University of Hertfordshire UH

COURSEWORK REPORT

Student Name: Tomiwa Adeyemi

Bio-inspired Robot Development

25th April 2025

Introduction

This report describes the creation and integration of a bio-inspired hexapod robot that reacts to environmental stimuli, particularly variations in light intensity, by displaying reactive behaviour. In order to create a robot that could sense, react, and carry out simple autonomous tasks, the project, called Bio-Inspired Robot Development, looked to integrate the fields of mechanical design, mechatronics, and embedded systems programming. With two reactive wings actuated by high-torque MG996R servos and six articulated legs driven by SG90 servos, the robot is designed to mimic the protective wing motions and natural mobility of some insect species.

A Raspberry Pi Pico microcontroller running CircuitPython was the foundation of the robot's operation, acting as the main hub for actuator control and sensor data collection. Thanks to the light sensor, the robot was able to recognise changes in its surroundings, and when it detected possible dangers or darkness, servo-driven wing movements imitated a defensive opening posture. The leg components were calibrated to lift and react to inputs, proving the possibility of enhanced locomotion functionality in subsequent iterations, even though they were not programmed for complete walking sequences.

Our team, which included Saeed Toukhai, Adham Osman, Isaac Anene, and me (Tomiwa Adeyemi), established an agile development methodology divided into four sprints to oversee this interdisciplinary endeavour. Every sprint was led by a different team member serving as the Scrum Master and concentrated on a particular robot subsystem. This arrangement guaranteed distributed leadership, distinct task ownership, and iterative improvement throughout the design lifecycle.

- **Sprint 1**, led by Isaac, centred on using Fusion 360 for mechanical design. The robot's entire CAD model was created with close attention to the servo mounting and joint articulation locations.
- Adham led the introduction of the electrical components in **Sprint 2**. Among the tasks were soldering a light sensor for environmental input, integrating the ultrasonic sensors, and soldering the Raspberry Pi Pico.
- **Sprint 3**, led by Saeed, focused on structural improvement and physical integration. The 3D-printed frame was used to mount the robot's servos, sensors, and body parts. At the end of the sprint, the hardware platform was stable and entirely constructed.
- As Scrum Master, I oversaw Sprint 4, centring on system-wide testing, servo control programming, sensor logic development, and code integration. This step necessitated tight cooperation from all participants to integrate the components into a unified and functional robot. During this sprint, I oversaw the delegation of tasks, the monitoring of bugs, and the final verification of the robot's behaviour in various lighting scenarios.

To stay organised and communicate throughout the project, we used collaboration tools. Our primary task management tool was Trello, where each sprint's tasks were arranged into "To Do," "Doing," and "Done" lists. This made deadlines and task responsibilities clearer. Additionally, we stored and accessed CAD files, code, wiring diagrams, and documentation via a shared Google Drive. Particularly when in-person lab access was scarce, WhatsApp enabled daily collaboration, quick feedback, and video-based troubleshooting.

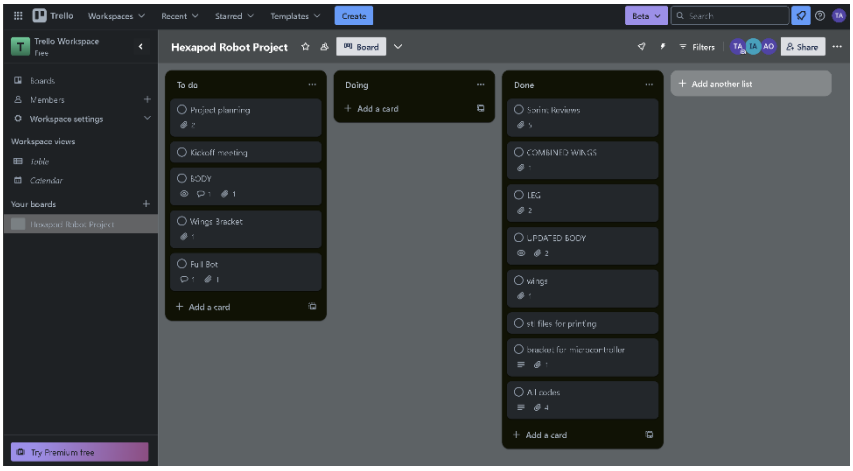

Figure 1 *Trello Workspace*

This project was a great way to practise interdisciplinary teamwork and real engineering. It required a delicate balancing act between software flexibility, electronic dependability, and mechanical precision. Each sprint aided the robot's gradual development from concept to prototype. For me, this project increased my expertise in agile team management, real-time programming, and system integration. It also tested my ability to lead, speak clearly, and adjust fast amid the most difficult and time-sensitive developmental stage.

