code

```
import RPi.GPIO as GPIO
import time
from picamera2 import Picamera2
import cv2
import numpy as np
import os
from rembg import remove
from PIL import Image, ImageOps
from multiprocessing import Process
import time

# ---------------- PIN DEFINITIONS ----------------
DIR_PIN      = 5
STEP_PIN     = 12   # hardware PWM-capable
ENABLE_PIN   = 6
LIGHT_PIN    = 13
IR_PIN       = 26

# ---------------- MOTION CONFIG ------------------
MAX_STEPS = 19322
ABS_POSITIONS = [0, 6453, 12956, 19459]  # adjusted absolute positions
MAX_FREQ = 8000         # Hz
STEP_DISTANCE_MM = 0.01  # mm per step
STEP_REDUCTION = 150      # steps to subtract before moving

# ---------------- GPIO SETUP ---------------------
GPIO.setmode(GPIO.BCM)
GPIO.setup(DIR_PIN, GPIO.OUT)
GPIO.setup(STEP_PIN, GPIO.OUT)
GPIO.setup(ENABLE_PIN, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(LIGHT_PIN, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(IR_PIN, GPIO.IN)

# ---------------- CAMERA SETUP & WARM-UP FUNCTION ----------------
def initialize_camera():
    """
    Initializes Picamera2, sets controls, and warms up exposure and white
balance.
    Returns the Picamera2 instance.
    """
    picam2 = Picamera2()
    picam2.configure(picam2.create_still_configuration(main={"size":
(2304, 1296)}))
    picam2.start()
    time.sleep(0.5)
    picam2.set_controls({"AfMode": 0, "LensPosition": 9})

    # Camera warm-up
    GPIO.output(LIGHT_PIN, GPIO.LOW)  # Turn on LED
    picam2.set_controls({"AeEnable": True, "AwbEnable": True})
    print("⏳ Warming up camera for consistent exposure and color...")
    time.sleep(3)
    picam2.set_controls({"AeEnable": False, "AwbEnable": False})
    GPIO.output(LIGHT_PIN, GPIO.HIGH)
    print("✅ Camera warm-up done. Exposure and WB locked.")

    return picam2

# Initialize camera once
picam2 = initialize_camera()

# ---------------- STATE -------------------------
current_pos = 0

# ---------------- FUNCTIONS ----------------------

# Motor Microstepping, Homing, and Pi Camera Set-up
def pulse_motor(freq, steps):
    delay = 1 / freq / 2
    GPIO.output(ENABLE_PIN, GPIO.LOW)
    for _ in range(steps):
        GPIO.output(STEP_PIN, GPIO.HIGH)
        time.sleep(delay)
        GPIO.output(STEP_PIN, GPIO.LOW)
        time.sleep(delay)
    GPIO.output(ENABLE_PIN, GPIO.HIGH)

def move_steps(steps, direction):
    global current_pos
    GPIO.output(DIR_PIN, direction)
    pulse_motor(MAX_FREQ, steps)
    current_pos += steps if direction == GPIO.HIGH else -steps
    print(f"✅ Moved {steps} steps ({steps*STEP_DISTANCE_MM:.2f} mm).
Current position = {current_pos}")

def move_motor_sensor_based(direction, freq=MAX_FREQ,
stall_check_interval=1.0):
    """
    Move motor until IR sensor is triggered.
    If motor stops for stall_check_interval seconds, check IR sensor and
retry if necessary.
    """
    global current_pos
    GPIO.output(DIR_PIN, direction)
    delay = 1 / freq / 2
    step_count = 0
    GPIO.output(ENABLE_PIN, GPIO.LOW)
    last_step_time = time.time()

    while True:
        if GPIO.input(IR_PIN) == GPIO.HIGH:
            break  # reached home

        # Pulse motor
        GPIO.output(STEP_PIN, GPIO.HIGH)
        time.sleep(delay)
        GPIO.output(STEP_PIN, GPIO.LOW)
        time.sleep(delay)
        step_count += 1
        last_step_time = time.time()

        # If no progress for a while, check IR again
        if time.time() - last_step_time > stall_check_interval:
            if GPIO.input(IR_PIN) == GPIO.HIGH:
                break
            else:
                print("⚠️ Motor stalled. Waiting and retrying...")
                last_step_time = time.time()

    GPIO.output(ENABLE_PIN, GPIO.HIGH)
    current_pos += step_count if direction == GPIO.HIGH else -step_count
    print(f"✅ Homed {step_count} steps. Current position =
{current_pos}")

def home_motor(retries=1):
    """
    Homing function with retries if IR sensor not detected.
