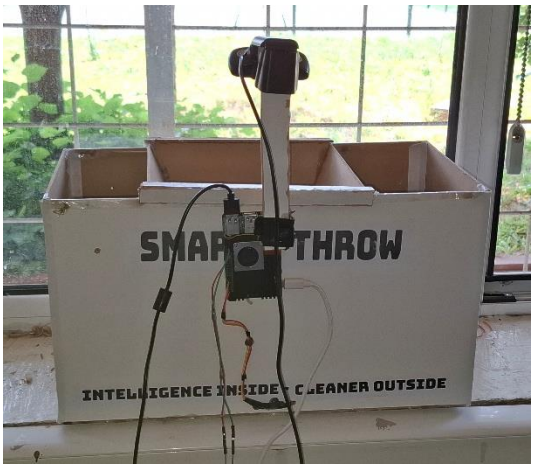
SmartThrow – intelligence inside, cleaner outside

SmartThrow is a trash can that operates automatically, recognizing the type of waste that is thrown into it and sorts it into the appropriate compartment. With this project, our goal is to reduce pollution and make the recycling process easier and more efficient. When a user throws waste into the bin, an integrated camera scans the item and, using a pre-trained database, identifies the type of waste. Then, a moving mechanism transfers the waste to the correct compartment and releases it, after which it returns to its starting position.





OBJECTIVES

- *Enhance practical knowledge in AI, computer vision, and IoT systems through hands-on work.*
- *Improve collaboration and teamwork by working together across hardware, software, and design roles.*
- *Develop problem-solving and critical thinking skills during the construction, testing, and troubleshooting phases.*
- *Strengthen creativity and innovation by designing a unique solution for real-world environmental challenges.*
- *Gain experience in project documentation and presentation, useful for future academic or professional pursuits.*

TARGET LEVEL

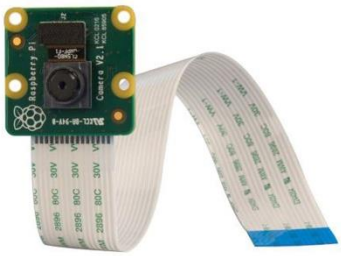
- *Secondary school students engaged in STEM-oriented vocational programs*
- *Learners specializing in Automation, Robotics, or Embedded Systems*
- *Tech enthusiasts or student clubs with foundational skills in coding and circuit design*

PRE-REQUISITE KNOWLEDGE:

- *Basic understanding of Python programming*
- *Familiarity with electronic components and breadboarding*
- *Introductory knowledge of AI or computer vision concepts*

- *Ability to follow schematics and connect hardware modules like servos and sensors*

HARDWARE

MATERIAL	
Raspberry Pi 5	
27W USB Type-C Power Supply for Raspberry Pi 5	
Flat Jumper Wires	
Servo Motor DS04-NFC 360° Continuous Rotation	
Raspberry Pi Camera Module 4	

Wood planks for the physical frame	
Wheels	

Table 1- List of material to implement the project SmartThrow

SOFTWARE

• **Operating System:** *Raspberry Pi OS (64-bit)*

• **Programming Environment:** *Python 3 (via Thonny or VS Code)*

• **Python Libraries:** *opencv-python, ultralytics, torch, gpiozero, lgpio, time, os*

• **Dataset Tool:** *Roboflow (for annotation and export of dataset used to train 157-2MAT.pt)*

• **Model Training Platform:** *Google Colab (used for training and testing the YOLOv8 model)*

• **Remote Access:** *VNC Viewer and SSH (for monitoring and control via local network)*

CIRCUIT

You will need to connect all components to the ground (GND) and 3.3V or 5V pins for power, depending on their requirements. The camera module connects directly to the Raspberry Pi's CSI port. The continuous servo motors are powered via the 5V and GND pins, and their signal pins are connected to GPIO 17 and GPIO 18. The buzzer is connected to GPIO 22, and the LED indicator is connected to GPIO 23 with a current-limiting resistor. All connections are made securely to ensure stable operation, and remote access is enabled via VNC or SSH for monitoring the system.

