

OTTO Bipedal Robot Control System Based on Arduino UNO Q

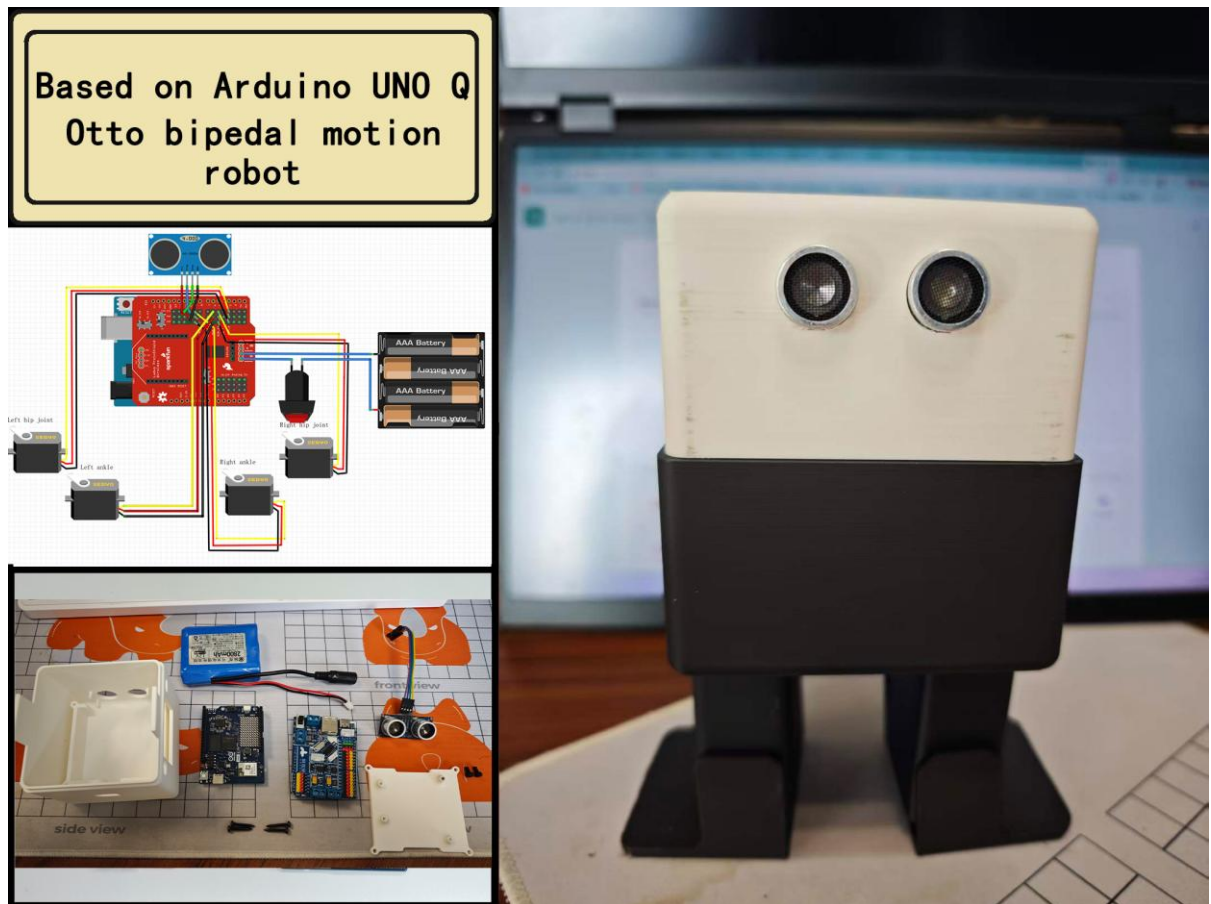
Tags

- Audio
- Remote Control
- Robots
- Web Control
- Music Synchronization

Difficulty Level

Intermediate — This project involves multiple technology layers: hardware assembly, Arduino firmware programming, Python backend development, and web frontend creation. While beginners can complete the basic setup with guidance, implementing advanced features like music synchronization requires understanding both hardware and software concepts.

Cover Image



Cover Image

Hardware Components

Building your OTTO robot requires several key components. Each part plays a crucial role in enabling this bipedal robot to walk, dance, and interact with music. Let me walk you through each component and explain its importance.