    """
    global current_pos
    for attempt in range(retries):
        print(f"🔍 Homing attempt {attempt + 1}...")
        GPIO.output(LIGHT_PIN, GPIO.HIGH)
        if GPIO.input(IR_PIN) == GPIO.HIGH and current_pos == 0:
            print("✔ Already at home")
            current_pos = 0
            return
        move_motor_sensor_based(GPIO.LOW)
        time.sleep(1)  # wait a bit and check sensor
        if GPIO.input(IR_PIN) == GPIO.HIGH:
            print("✔ Home reached")
            current_pos = 0
            return
        else:
            print("⚠️ IR sensor not triggered. Retrying...")

    print("❌ Homing failed after multiple attempts!")

def ensure_home_before_scan():
    if GPIO.input(IR_PIN) == GPIO.HIGH:
        print("🏠 Home confirmed before scanning.")
        return
    print("⚠️ Not at home before scanning → Re-homing now...")
    home_motor()

def capture_image(frame_num):
    filename = f"frame_{frame_num:02d}.jpg"
    GPIO.output(LIGHT_PIN, GPIO.LOW)
    time.sleep(0.5)
    picam2.capture_file(filename)
    print(f"📸 Saved: {filename} (overwrites previous)")
```

```python
    GPIO.output(LIGHT_PIN, GPIO.HIGH)
    time.sleep(0.5)

# ---------------- MAIN FUNCTION -------------------
def scan_and_stitch():
    global current_pos
    home_motor()
    ensure_home_before_scan()
    print("➡️ Starting scan sequence...")

    # Capture frames
    for current_frame, target in enumerate(ABS_POSITIONS):
        direction = GPIO.HIGH if target > current_pos else GPIO.LOW
        steps_to_move = max(abs(target - current_pos) - STEP_REDUCTION, 0)
        move_steps(steps_to_move, direction)
        capture_image(current_frame)

    print("✔ Scan complete. Stitching frames...")

    # --------- STITCHING CONFIG ---------
    frames = [f"frame_{i:02d}.jpg" for i in range(len(ABS_POSITIONS))]
    images = [cv2.imread(f) for f in frames]
    for i, img in enumerate(images):
        if img is None:
            raise FileNotFoundError(f"{frames[i]} not found")

    crop_top_px = [0, 169, 133, 120]  # top crop per frame
    left_shifts = [0, -9, -13, -29]   # horizontal shift per frame

    # Apply vertical crop
    for i in range(1, 4):
        h = images[i].shape[0]
        crop_amt = min(crop_top_px[i], h - 1)
        images[i] = images[i][crop_amt:, :].copy()

    # Compute stitched canvas size
    width = max(img.shape[1] for img in images)
    total_height = sum(img.shape[0] for img in images)
    stitched = np.zeros((total_height, width, 3), dtype=np.uint8)
    # Stitch with left shifts
    current_y = 0
    for img, shift in zip(images, left_shifts):
        h, w = img.shape[:2]
        if shift < 0:
            src_x_start = -shift
            src_x_end = w
            x_start = 0
        else:
            src_x_start = 0
            src_x_end = w
            x_start = shift
        width_to_paste = src_x_end - src_x_start
        stitched[current_y:current_y+h, x_start:x_start+width_to_paste] =
img[:, src_x_start:src_x_end]
        current_y += h

    cv2.imwrite("full_leaf_stitched_v3_separate.jpg", stitched)
    print("✅ Saved full_leaf_stitched_v3_separate.jpg")

    # Return to home after scanning
    print("➡️ Returning motor to home...")