CAD Design and Development of the Reactive Wing System

Role: Team contributor
Tools: Autodesk Fusion 360

The Bio-Inspired Robot Development project started with Sprint 1, during which we worked to turn preliminary concepts into tangible mechanical parts. The sprint's focus was the hexapod robot's dynamic qualities and complete mechanical conceptualisation. The reactive wing system's overall design, which included incorporating the servo-driven wing mechanism, structural servo brackets, and motion-clearance solutions, was my main contribution during this period. This part would eventually provide the project's primary "bio-inspired" behavioural feature by allowing the robot to expand its wings in reaction to external cues like light.

Design Motivation and Approach

I wanted to create a set of wings that could fold open to simulate an abrupt, reactionary movement, drawing inspiration from animal defence mechanisms, especially those found in bats and moths. To give the robot an expressive and natural-looking appearance, I used Fusion 360 to construct organic, membrane-style wing designs that closely resembled feathered or bat-like elements.

MG996R high-torque servos, which provide both strength and speed, were supposed to power the wings, but their size and weight created further limitations. Since I designed them to be lightweight, the

wings have a striking shape and minimal surface area. The bracketed MG996Rs positioned on the robot's rear would be used to actuate the servo arms to which each wing is clipped.

Bracket Design and Mechanical Integration

I created a unique dual-servo bracket system to support this, enabling the wings to rotate outward at precise angles. The bracket's dimensions were meticulously determined to:

- Secure the MG996Rs while avoiding physical interference with other robot components
- Allow up to 120 degrees of wing rotation without collision
- Route wiring cleanly through the side to avoid snags or exposed connections

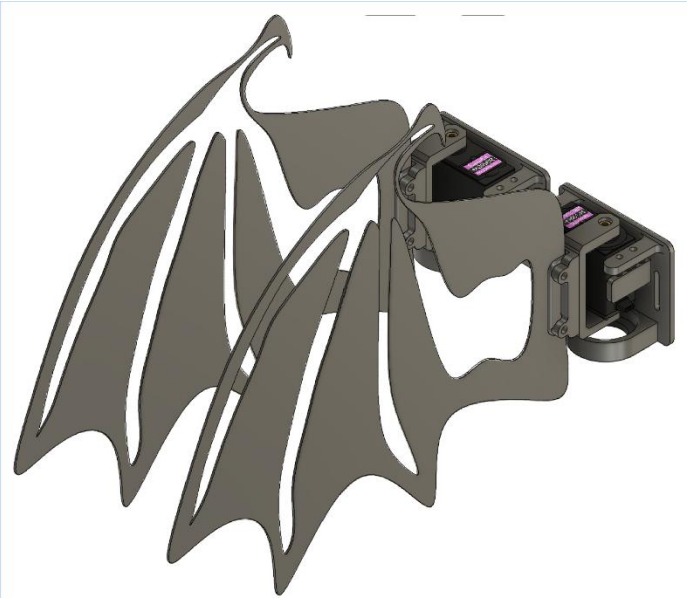

Figure 2 *Full wings integrated*

I used fillets and chamfers throughout the design to lessen stress points and enhance 3D printability. The brackets were made to be printed without the need for support material to increase manufacturing efficiency and minimise cleanup. Using the stiff joint simulation tool in Fusion 360, I verified movement and made sure the servo arms wouldn't collide with the robot's wiring or chassis when rotating.

Ball Bearing Load Distribution System

One of the biggest engineering challenges was the problem of torque stress on the MG996R servo arms. Despite their strength, these servos would harm or shorten the gearboxes' lifespan if they were required to sustain the entire weight of the wings during numerous actuation cycles. To lessen this, I used a ball bearing-based support system in the bracket design.

I made a tiny but crucial component known as the "pin post," which is a cylindrical peg that is screwed to the servo arm after being press-fitted through a ball bearing. To enable this, I made a hex nut recess on the bracket's other face. As a result, a nut could be inserted during installation, allowing the screw to secure the servo arm to the bearing directly and dump the full weight of the wings onto the bracket rather than the servo. This fix increased the wing movement's stability and dependability while lowering the servo's torsional stress.

