preview window to quit.")



try:

    while camera_running:

        # Check ADD button

        if GPIO.input(BUTTON_ADD) == GPIO.LOW:

            if   button_pressed("add")   and   system_state   in   ["READY",
"SCANNING"]:

                add_items()



        # Check FINISH button

        if GPIO.input(BUTTON_FINISH) == GPIO.LOW:

            if button_pressed("finish") and system_state == "SCANNING":

                finish_checkout()



        # Check RESET button

        if GPIO.input(BUTTON_RESET) == GPIO.LOW:

            if button_pressed("reset"):

                reset_system()



        time.sleep(0.01)



except KeyboardInterrupt:

    print("\n> Program interrupted by user")
```

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.

```
finally:
    print("> Cleaning up...")
    camera_running = False
    time.sleep(0.5)  # Give threads time to stop
    GPIO.cleanup()
    cv2.destroyAllWindows()
    picam2.stop()
    oled.fill(0)
    oled.show()
    print("> Program ended")
```

---

11. Evaluation Criteria:

- **System Functionality:** Ability to replicate the project.
- **Creativity:** Suggestions for improvements (new features, new objects detected, etc.). If suggested and tried (with success or failure) should be more rewarded.
- **Detection Accuracy:** Correct identification of supported products with sufficient confidence.
- **System Stability:** Smooth operation of camera preview, display updates, and threading logic.

---

12. Languages Available:

English

---

13. Contributors:

- **Fátima Pais, Luís Soares** – Project Designers and Facilitators

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.

14. Feedback and Continuous Improvement:

Describe how feedback will be collected from participants and how this feedback will be used to improve future projects or iterations of this project.

- **Post-Implementation Survey:** After deploying the project, feedback will be collected through surveys targeting participants involved in the testing and observation process.
- **Team Reflection Sessions:** The project team will organize internal review sessions to discuss challenges faced, what worked well, and areas that need refinement.
- **Technical Evaluation:** Observations regarding detection accuracy, system stability, and hardware durability will be documented for improving future versions.
- **Iterative Updates:** Based on feedback and performance, updates to the software code and system design will be implemented to improve reliability and effectiveness.

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.