# Processing Image and Removing background
# Suppress ONNX Runtime warnings
os.environ["ORT_LOG_SEVERITY_LEVEL"] = "3"  # errors only

def process_leaf_image(input_path: str, output_path: str, target_width:
int = 480, target_height: int = 800):
    """
    Removes background from a leaf image, crops to leaf, resizes and pads
for a given LCD size.
    """
    # 1. Load image
    img_pil = Image.open(input_path)

    # 2. Remove background using rembg (CPU)
    img_no_bg = remove(img_pil)  # returns RGBA image

    # 3. Convert transparent background to white
    img_no_bg = img_no_bg.convert("RGBA")
    background = Image.new("RGB", img_no_bg.size, (255, 255, 255))
    background.paste(img_no_bg, mask=img_no_bg.split()[3])  # use alpha
channel as mask

    # 4. Convert to NumPy array for cropping
    img_cv = cv2.cvtColor(np.array(background), cv2.COLOR_RGB2BGR)

    # Convert to grayscale for leaf detection
    gray = cv2.cvtColor(img_cv, cv2.COLOR_BGR2GRAY)
    _, leaf_mask = cv2.threshold(gray, 250, 255, cv2.THRESH_BINARY_INV)

    # Morphological cleanup
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    leaf_mask = cv2.morphologyEx(leaf_mask, cv2.MORPH_CLOSE, kernel)
    leaf_mask = cv2.morphologyEx(leaf_mask, cv2.MORPH_OPEN, kernel)

    # Find largest contour (the leaf)
    contours, _ = cv2.findContours(leaf_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        raise ValueError("No leaf detected in the image.")
    leaf_contour = max(contours, key=cv2.contourArea)

    x, y, w_crop, h_crop = cv2.boundingRect(leaf_contour)

    # Crop image
    cropped_leaf = img_cv[y:y+h_crop, x:x+w_crop]

    # --- SAVE cropped leaf with white background BEFORE resize ---
    white_bg_output = "output_image_original.png"
    Image.fromarray(cv2.cvtColor(cropped_leaf,
cv2.COLOR_BGR2RGB)).save(white_bg_output)
    print("✔ Saved leaf with white background (no resize):",
white_bg_output)
    # --------------------------------------------------------------

    # 5. Resize + pad for target LCD
    img_final = Image.fromarray(cv2.cvtColor(cropped_leaf,
cv2.COLOR_BGR2RGB))
```

```
    img_ratio = img_final.width / img_final.height
    target_ratio = target_width / target_height

    if img_ratio > target_ratio:
        new_width = target_width
        new_height = int(target_width / img_ratio)
    else:
        new_height = target_height
        new_width = int(target_height * img_ratio)

    img_resized = img_final.resize((new_width, new_height),
Image.Resampling.LANCZOS)
    img_final_padded = ImageOps.pad(img_resized, (target_width,
target_height), color="white")

    # 6. Save final image
    img_final_padded.save(output_path)
    print("✔ Leaf processed and saved as", output_path)

# ---------------- RUN -------------------
start_total = time.time()  # ← measure whole program

try:
    # Step 1: Home before scan
    home_motor()

    # Step 2: Scan & stitch
    scan_and_stitch()

    # Step 3 + 4: Run in parallel
    print("⚙️ Starting homing + preprocessing in parallel...")

    # Measure inside each process
    def timed_home_motor():
        home_motor()

    def timed_process_leaf():
        process_leaf_image(
            "full_leaf_stitched_v3_separate.jpg",
```

```python
            "output_image_reduced.png"
        )

    p1 = Process(target=timed_home_motor)
    p2 = Process(target=timed_process_leaf)

    p1.start()
    p2.start()

    p1.join()
    p2.join()

except KeyboardInterrupt:
    print("\n🛑 User interrupted.")

finally:
    GPIO.output(LIGHT_PIN, GPIO.HIGH)
    GPIO.output(ENABLE_PIN, GPIO.HIGH)
    picam2.stop()
    GPIO.cleanup()
    print("✅ GPIO cleaned. Program ended.")