Here is the circuit for this project:

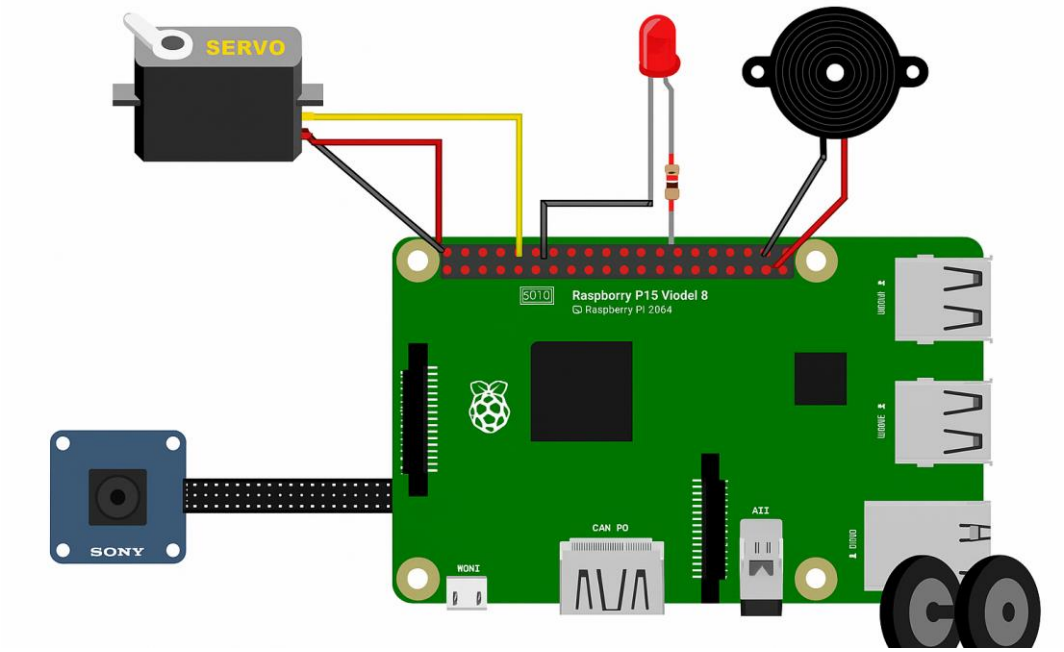


Figure 1- Circuite of the project Smart Throw

STEP-BY-STEP INSTRUCTIONS FOR SMART THROW

1. Make the circuit

Make sure that all electronic components are correctly connected:

- Connect the continuous servo motors to GPIO 17 and GPIO 18.

- Connect the buzzer to GPIO 22.

- Connect the status LED to GPIO 23 using a current-limiting resistor.

- Connect the Raspberry Pi Camera Module 4 to the CSI camera port.

- Power all components via the 5V and GND pins.

Use a common ground for all components and double-check GPIO pin assignments.

2. Prepare the software environment

Make sure you have installed the following:

- Raspberry Pi OS (64-bit)

- Python 3

- Required libraries: opencv-python, ultralytics, pytorch, gpiozero, lgpio, os, time

- Roboflow account for dataset management

- VNC or SSH enabled for remote access

It is recommended to test the camera using a basic script before running the full system.