Strategic Component Placement

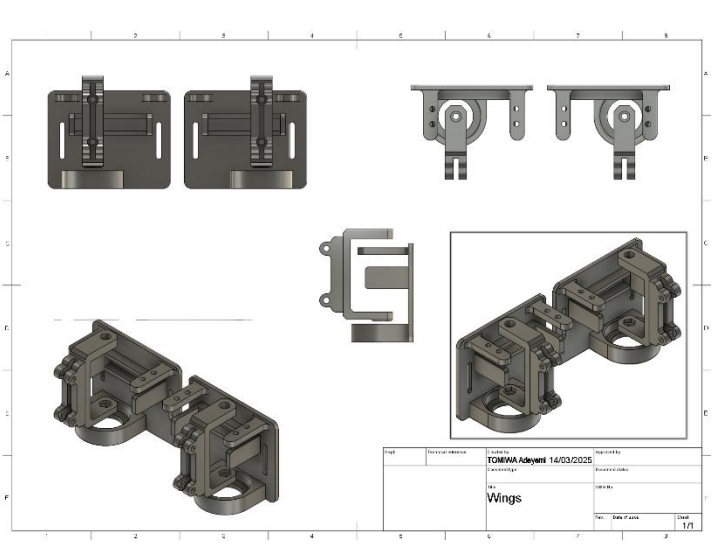

Figure 3 *Wing Bracket*

The servo brackets were also constructed with strategic servo orientation to avoid physical collisions during rotation. During the simulation, I saw that the bracket walls would interfere if the servos were orientated in a traditional configuration. I slightly shifted the mount angles to make clearance channels for the wires, which now exit upward and out through authorised routes. These little changes significantly impacted the ease of wiring and the overall integration in subsequent sprints.

Even though I was in charge of designing the wing system, teamwork was crucial. Adham assisted in modifying the design during the actual physical construction of the bracket after it was finished. He made mechanical improvements without changing the bracket's essential structure because it was modular.

Evaluation and Forward Planning

For the coding and testing completed in Sprint 4, this sprint laid a solid basis for subsequent sprints. I entered suggestions for improvements, such as weight balancing, servo over-rotation, bracket reinforcement, and possible mechanical hazards, into Trello for our backlog. We uploaded the CAD files and schematics to our shared Google Drive so that the team could examine and modify them.

By the conclusion of Sprint 1, my work had demonstrated the reactive wing system's mechanical viability and modularity. The system's successful final integration confirmed the design methodology and functioned as anticipated in simulations and 3D print tests. This procedure improved my abilities in CAD modelling, load analysis, and mechanical foresight, which were crucial in bringing our robot's most recognisable characteristics to life.

Mechatronics and Power System Development

Role: Team contributor

Focus: Soldering, Power Supply, Component Integration

The second sprint of the Bio-Inspired Robot Development project focused on creating a reliable and efficient electronic system to power, regulate, and coordinate the robot's sensors and actuators. After

completing the mechanical construction in Sprint 1, we focused on setting up and testing the servos, sensors, power distribution system, and Raspberry Pi Pico. To ensure reliable servo performance under stress, I contributed to this sprint by soldering the microcontroller, wiring and configuring the Adafruit power system, and resolving significant voltage instability issues.

Soldering Components

Soldering a single side of the Raspberry Pi Pico, the robot's brain, was one of my first assignments. Afterwards, this microcontroller would carry out all programmed logic, analyse sensor data, and regulate servo actuation. I carefully affixed male headers to the board using a precision solder and a fine-tip soldering iron. I then used a digital multimeter to perform continuity tests, verifying every joint's integrity and ensuring there were no open circuits or shorts.

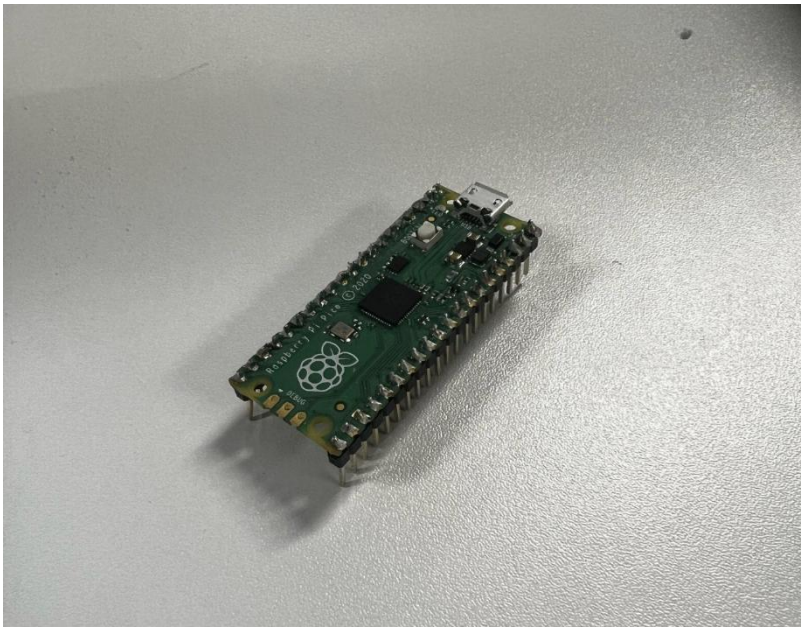

Figure 4 *Pico fully soldered*

This was a crucial step in ensuring that the microcontroller could consistently interface with other system components and that control signals would reach all linked servos. Thanks to my effective execution of this task, we had a completely functional platform to expand upon in subsequent sprints.