```

implementation

```python
# Ito yung sa pag import
from scanner import home_motor, scan_and_stitch, process_leaf_image



def run_scan_pipeline(self):
    try:
        # Initialize camera (optional if done at start)
        self.progress_stage = "Initializing camera..."
        # If you have a function like initialize_camera(), call it here
        initialize_camera()

        # Homing motor
        self.progress_stage = "Homing motor..."
        home_motor()  # your actual homing function

        # Scan & capture frames
        self.progress_stage = "Capturing frames..."
        scan_and_stitch()  # will capture and stitch frames
        # If you want fine-grained updates, modify capture_image to call a
callback

        # Process the stitched image
        self.progress_stage = "Processing leaf image..."
        process_leaf_image("full_leaf_stitched_v3_separate.jpg",
                           "output_image_reduced.png")

        # Done
        self.progress_stage = "Scan complete!"
    except Exception as e:
        self.progress_stage = f"Error: {e}"
    finally:
        self.start_btn.disabled = False
```

puro functions nalang na mas cleaned

```python
import RPi.GPIO as GPIO
import time
from picamera2 import Picamera2
import cv2
import numpy as np
import os
from rembg import remove
from PIL import Image, ImageOps
from multiprocessing import Process

# ---------------- PIN DEFINITIONS ----------------
DIR_PIN     = 5
STEP_PIN    = 12
ENABLE_PIN  = 6
LIGHT_PIN   = 13
IR_PIN      = 26

# ---------------- MOTION CONFIG ------------------
MAX_STEPS = 19322
ABS_POSITIONS = [0, 6453, 12956, 19459]
MAX_FREQ = 8000
STEP_DISTANCE_MM = 0.01
STEP_REDUCTION = 150

# ---------------- GPIO SETUP ---------------------
GPIO.setmode(GPIO.BCM)
GPIO.setup(DIR_PIN, GPIO.OUT)
GPIO.setup(STEP_PIN, GPIO.OUT)
GPIO.setup(ENABLE_PIN, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(LIGHT_PIN, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(IR_PIN, GPIO.IN)

# ---------------- STATE --------------------------
current_pos = 0
picam2 = None  # Will be initialized externally

# ---------------- CAMERA SETUP & WARM-UP ----------------
def initialize_camera():
    global picam2
    picam2 = Picamera2()
    picam2.configure(picam2.create_still_configuration(main={"size":
(2304, 1296)}))
    picam2.start()
    time.sleep(0.5)
    picam2.set_controls({"AfMode": 0, "LensPosition": 9})

    GPIO.output(LIGHT_PIN, GPIO.LOW)
    picam2.set_controls({"AeEnable": True, "AwbEnable": True})
    print("⏳ Warming up camera for consistent exposure and color...")
    time.sleep(3)
    picam2.set_controls({"AeEnable": False, "AwbEnable": False})
    GPIO.output(LIGHT_PIN, GPIO.HIGH)
    print("✅ Camera warm-up done. Exposure and WB locked.")
    return picam2

# ---------------- MOTOR FUNCTIONS -----------------
def pulse_motor(freq, steps):
    delay = 1 / freq / 2
    GPIO.output(ENABLE_PIN, GPIO.LOW)
    for _ in range(steps):
        GPIO.output(STEP_PIN, GPIO.HIGH)
        time.sleep(delay)
        GPIO.output(STEP_PIN, GPIO.LOW)
        time.sleep(delay)
    GPIO.output(ENABLE_PIN, GPIO.HIGH)

def move_steps(steps, direction):
    global current_pos
    GPIO.output(DIR_PIN, direction)
    pulse_motor(MAX_FREQ, steps)
    current_pos += steps if direction == GPIO.HIGH else -steps
    print(f"✅ Moved {steps} steps ({steps*STEP_DISTANCE_MM:.2f} mm).