3. Run the object detection script

Use the following Python code to run the object detection model and control the servo motors based on the recognized waste type:

```
#!/home/Desktop/SmartTrash/venv/bin/python
import cv2
from ultralytics import YOLO
from gpiozero import TonalBuzzer, Servo
from gpiozero.pins.lgpio import LGPIOFactory
from gpiozero import Device
from time import sleep
import time
import threading
import queue
import numpy as np
import os

# --- GPIO and Global Setup ---
try:
```

```
    Device.pin_factory = LGPIOFactory()
except Exception as e:
    print(f"Error setting LGPIOFactory: {e}. GPIO operations might fail.")
    print("Ensure pigpiod or lgpiod service is running and gpiozero is configured correctly.")

BUZZER_PIN_NUM = 26
SERVO1_PIN_NUM = 18
SERVO2_PIN_NUM = 19

buzzer = None
servo = None

# Camera settings
CAMERA_INDEX = 0
CAMERA_WIDTH = 640
CAMERA_HEIGHT = 480
CAMERA_FPS = 30

BEEP_FREQUENCY = 800
OBJECT_DETECTED_BEEP_DURATION = 0.2
BEEP_COOLDOWN = 2.0
TARGET_CLASSES = ["limenka", "shishe"]
LATCH_DURATION = 2.0

# --- OpenCV Camera Setup ---
cap = None

def initialize_camera():
    global cap
    cap = cv2.VideoCapture(CAMERA_INDEX)
    if not cap.isOpened():
        print(f"Error: Could not open camera at index {CAMERA_INDEX}")
        return False

    # Set camera properties for low latency
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)
    cap.set(cv2.CAP_PROP_FPS, CAMERA_FPS)
    cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)  # Minimize camera buffer

    # Additional low-latency settings (may not work on all cameras)
    try:
        cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'))  # MJPEG for lower latency
    except:
        pass

    # Verify settings
    actual_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    actual_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    actual_fps = cap.get(cv2.CAP_PROP_FPS)

    print(f"Camera initialized:")
    print(f"  Index: {CAMERA_INDEX}")
    print(f"  Resolution: {actual_width}x{actual_height}")
    print(f"  FPS: {actual_fps}")

    return True

# --- Model Configuration ---
SPECIFIED_NCNN_MODEL_PATH = "157epoch_ncnn_model"
SPECIFIED_BASE_MODEL_PATH = None
ncnn_model = None

print("--- Model Loading ---")
```

```
if SPECIFIED_NCNN_MODEL_PATH:
    print(f"Attempting to load specified NCNN model from: {SPECIFIED_NCNN_MODEL_PATH}")
    if os.path.isdir(SPECIFIED_NCNN_MODEL_PATH):
        try:
            ncnn_model = YOLO(SPECIFIED_NCNN_MODEL_PATH)
            print(f"Successfully loaded NCNN model from specified path: {SPECIFIED_NCNN_MODEL_PATH}")
        except Exception as e:
            print(f"Failed to load NCNN model from {SPECIFIED_NCNN_MODEL_PATH}: {e}")
            print("Will fallback to base model logic if SPECIFIED_BASE_MODEL_PATH is set.")
            ncnn_model = None
    else:
        print(f"Specified NCNN model path '{SPECIFIED_NCNN_MODEL_PATH}' is not a valid directory.")
        print("Will fallback to base model logic if SPECIFIED_BASE_MODEL_PATH is set.")

if ncnn_model is None:
    if SPECIFIED_BASE_MODEL_PATH:
        print(f"NCNN model not loaded directly. Using base model path: {SPECIFIED_BASE_MODEL_PATH}")
        if not os.path.isfile(SPECIFIED_BASE_MODEL_PATH):
            print(f"Error: Base model file '{SPECIFIED_BASE_MODEL_PATH}' not found.")
            cv2.destroyAllWindows()
                exit("Fatal: No valid model path provided or base model file not found.")
            else:
                base_model_name_no_ext = os.path.splitext(os.path.basename(SPECIFIED_BASE_MODEL_PATH))[0]
                derived_ncnn_model_dir = f"{base_model_name_no_ext}_ncnn_model"
                print(f"Derived NCNN model directory name: {derived_ncnn_model_dir}")
                try:
                    print(f"Checking for existing NCNN model at: {derived_ncnn_model_dir}")
                    if os.path.isdir(derived_ncnn_model_dir):
                        ncnn_model = YOLO(derived_ncnn_model_dir)
                        print(f"Successfully loaded existing NCNN model from {derived_ncnn_model_dir}")
                    else:
                        raise FileNotFoundError
                except Exception as e_load_derived:
                    print(f"NCNN model not found at '{derived_ncnn_model_dir}' or failed to load ({e_load_derived}).")
                    print(f"Exporting base model '{SPECIFIED_BASE_MODEL_PATH}' to NCNN format...")
                    try:
                        base_model_for_export = YOLO(SPECIFIED_BASE_MODEL_PATH)
                        base_model_for_export.export(format="ncnn", half=False)
                        print(f"Export complete. Loading exported NCNN model from {derived_ncnn_model_dir}...")
                        ncnn_model = YOLO(derived_ncnn_model_dir)
                        print(f"Successfully loaded exported NCNN model from {derived_ncnn_model_dir}")
                    except Exception as e_export_load:
                        print(f"FATAL error during base model loading, export, or NCNN loading: {e_export_load}")
                        cv2.destroyAllWindows()
                        exit()
```

```
            else:
                print("FATAL: Neither SPECIFIED_NCNN_MODEL_PATH nor SPECIFIED_BASE_MODEL_PATH were provided or valid.")
                cv2.destroyAllWindows()
                exit()

        if ncnn_model is None:
            print("FATAL: Could not load any model after all attempts. Exiting.")
            cv2.destroyAllWindows()
            exit()
        print("--- Model loading successful ---")

        # --- Threading Setup ---
        frame_queue = queue.Queue(maxsize=1)  # Reduced buffer size
        results_queue = queue.Queue(maxsize=1)  # Reduced buffer size
        stop_event = threading.Event()

        # --- Thread Functions ---
        def camera_thread_func():
            global stop_event, cap
            print("Camera thread started.")
            while not stop_event.is_set():
                try:
                    ret, frame_bgr = cap.read()
                    if not ret:
                        print("Camera thread: Failed to read frame. Retrying...")
                        sleep(0.01)  # Reduced sleep time
                        continue

                    # Convert BGR to RGB for YOLO processing
                    frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)

                    # Non-blocking put - drop frames if queue is full to reduce latency
                    try:
                        frame_queue.put(frame_rgb, block=False)
                    except queue.Full:
                        # Drop the frame if queue is full to prevent backlog
                        pass

                except Exception as e:
                    print(f"Camera thread error: {e}")
                    stop_event.set()
                    break
            print("Camera thread finished.")

        def yolo_thread_func():
            global stop_event, ncnn_model
            print("YOLO thread started.")
            while not stop_event.is_set():
                try:
                    frame_rgb = frame_queue.get(timeout=0.1)  # Reduced timeout
                    results = ncnn_model(frame_rgb, verbose=False, stream=False)

                    # Non-blocking put - drop results if queue is full
                    try:
                        results_queue.put((frame_rgb, results), block=False)
                    except queue.Full:
                        # Drop old results to prevent backlog
                        try:
                            results_queue.get(block=False)  # Remove old result
                            results_queue.put((frame_rgb, results), block=False)
```

```
                        except queue.Empty:
                            pass

                    frame_queue.task_done()
                except queue.Empty:
                    continue
                except Exception as e:
                    print(f"YOLO thread error: {e}")
                    stop_event.set()
                    break
            print("YOLO thread finished.")

        # --- Servo Control Functions ---
        def servo_stop(s):
            if s:
                s.value = None