I helped with the wiring and configuration of the Adafruit PowerBoost module and worked on the Pico. In addition to low-power components like sensors and logic lines, this power board provided 5V regulated electricity throughout the system, including the 12 SG90 leg servos and 2 MG996R wing servos.

I assisted in routing power output lines through a shared breadboard that served as our power distribution bus and soldering the required terminals (VOUT, GND, and EN). These lines went straight to the power rail of each servo, and in order to guarantee correct referencing, GND lines were shared with the Raspberry Pi Pico.

When necessary, I used thicker gauge wires to lower resistance and avoid overheating while operating the 14 servos under current stress. When the Adafruit board was firmly installed, voltage constancy was checked for under both light-load and no-load scenarios. The results were recorded and found to be within tolerance.

Solving Servo Jitter and Voltage Drop Issues

Servos jittered erratically, failed to complete movements, and occasionally caused the Raspberry Pi Pico to reset during system activation. These symptoms were typical voltage instability or power depletion indicators, mainly when high-torque servos like the MG996R were engaged.

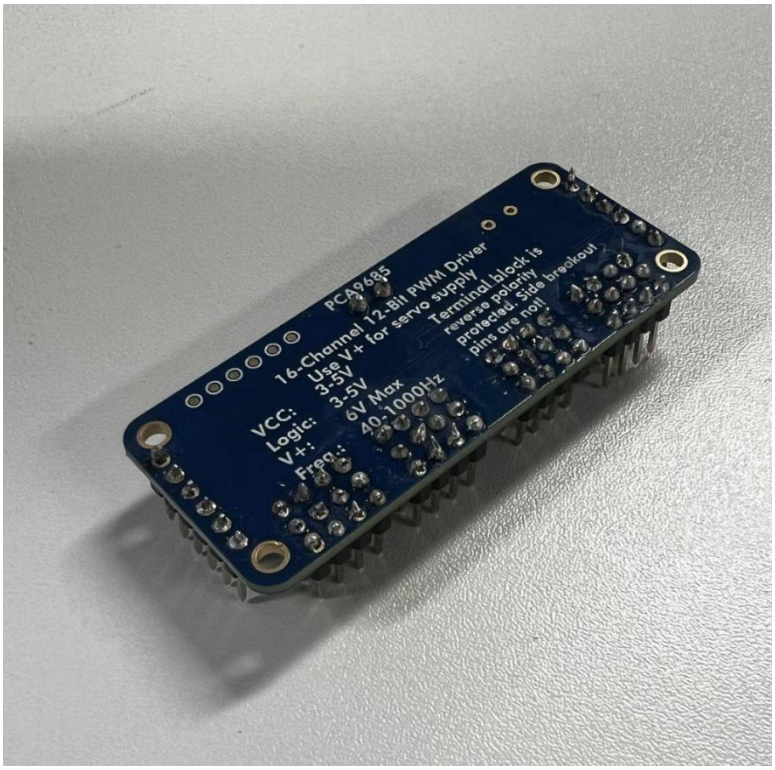

Figure 5 *Fully soldered Adafruit*

I started an infrastructure upgrade for power to fix this. The first thing I found and attached was a regulated 5V/10A external DC power supply that could support all 14 servos at full load. This was a replacement for our USB-based system, which had been unreliable and inadequate.

Using a DC barrel jack to screw the terminal adaptor, I was able to securely incorporate this new power source and connect the power supply straight to the system's breadboard. Additionally, this modular connection made it possible to disconnect and debug in subsequent sprints safely.

Understanding that brief current surges could still cause voltage dips; I wired a 1000µF electrolytic capacitor straight into the Adafruit module's output. The capacitor maintained a steady voltage over the whole 5V rail by buffering surges in current demand, especially when several servos started moving simultaneously.

Validation and Testing of Power Stability

Once the new power supply and capacitor were installed, I used a multimeter to measure the voltage across:

- The output of the capacitor
- The Pico’s VIN pin
- The servo rail under load

Even during rapid servo actuation, the voltage stayed between 4.95V and 5.05V in every case. The Raspberry Pi Pico stayed steady, the servo jitter vanished, and the component response became steady and fluid.

This verified that the new power supply and filtering capacitor had completely stabilised the system, opening the door for the behaviour logic and successful integration created in Sprint 4.