Current position = {current_pos}")

def move_motor_sensor_based(direction, freq=MAX_FREQ,
stall_check_interval=1.0):
    global current_pos
    GPIO.output(DIR_PIN, direction)
    delay = 1 / freq / 2
    step_count = 0
    GPIO.output(ENABLE_PIN, GPIO.LOW)
    last_step_time = time.time()

    while True:
        if GPIO.input(IR_PIN) == GPIO.HIGH:
            break
        GPIO.output(STEP_PIN, GPIO.HIGH)
        time.sleep(delay)
        GPIO.output(STEP_PIN, GPIO.LOW)
        time.sleep(delay)
        step_count += 1
        last_step_time = time.time()
        if time.time() - last_step_time > stall_check_interval:
            if GPIO.input(IR_PIN) == GPIO.HIGH:
                break
            else:
                print("⚠️ Motor stalled. Waiting and retrying...")
                last_step_time = time.time()

    GPIO.output(ENABLE_PIN, GPIO.HIGH)
    current_pos += step_count if direction == GPIO.HIGH else -step_count
    print(f"✅ Homed {step_count} steps. Current position =
{current_pos}")

def home_motor(retries=1):
    global current_pos
    for attempt in range(retries):
        print(f"🔍 Homing attempt {attempt + 1}...")
        GPIO.output(LIGHT_PIN, GPIO.HIGH)
        if GPIO.input(IR_PIN) == GPIO.HIGH and current_pos == 0:
            print("✔ Already at home")
            current_pos = 0
            return
        move_motor_sensor_based(GPIO.LOW)
        time.sleep(1)
        if GPIO.input(IR_PIN) == GPIO.HIGH:
            print("✔ Home reached")
            current_pos = 0
            return
        else:
            print("⚠️ IR sensor not triggered. Retrying...")
    print("❌ Homing failed after multiple attempts!")

def ensure_home_before_scan():
    if GPIO.input(IR_PIN) == GPIO.HIGH:
        print("🏠 Home confirmed before scanning.")
        return
    print("⚠️ Not at home before scanning → Re-homing now...")
    home_motor()

def capture_image(frame_num, picam2_instance=None):
    global picam2
    if picam2_instance is None:
        picam2_instance = picam2
    filename = f"frame_{frame_num:02d}.jpg"
    GPIO.output(LIGHT_PIN, GPIO.LOW)
    time.sleep(0.5)
    picam2_instance.capture_file(filename)
    print(f"📸 Saved: {filename} (overwrites previous)")
    GPIO.output(LIGHT_PIN, GPIO.HIGH)
    time.sleep(0.5)

# ---------------- SCAN & STITCH -----------------
def scan_and_stitch(progress_callback=None):
    global current_pos
    home_motor()
    ensure_home_before_scan()
    if progress_callback:
        progress_callback(0, "Starting scan sequence...")

    total_frames = len(ABS_POSITIONS)
    for current_frame, target in enumerate(ABS_POSITIONS):
        direction = GPIO.HIGH if target > current_pos else GPIO.LOW
        steps_to_move = max(abs(target - current_pos) - STEP_REDUCTION, 0)
        move_steps(steps_to_move, direction)
        capture_image(current_frame)
        if progress_callback:
            pct = int((current_frame+1)/total_frames*50)
            progress_callback(pct, f"Captured frame 
{current_frame+1}/{total_frames}")
```

```
    if progress_callback:
        progress_callback(50, "Frames captured. Stitching images...")

    frames = [f"frame_{i:02d}.jpg" for i in range(total_frames)]
    images = [cv2.imread(f) for f in frames]
    crop_top_px = [0, 169, 133, 120]
    left_shifts = [0, -9, -13, -29]
    for i in range(1, total_frames):
        h = images[i].shape[0]
        crop_amt = min(crop_top_px[i], h - 1)
        images[i] = images[i][crop_amt:, :].copy()

    width = max(img.shape[1] for img in images)
    total_height = sum(img.shape[0] for img in images)
    stitched = np.zeros((total_height, width, 3), dtype=np.uint8)
    current_y = 0
    for idx, (img, shift) in enumerate(zip(images, left_shifts)):
        h, w = img.shape[:2]
        src_x_start = -shift if shift < 0 else 0
        src_x_end = w
        x_start = 0 if shift < 0 else shift
        stitched[current_y:current_y+h,
x_start:x_start+(src_x_end-src_x_start)] = img[:, src_x_start:src_x_end]
        current_y += h
        if progress_callback:
            pct = 50 + int((idx+1)/total_frames*20)
            progress_callback(pct, f"Stitching frame
{idx+1}/{total_frames}")

    cv2.imwrite("full_leaf_stitched_v3_separate.jpg", stitched)
    if progress_callback:
        progress_callback(70, "Stitching completed. Returning motor
home...")