        def servo_turn_continuous(s, speed_value):
            """
           Controls continuous servo.
           speed_value: -1.0 (full reverse) to 1.0 (full forward). 0 should be stop.
           """
            if s:
                if -1.0 <= speed_value <= 1.0:
                    s.value = speed_value
                else:
                    print(f"Invalid servo speed_value: {speed_value}. Must be between -1.0 and 1.0.")

        # --- Main Execution ---
        if __name__ == "__main__":
            main_thread_terminated_normally = False
            try:
                # Initialize camera first
                if not initialize_camera():
                    print("Failed to initialize camera. Exiting.")
                    exit()

                buzzer = TonalBuzzer(BUZZER_PIN_NUM)
                servo1 = Servo(SERVO1_PIN_NUM)
                servo2 = Servo(SERVO2_PIN_NUM)
                servo_stop(servo1)
                servo_stop(servo2)
                print("Servo initialized and stopped.")

                print("DEBUG: Testing TonalBuzzer immediately after init...")
                try:
                    buzzer.play(BEEP_FREQUENCY)
                    sleep(OBJECT_DETECTED_BEEP_DURATION)
                    buzzer.stop()
                    print(f"DEBUG: TonalBuzzer test beep at {BEEP_FREQUENCY}Hz finished. Did you hear it?")
                    sleep(0.5)
                except Exception as e:
                    print(f"DEBUG: Error during initial TonalBuzzer test: {e}")

                print("Camera ready.")
                sleep(1)

                cam_thread = threading.Thread(target=camera_thread_func, daemon=True)
                proc_thread = threading.Thread(target=yolo_thread_func, daemon=True)

                print("Starting worker threads...")
```

```
246.                 cam_thread.start()
247.                 proc_thread.start()
248.
249.                 prev_time_fps = time.time()
250.                 fps = 0.0
251.                 last_beep_trigger_time = 0
252.                 object_currently_triggering_beep = False
253.                 object_detected_start_time_for_latch = None
254.                 latch_active = False
255.                 active_detected_class_name = ""
256.
257.                 print("Starting detection loop. Press 'q' in OpenCV window to
    quit.")
258.
259.                 while not stop_event.is_set():
260.                     try:
261.                         original_frame_rgb, yolo_results_list =
    results_queue.get(timeout=0.1)  # Reduced timeout
262.                     except queue.Empty:
263.                         if not cam_thread.is_alive() and cam_thread.ident is
    not None:
264.                             print("Error: Camera thread seems to have stopped
    unexpectedly.")
265.                             stop_event.set()
266.                             break
267.                         if not proc_thread.is_alive() and proc_thread.ident is
    not None:
268.                             print("Error: YOLO processing thread seems to have
    stopped unexpectedly.")
269.                             stop_event.set()
270.                             break
271.
272.                         # Don't show "waiting" frame as frequently to reduce
    overhead
273.                         key = cv2.waitKey(1) & 0xFF
274.                         if key == ord("q"):
275.                             print("User pressed 'q'.")
276.                             stop_event.set()
277.                         continue
278.
279.                     current_time = time.time()
280.                     delta_time_fps = current_time - prev_time_fps
281.                     if delta_time_fps > 0:
282.                         fps = 1.0 / delta_time_fps
283.                     prev_time_fps = current_time
284.
285.                     found_target_object_this_frame = False
286.                     confidence_of_detected_object = 0.0
287.                     yolo_results = yolo_results_list[0]
288.
289.                     if yolo_results.boxes:
290.                         for box in yolo_results.boxes:
291.                             class_id = int(box.cls[0])
292.                             class_name = ncnn_model.names[class_id]
293.                             confidence = float(box.conf[0])
294.                             if class_name in TARGET_CLASSES and confidence >
    0.5:
295.                                 found_target_object_this_frame = True
296.                                 confidence_of_detected_object = confidence
297.                                 active_detected_class_name = class_name
298.                                 break
299.
300.                     if found_target_object_this_frame:
301.                         if not object_currently_triggering_beep:
302.                             if current_time - last_beep_trigger_time >
    BEEP_COOLDOWN:
```

```
                                print(f"Object '{active_detected_class_name}' detected! Confidence: {confidence_of_detected_object:.2f}. Activating beep/servo...")
                                try:
                                    buzzer.play(BEEP_FREQUENCY)
                                except Exception as e:
                                    print(f"Error during beep/servo activation: {e}")
                                finally:
                                    servo_stop(servo)
                                    if buzzer:
                                        buzzer.stop()
                                    print(f"DEBUG: Beep/servo sequence for '{active_detected_class_name}' finished.")
                                last_beep_trigger_time = current_time
                                object_currently_triggering_beep = True
                    else:
                        if object_currently_triggering_beep:
                            servo_stop(servo)
                            object_currently_triggering_beep = False

                    if found_target_object_this_frame:
                        if object_detected_start_time_for_latch is None:
                            object_detected_start_time_for_latch = current_time
                        elif not latch_active and (current_time - object_detected_start_time_for_latch >= LATCH_DURATION):
                            print(f"LATCH TRIGGERED: '{active_detected_class_name}' continuously detected for {LATCH_DURATION:.1f}s.")
                            latch_active = True
                            if active_detected_class_name == "limenka":
                                servo1.value=-1.0
                                sleep(1.2)
                                servo1.value=None
                                servo2.value=1.0
                                sleep(0.4)
                                servo2.value=None
                                sleep(2.0)
                                servo2.value=0.0
                                sleep(0.8)
                                servo1.value=1.0
                                sleep(0.8)
                                servo2.value=None
                                servo1.value=None
                            elif active_detected_class_name == "shishe":
                                servo1.value=1.0
                                sleep(1.2)
                                servo1.value=None
                                servo2.value=1.0
                                sleep(0.4)
                                servo2.value=None
                                sleep(2.0)
                                servo2.value=0.0
                                sleep(0.8)
                                servo1.value=-1.0
                                sleep(0.8)
                                servo2.value=None
                                servo1.value=None
                    else:
                        if object_detected_start_time_for_latch is not None:
                            object_detected_start_time_for_latch = None
                            if latch_active:
                                print("Latch deactivated (target object lost).")
                                latch_active = False