Evaluation of Contribution

My contributions during Sprint 2 directly contributed to the robot's electronic dependability. I ensured the robot could function safely under load by meticulously soldering and testing the Raspberry Pi Pico, setting up the Adafruit module to power 14 servos, and resolving the system's voltage drop problems using actual hardware improvements.
The tools and techniques used included:

- Soldering and header alignment
- Multimeter-based voltage and continuity testing
- High-current DC power integration
- Capacitor-based electrical filtering
- Safe wiring and current handling via terminal adapters

As a result of this sprint, my comprehension of electrical integration and practical mechatronics has improved. These involve not simply joining parts but setting them up to function dependably in real-world situations.

System Validation and Basic Mechatronics Testing

Role: Team Contributor
Focus: Sensor Reading Validation, Servo Actuation Testing, and System Stability Confirmation

The Bio-Inspired Robot Development project's third sprint was devoted to verifying the fundamental mechatronic subsystems. The objective of Sprint 3 was to make sure the robot could move actuators, read sensor inputs, and maintain stable electrical functioning after putting the mechanical structure together in Sprint 1 and connecting the electronics in Sprint 2.
Despite not serving as Sprint 3's Scrum Master, I was crucial in assisting with system-level validation. I helped verify the servo actuation sequences and the ultrasonic sensor's performance, which verified that the robot's embedded control system was prepared for the last integration step.

Sensor Validation (HC-SR04 Ultrasonic Testing)
One of Sprint 3's main goals was to verify that our two HC-SR04 ultrasonic sensors could precisely measure distance and provide useful data for behaviour programming.
Together with Adham, I used CircuitPython and the Adafruit hcsr04 module to execute a simple test script that:

- Triggered ultrasonic pulses from each sensor (GP2/GP3 and GP4/GP5 on the Pico)
- Measured the returned echo to determine distance
- Printed sensor outputs via the serial console every 0.5 seconds. Full Example snippet is shown in Appendix A

We created and tested the script using the Thonny IDE, which enabled real-time debugging, REPL interaction, and direct code upload. **Try/except** error handling was incorporated into the sensor code to guarantee that the program could continue functioning even if an echo was missed. Physical objects were positioned at predetermined intervals (5, 10, and 20 cm) to test the sensors, and the measured

values were compared to the expected values. The sensors demonstrated their dependability in interior settings by consistently providing correct feedback within ±2 cm.

This step verified the robot's ability to recognise items in its surroundings, which was crucial for the following behaviour modules: predator response and object avoidance.

BH1750 Light Sensor Testing

Our project also incorporated a BH1750 ambient light sensor for Task 3: Phototaxis (approaching light and stopping at 20 cm). We examined the sensor to ensure it provided valuable data even if this operation was unfinished. The full snippet code for testing is shown in Appendix B.

Isaac used CircuitPython to incorporate the sensor, and I helped him with testing and troubleshooting. The sensor, which was connected to the I2C bus (GP0 for SDA and GP1 for SCL), produced responsive, fluid lux readings. Real-time response was confirmed, for instance, when the sensor was covered with a hand or illuminated with a torch.

This validation confirmed the sensor's functionality even though the phototaxis behaviour it was intended to allow was incomplete.

Servo Movement Validation – Leg and Hip Actuation

Alongside sensor testing, I helped validate actuator control using servo testing scripts Saeed primarily authored.

The key goal was to ensure that:

- All 12 servo motors (6 hips, 6 legs) were operational
- Each servo could move smoothly between its resting, raised, and swung positions
- The PCA9685 servo controller correctly generated PWM signals at the appropriate frequency (50 Hz)

Testing Method:

- We assigned each servo an initial "stand" position
- Then sequentially commanded each leg to lift, swing, lower, and reset
- Observed physical movement and monitored current draw stability

Example movement logic:

```
for i in range(6):
    legs[i].angle = LEG_UP
    hips[i].angle = HIP_SWING
    legs[i].angle = LEG_DOWN
    hips[i].angle = HIP_NEUTRAL
    time.sleep(DELAY)
```

The servo motion did not exhibit any noticeable jitter or unusual behaviour. This verified that our soldered connections (from Sprint 2) and the PCA9685 driver were both functioning properly.

Evaluation of Contribution

By testing the ultrasonic sensors, helping to validate the light sensors, and supporting servo movement checks, I contributed significantly to Sprint 3. Additionally, I assisted in verifying the stability of the power system under practical load conditions. Despite not writing the light sensor code, I helped with testing and ensured the outcomes performed as planned.

This sprint required de-risking the project's latter phases. We couldn't have confidently moved on to the behavioural development in Sprint 4 if we hadn't verified that our robot could sense and move reliably.

Behavioural Programming and Testing (Task 4: Predator Response)

Role: Developer

Focus: Threat Detection, Servo Actuation, and Reactive Control

I oversaw executing Task 4: Predator Response for Sprint 4, which involves the robot detecting abrupt motion in its surroundings and reacting by spreading its wings. The aim was to develop a defensive behavioural response akin to insect startle systems. This necessitated full utilisation of the robot's servo system, programming interface, and sensor suite, indicating comprehensive involvement with the project's hardware and software layers.

