Phase 4: Testing, Calibration & Refinement

Systematic testing is crucial for identifying and fixing issues early.

1. Individual Component Tests (Re-verify)

- **MicroPython Firmware:** Ensure MicroPython firmware is successfully flashed on both ESP32s. You should be able to connect via Thonny IDE and access the REPL.
- **ESP32 (Animatronic Brain) & PCA9685 Connection:**
  - **Purpose:** Confirm I2C communication and basic servo control is robust.
  - **Wiring:** Connect only the Animatronic Brain ESP32 to the PCA9685 (SDA/SCL, VCC, GND). Connect your dedicated 5V power supply to the PCA9685's V+ and GND, and ensure common ground with the ESP32. Connect *one* servo to Channel 0.
  - **Procedure:** Upload the `main.py` code for the Animatronic Brain. Open the Thonny IDE's Shell. If the PCA9685 is detected, you should see "PCA9685 initialized successfully." If not, check wiring and the I2C address. You can also manually test in the REPL:

```
import machine
from your_main_script import PCA9685 # Assuming you have the class in your main.py
i2c = machine.I2C(1, sda=machine.Pin(21), scl=machine.Pin(22), freq=400000)
pwm = PCA9685(i2c)
pwm.set_pwm(0, 0, 300) # Set servo to a position
```

- **Calibrate All Servos:**
  - **Purpose:** Precisely identify the PWM values for each of your 8 servos for their specific range of motion in the actual endoskeleton.
  - **Procedure:** Using the Animatronic Brain ESP32 with the `main.py` script loaded, modify the `set_servo_pos` function or add a temporary loop in `main.py` to slowly sweep a single servo's range (e.g., from 100 to 600). Observe the physical movement and note the precise PWM values where it reaches its desired limits without straining or buzzing. Update the `SERVO_RANGES` dictionary in your `main.py` for the Animatronic Brain. Repeat for all 8 servos.
- **Joystick (Controller ESP32) Test:**
  - **Purpose:** Verify accurate analog readings and button input.
  - **Wiring:** Connect only the Controller ESP32 and the Joystick. Power the ESP32 via USB.
  - **Procedure:** Upload the `main.py` code for the Controller. Open the Thonny IDE's Shell. Move the joystick around and press the button. You should see print statements showing the raw X, Y values and button state. Confirm they respond correctly to your inputs.

2. Wireless Communication Test (ESP-NOW)

- **1. Get MAC Address of Animatronic Brain ESP32:**
    - **Purpose:** The Controller ESP32 needs the MAC address of the Brain ESP32 to send targeted ESP-NOW messages.
    - **Procedure:** Connect the Animatronic Brain ESP32 to Thonny. Open the Shell (REPL). Type:

```
import network
sta = network.WLAN(network.STA_IF)
sta.active(True)
print(sta.config('mac').hex())
```

    - Note down the MAC address (e.g., `a0b1c2d3e4f5`). You will use this to update the `PEER_MAC` variable in your Controller's `main.py`.
- **2. Basic ESP-NOW Send/Receive Test:**
    - **Purpose:** Verify that the two ESP32s can communicate wirelessly.
    - **Wiring:** Both ESP32s powered.
    - **Procedure:**
        - Ensure the `PEER_MAC` in the Controller's `main.py` is correctly updated.
        - Upload the `main.py` to both ESP32s (Animatronic Brain and Controller).
        - Open the Shell in Thonny for *both* ESP32s.
        - Observe the Controller's output (should show "Data sent successfully" or errors) and the Animatronic Brain's output (should show "Received from..." and unpacked data).
    - **Troubleshooting:**
        - "Error sending data" or no data received: Double-check `PEER_MAC`. Ensure both ESP32s are active, within range, and their `espnow` instances are active.

3. Integrated System Test & Refinement

- **Purpose:** Test the entire animatronic system as a single unit and make final adjustments for smooth, coordinated movements.

- **Actionable Steps:**

    - **Full System Power-Up:** Ensure both the animatronic and controller are powered on.
    - **Control Verification:** Use the joystick to control eye movement, torso movement, and trigger blinking/waving.
    - **Code Refinement:**
        - **Movement Mapping:** Adjust the `map()` function parameters in the Controller's code. Does a small joystick movement translate to a small, precise animatronic movement? Does full joystick deflection give the desired full range?

        - **Speed & Easing:** For smoother, more natural movements, consider adding basic easing functions or gradual changes to servo positions in the Brain's code instead of instant jumps. This might involve tracking the current and target positions and moving the servo incrementally over several loop iterations.
        - **Blinking/Waving Timing:** Adjust the `time.sleep_ms()` values in the `blink_eye()` and `wave_hand()` functions to achieve the desired speed and naturalness.
    - **Mechanical Adjustments:** Make any minor physical adjustments to linkages, pivots, or servo mounts if movements are stiff, wobbly, or limited.
    - **Stress Test:** Run the animatronic for an extended period (e.g., 30-60 minutes) to check for:
        - **Overheating:** Feel servos and the PCA9685. If they are very hot, you might be over-straining servos or need more power.
        - **Power Issues:** Observe for erratic movements, resets, or dimming LEDs, which could indicate insufficient power to the servos (requiring a stronger 5V power supply) or the ESP32.
        - **Mechanical Wear:** Listen for grinding or binding noises.
- **Deliverables:** A fully functional and refined animatronic endoskeleton that responds smoothly to wireless commands.