```

```
                    annotated_frame_rgb = yolo_results.plot()
                    annotated_frame_bgr_display = cv2.cvtColor(annotated_frame_rgb, cv2.COLOR_RGB2BGR)

                    cv2.putText(annotated_frame_bgr_display, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
                    latch_text = f"Latch: {'ACTIVE' if latch_active else 'Inactive'}"
                    latch_color = (0, 255, 0) if latch_active else (0, 0, 255)
                    cv2.putText(annotated_frame_bgr_display, latch_text, (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, latch_color, 2)
                    if found_target_object_this_frame:
                        cv2.putText(annotated_frame_bgr_display, f"Detected: {active_detected_class_name}", (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 100, 0), 2)

                    cv2.imshow("Camera", annotated_frame_bgr_display)
                    results_queue.task_done()

                    key = cv2.waitKey(1) & 0xFF
                    if key == ord("q"):
                        print("User pressed 'q'.")
                        stop_event.set()

                main_thread_terminated_normally = True

            except KeyboardInterrupt:
                print("KeyboardInterrupt received. Exiting...")
                stop_event.set()
            except Exception as e:
                print(f"An unexpected error occurred in the main loop: {e}")
                import traceback
                traceback.print_exc()
                stop_event.set()
            finally:
                print("Initiating shutdown sequence...")
                if not stop_event.is_set():
                    stop_event.set()

                if 'cam_thread' in locals() and cam_thread.is_alive():
                    print("Joining camera thread...")
                    cam_thread.join(timeout=2)
                    if cam_thread.is_alive():
                        print("Camera thread did not join in time.")
                if 'proc_thread' in locals() and proc_thread.is_alive():
                    print("Joining YOLO thread...")
                    proc_thread.join(timeout=2)
                    if proc_thread.is_alive():
                        print("YOLO thread did not join in time.")

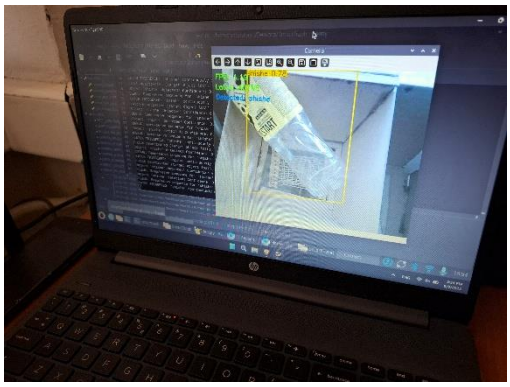
                print("Closing GPIO resources...")
                if buzzer:
                    try:
                        buzzer.stop()
                        buzzer.close()
                        print("Buzzer stopped and closed.")
                    except Exception as e:
                        print(f"Error closing buzzer: {e}")
                if servo:
                    try:
                        servo_stop(servo)
                        servo.close()
                        print("Servo stopped and closed.")
                    except Exception as e:
                        print(f"Error closing servo: {e}")