CircuitPython on the Thonny IDE was used to create the behaviour, and real-time REPL monitoring was used for testing and debugging. Two forward-facing HC-SR04 ultrasonic sensors provided the primary input, and the Adafruit PCA9685 16-channel PWM driver was used to control the output through two MG996R high-torque servos.

The script started by setting up the Raspberry Pi Pico and PCA9685 module's I2C communication:

```
i2c = busio.I2C(scl=board.GP1, sda=board.GP0)
pca = PCA9685(i2c)
pca.frequency = 50
```

Servo motors were mapped to Channels 12 (right wing) and 13 (left wing), utilising modified pulse width parameters for precise angle control. I determined the upper and lower positions for each wing based on physical testing to guarantee a symmetrical, safe, and dramatic motion:

```
wing_right = servo.Servo(pca.channels[12], min_pulse=600, max_pulse=2400)
wing_left  = servo.Servo(pca.channels[13], min_pulse=600, max_pulse=2400)

WING_DOWN_LEFT  = 120
WING_UP_LEFT    = 180
```

```
WING_DOWN_RIGHT = 150
WING_UP_RIGHT   = 90
```

The two HC-SR04 sensors were then initialised on GPIO pins GP2/GP3 and GP4/GP5, respectively. These sensors obtained continuous distance readings:

```
sonar1 = HCSR04(trigger_pin=board.GP2, echo_pin=board.GP3)
sonar2 = HCSR04(trigger_pin=board.GP4, echo_pin=board.GP5)
```

The program loop continuously compared each new distance value with the previous one. If either sensor detected a rapid decrease in distance greater than 40 cm within ~0.2 seconds, this was treated as a potential “threat”:

```
if prev_d1 is not None and d1 is not None:
    if prev_d1 - d1 > 40:
        threat = True
```

When triggered, the servos moved the wings to their "up" positions using a **raise_wings()** function, holding them for two seconds before returning to their normal state:

```
def raise_wings():
    wing_left.angle  = WING_UP_LEFT
    wing_right.angle = WING_UP_RIGHT
    print("Rapid approach detected! WINGS UP")
```

If there was no threat, the wings were returned to their closed condition via a lower_wings() method. The REPL was printed with action logs and real-time distance feedback for testing and calibration.

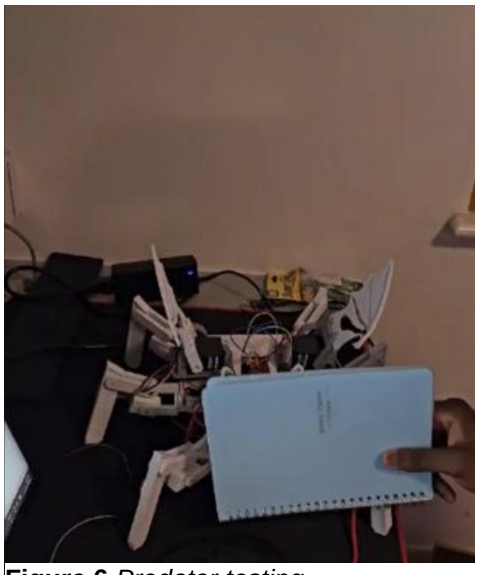

Figure 6 *Predator testing*

For both sensors, I incorporated strong **try/except** blocks to ensure that the robot would keep operating even if one reading failed. This lessened the likelihood that brief sensor noise would cause the entire behaviour to crash.

Through this challenge, I demonstrated my abilities to combine servo control and sensory input, apply conditional behaviour based on abrupt environmental changes, and optimise mechanical safety and responsiveness. The behaviour performed consistently in a range of illumination settings throughout lab testing, and it reacted reactively to fast motion, such as a hand waving or an object being pushed rapidly in the robot's direction.

By the end of Sprint 4, I had used hardware and software tools such as Thonny, CircuitPython, HC-SR04s, MG996R servos, and the PCA9685 driver to finish Task 4 to specification. This proved that I could create a functional, expressive robot feature by using testing, calibration, and logical planning in addition to my technical execution. The Full source code is included in Appendix C

Scrum Master Leadership Reflection

Role: Scrum Master
Focus: Final Integration, Team Coordination, Task Delivery

As the Scrum Master for the Bio-Inspired Robot Development project during Sprint 4, I was in charge of final system integration, managed behavioural implementation, and ensured all four project tasks were completed successfully. This sprint signalled the change from isolated development to a behaviour-driven, fully functional robot. To guarantee seamless delivery for the team, my leadership concentrated on balancing technical execution and project coordination.