# ---------------- LEAF PROCESSING -----------------
os.environ["ORT_LOG_SEVERITY_LEVEL"] = "3"

def process_leaf_image(input_path, output_path, target_width=480,
target_height=800, progress_callback=None):
    if progress_callback:
        progress_callback(70, "Loading image...")
    img_pil = Image.open(input_path)

    if progress_callback:
        progress_callback(75, "Removing background...")
    img_no_bg = remove(img_pil)
    img_no_bg = img_no_bg.convert("RGBA")
    background = Image.new("RGB", img_no_bg.size, (255, 255, 255))
    background.paste(img_no_bg, mask=img_no_bg.split()[3])
    img_cv = cv2.cvtColor(np.array(background), cv2.COLOR_RGB2BGR)

    gray = cv2.cvtColor(img_cv, cv2.COLOR_BGR2GRAY)
    _, leaf_mask = cv2.threshold(gray, 250, 255, cv2.THRESH_BINARY_INV)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    leaf_mask = cv2.morphologyEx(leaf_mask, cv2.MORPH_CLOSE, kernel)
    leaf_mask = cv2.morphologyEx(leaf_mask, cv2.MORPH_OPEN, kernel)
    contours, _ = cv2.findContours(leaf_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        raise ValueError("No leaf detected in the image.")
    leaf_contour = max(contours, key=cv2.contourArea)
    x, y, w_crop, h_crop = cv2.boundingRect(leaf_contour)
    cropped_leaf = img_cv[y:y+h_crop, x:x+w_crop]

    white_bg_output = "output_image_original.png"
    Image.fromarray(cv2.cvtColor(cropped_leaf,
cv2.COLOR_BGR2RGB)).save(white_bg_output)
    if progress_callback:
        progress_callback(85, "Cropped leaf saved.")

    img_final = Image.fromarray(cv2.cvtColor(cropped_leaf,
cv2.COLOR_BGR2RGB))
    img_ratio = img_final.width / img_final.height
    target_ratio = target_width / target_height
    if img_ratio > target_ratio:
        new_width = target_width
        new_height = int(target_width / img_ratio)
    else:
        new_height = target_height
        new_width = int(target_height * img_ratio)
    img_resized = img_final.resize((new_width, new_height),
Image.Resampling.LANCZOS)
    img_final_padded = ImageOps.pad(img_resized, (target_width,
target_height), color="white")
    img_final_padded.save(output_path)
    if progress_callback:
        progress_callback(100, f"Leaf processed and saved as
{output_path}")

# ---------------- MULTIPROCESS WITH PARALLEL HOMING -----------------
def
run_leaf_processing_parallel(input_path="full_leaf_stitched_v3_separate.jp
g",
                                 output_path="output_image_reduced.png",
                                 progress_callback=None):
    """
    Runs leaf preprocessing in parallel with homing.
    Homing runs silently, leaf processing reports progress.
    """
    p_leaf = Process(target=process_leaf_image,
                     args=(input_path, output_path),
                     kwargs={"progress_callback": progress_callback})
    p_home = Process(target=home_motor)

    p_leaf.start()
    p_home.start()

    p_leaf.join()
    p_home.join()

def cleanup_gpio():
    GPIO.output(LIGHT_PIN, GPIO.HIGH)
    GPIO.output(ENABLE_PIN, GPIO.HIGH)
    if picam2:
        picam2.stop()
    GPIO.cleanup()
    print("✅ GPIO cleaned. Program ended.")
```