```

```
421.                 if cap and cap.isOpened():
422.                     print("Releasing camera...")
423.                     cap.release()
424.                     print("Camera released.")
425.
426.                 cv2.destroyAllWindows()
427.                 print("OpenCV windows destroyed.")
428.                 print("Program shutdown complete.")
```

4. Observe the results





EVALUATION CRITERIA:

Hardware Implementation (40%): *Properly connected circuit, no loose wires, correct sensor placement.*
Code Functionality (30%): *Script runs without errors, correctly detects light changes.*
Understanding & Explanation (20%): *Ability to explain how the system works and its real-world applications.*
Creativity & Optimization (10%): *Suggestions for improvements, such as adding additional automation features.*

ESTIMATED DURATION:

- *Start Date: March 5, 2025*
- *End Date: May 20, 2025*

LEARNING OUTCOMES:

- *Understand how AI can be applied to solve real-world environmental problems*
- *Develop skills in integrating hardware (sensors, servos) with AI-based software solutions*
- *Gain hands-on experience with tools like Raspberry Pi, Python, and Roboflow*
- *Improve teamwork, project planning, and technical documentation skills*
- *Learn to evaluate and refine prototypes through feedback and testing*

1. **Run end-to-end system tests:** *Validate that the entire system—from waste detection to sorting—is working correctly and reliably.*

CHALLENGES & TROUBLESHOOTING TIPS:

- *Camera Recognition Failures – Test the camera with basic scripts to ensure it's capturing input. Recalibrate the model or check lighting conditions if accuracy drops.*
- *Servo Motor Malfunctions – Double-check GPIO pin configuration and verify power supply; replace motors if they overheat or jitter.*
- *Model Inaccuracy – Use more annotated images in Roboflow and retrain the model if object detection fails or misclassifies.*
- *Software Errors – Check Python dependencies and library versions; use print statements or logging for easier debugging.*
- *Connectivity Problems (VNC/SSH) – Make sure the Raspberry Pi is on the correct network, and verify SSH is enabled in settings.*