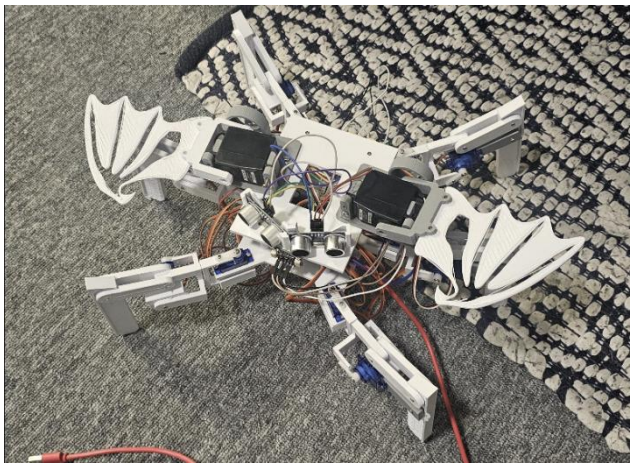

Figure 7 *Each sprint led to the creation of the robot*

We began the sprint by assigning clear ownership of each behavioural task:

- Task 1: Locomotion – Saeed
- Task 2: Object Avoidance – Adham
- Task 3: Phototaxis – Isaac
- Task 4: Predator Response – Myself (Tomiwa)

Each team member's talents and past experience with the sensors or code were considered while dividing up the tasks. Running a sprint planning session was my first responsibility as a Scrum Master. During this session, I sketched out the dependencies between each task, assisted in defining what each team member would consider "done," and made sure that nobody was blocked right away.
I kept an organised Trello board with task columns labelled "To Do," "In Progress," "Testing," and "Done" and added subtasks, labels, and target deadlines to oversee sprint progress. Each team member reported their progress on the board, and I followed up frequently to resolve problems, provide technical assistance, and redistribute tasks as needed.

An essential component of my leadership was communication. We used voice notes, live screen shares, and short stand-ups when needed, and we coordinated every day via a shared WhatsApp group. I also came in one-on-one to assist with wiring verification, servo calibration, and code inspections. This kept the momentum going while accommodating each team member's availability and speed.

One instance of this occurred when Saeed encountered servo stability problems while honing his walking technique (Task 1). I recommended using staggered movement sequences, which I had already created in Task 4, while honing my wing reaction logic. Likewise, I collaborated with Adham to implement delay buffers and smoothing logic based on real-time sensor feedback when he encountered sensitivity problems with his object avoidance levels.

I completed my Task (Predator Response) while also assisting the team. This task involved using ultrasonic sensors to detect rapidly approaching objects and trigger a defensive wing response. With the PCA9685 PWM controller, I could integrate threat detection, timed servo movement, and real-time distance awareness. I conducted live demonstrations to ensure stability before integration, maintained accessible code for reference, and kept my process documentation open.

After completing each task, I guided the group through a planned integration stage. Each behaviour was checked one after the other to ensure there were no power problems, pin conflicts, or performance snags. In addition to testing system stability under load and ensuring that sensor polling from various scripts did not conflict with one another, I oversaw the final wiring architecture. This stage was essential to producing a functional demo that met our objectives.

It is crucial to remember that we failed to complete all of the group video submission assignments before the due date. Even after submission, I kept up my productive and close collaboration with the team to ensure the remaining behaviours were carried out. Consequently, our robot was completely operational, and the project's conclusion had finished all tasks. Beyond what was shown in the initial group film, we would truly welcome the chance to demonstrate the entire system in operation. We are proud of how far we have pushed the development.

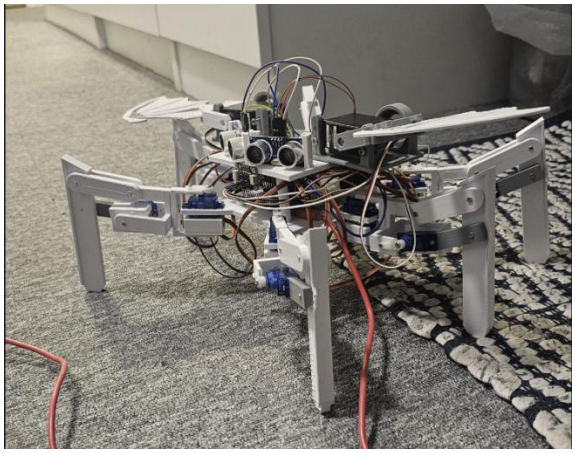

Figure 8 *Fully Integrated Robot*

After considering this experience, I saw how important it is to be a Scrum Master to delegate and support. I developed my ability to swiftly solve issues, manage team dynamics, and foster an atmosphere that allowed advancement despite adversity. The most important lesson I learnt was the need for proactive communication and visibility: if work was "stuck" for one team member, it might easily affect another. Therefore, it was crucial to address problems as soon as possible.

As Scrum Master, I ensured that every member contributed usefully and that every task was finished. I also led integration to a successful conclusion, unblocked development, and preserved the sprint structure. Above all, I maintained the team's commitment to a common goal and alignment. I will apply what I learnt from this sprint to all of my future engineering endeavours because it taught me how to lead not just individually but also cooperatively.

APPENDIX A

```
import time
import board
from adafruit_hcsr04 import HCSR04

#  Define which pins your sensors use:
#    Sensor 1: trigger on GP2, echo on GP3
#    Sensor 2: trigger on GP4, echo on GP5
sonar1 = HCSR04(trigger_pin=board.GP2, echo_pin=board.GP3)
sonar2 = HCSR04(trigger_pin=board.GP4, echo_pin=board.GP5)

#  Main loop: read and print both distances
while True:
    try:
        dist1 = sonar1.distance  # in centimetres
    except RuntimeError:
        dist1 = None          # timeout or out of range

    try:
        dist2 = sonar2.distance
    except RuntimeError:
        dist2 = None

    # Print nicely, handle None
    d1 = f"{dist1:.1f} cm" if dist1 is not None else "Out of range"
    d2 = f"{dist2:.1f} cm" if dist2 is not None else "Out of range"
    print(f"Sensor 1: {d1}    Sensor 2: {d2}")

    time.sleep(0.5)
```

Testing Coded By Adham Osman

APPENDIX B

```
import time
import board
import busio
from adafruit_pca9685 import PCA9685
import adafruit_bh1750

# Setup I2C
# GP0 = SDA, GP1 = SCL
i2c = busio.I2C(scl=board.GP1, sda=board.GP0)

# Setup PCA9685 (no servos attached)
pca = PCA9685(i2c)
pca.frequency = 50

# Setup BH1750 light sensor
light_sensor = adafruit_bh1750.BH1750(i2c)

# Main loop: read and print light level
while True:
    lux = light_sensor.lux
    print(f"Light level: {lux:.2f} lux")
    time.sleep(1)
```

Testing coded By Isaac Anene

APPENDIX C

```
import time
import board
import busio
from adafruit_hcsr04 import HCSR04
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo

# Initialize I2C & PCA9685
i2c = busio.I2C(scl=board.GP1, sda=board.GP0)
pca = PCA9685(i2c)
pca.frequency = 50  # Standard servo frequency

# Servo setup — CORRECTED sides
# Channel 12 is reversed (physically RIGHT wing)
# Channel 13 is normal (physically LEFT wing)
wing_right = servo.Servo(pca.channels[12], min_pulse=600, max_pulse=2400)  # reversed
wing_left  = servo.Servo(pca.channels[13], min_pulse=600, max_pulse=2400)  # normal

# Wing positions based on your tests
WING_DOWN_LEFT  = 120   # channel 13
WING_UP_LEFT    = 180
WING_DOWN_RIGHT = 150   # channel 12
WING_UP_RIGHT   = 90

def raise_wings():
    wing_left.angle  = WING_UP_LEFT
    wing_right.angle = WING_UP_RIGHT
    print(" Rapid approach detected! WINGS UP")

def lower_wings():
    wing_left.angle  = WING_DOWN_LEFT
    wing_right.angle = WING_DOWN_RIGHT
    print(" Wings down")

# Ultrasonic sensors (forward-facing)
sonar1 = HCSR04(trigger_pin=board.GP2, echo_pin=board.GP3)
sonar2 = HCSR04(trigger_pin=board.GP4, echo_pin=board.GP5)

# Initialize
```

```
lower_wings()
prev_d1 = None
prev_d2 = None

print(" Threat detection active...")

# Main loop — raise wings on rapid approach
while True:
    try:
        d1 = sonar1.distance
    except RuntimeError:
        d1 = None

    try:
        d2 = sonar2.distance
    except RuntimeError:
        d2 = None

    # Check for rapid change
    threat = False
    if prev_d1 is not None and d1 is not None:
        if prev_d1 - d1 > 40:
            threat = True
    if prev_d2 is not None and d2 is not None:
        if prev_d2 - d2 > 40:
            threat = True

    if threat:
        raise_wings()
        time.sleep(2)  # Hold wings up for 2 seconds
    else:
        lower_wings()

    # Update previous distances
    if d1 is not None:
        prev_d1 = d1
    if d2 is not None:
        prev_d2 = d2

    # Print debug
```

```
    d1_str = f"{d1:.1f} cm" if d1 is not None else "out"
    d2_str = f"{d2:.1f} cm" if d2 is not None else "out"
    print(f"Sonar1: {d1_str}  |  Sonar2: {d2_str}")

    time.sleep(0.2)
```