Servo Motors (4x DS3115/Mg996r)

The heart of OTTO' s mobility lies in four digital servo motors. These aren' t just any servos—they' re specifically chosen for their balance of torque, precision, and affordability. Each servo controls one joint:

- **Left Hip (YL):** Connected to digital pin D3, this servo drives the entire left leg forward and backward—the primary driver for leg movement during walking
- **Right Hip (YR):** Connected to D5, controlling the right leg's movement, coordinating with the left hip for alternating steps
- **Left Ankle (RL):** Connected to D6, managing left foot tilt up and down—critical for balance control during single-leg support phases
- **Right Ankle (RR):** Connected to D9, handling right foot tilt adjustments to keep the robot stable

The DS3115 servos operate within a 0-180 degree range, with 90 degrees representing the neutral standing position. They provide 2.5 kg-cm of torque—sufficient for the lightweight OTTO frame while maintaining precise control. The symmetric distribution between left and right sides creates a mirror image that enables natural bipedal locomotion.



OTTO Robot Side View

Figure 1: Assembled OTTO robot side view showing servo positions

Custom Expansion Board

A carefully designed custom expansion board serves multiple critical functions that keep everything working together:

- **Power distribution:** Safely routes battery voltage to all components
- **Servo signal routing:** Connects Arduino digital pins to each servo motor

- **Voltage regulation:** Ensures stable 5V supply, preventing servo jitter and damage
- **I/O expansion:** Provides additional pin access for future sensors like ultrasonic distance sensors

Arduino UNO Q Development Board

The brain of the operation is Arduino UNO Q—a modern evolution of the classic Arduino platform. What makes UNO Q special is its built-in WiFi capability and web-based programming environment (Arduino App Lab). This means once the robot is configured, it no longer needs a separate computer, enabling truly standalone operation.

The board runs at 48MHz with 32KB of flash memory, providing ample processing power for real-time motion control and wireless communication.

Power System

The robot runs on a 12V 2800mAh lithium battery, providing several hours of operation. Voltage is regulated to 5V for servos and Arduino, ensuring consistent performance even as the battery depletes. This means OTTO can operate completely independently from any computer.

3D Printed Frame

The mechanical body consists of 3D printed components using PLA or PETG filament. The design includes:

- **Torso shell:** Houses electronics and battery in a compact package
- **Hip brackets:** Secure mounting points for hip servos
- **Leg segments:** Lightweight connectors linking hips to ankles

- **Foot plates:** Wide contact surfaces providing stable ground contact

All parts use press-fit connections and dedicated screws for tool-free assembly, making the build process accessible to beginners.

Ultrasonic Sensor (HC-SR04)

For advanced obstacle detection capabilities, an HC-SR04 ultrasonic sensor can be added. This enables the robot to detect and respond to objects in its path, adding autonomous navigation possibilities.

Complete Hardware Bill of Materials

Component	Quantity	Model	Purpose
Servo Motors	4	DS3115	Leg joint control (left hip, right hip, left ankle, right ankle)
Expansion Board	1	Custom	Power distribution and I/O expansion
Development Board	1	Arduino UNO Q	Main controller running robot software
USB Cable	1	USB-C to USB-A	Connect board to computer for programming

Ultrasonic Sensor	1	HC-SR04	Obstacle detection
		12V	
Battery	1	2800mAh Li-ion	Autonomous power supply
C to C Cable	1	Custom	Connect expansion board to Arduino
3D Printed Parts	1 set	OTTO model	Mechanical body structure
Screws	Various	M2, M3	Assembly hardware

Tools and Machines

- **Personal Computer:** For programming and initial configuration
 - **3D Printer:** For printing OTTO body components (PLA/PETG recommended)
 - **Soldering Kit:** For any necessary wire connections
 - **Screwdriver Set:** For assembly (primarily Phillips head)
-

Development Platform

- **Arduino App Lab:** Web IDE for programming Arduino UNO Q
- **Python 3.13+:** Backend motion algorithm processing
- **Modern Web Browser:** Chrome, Firefox, or Edge for control interface

Project Description

Introduction

OTTO is one of the most popular bipedal robots in the DIY robotics community. Its open-source nature and charming appearance make it an ideal platform for learning robotics. This project builds on the classic OTTO robot design, introducing modern web technologies to achieve browser-based remote control and music-synchronized dancing. The entire system uses Arduino UNO Q as the core controller, combined with Python backend algorithms and JavaScript frontend interface, constructing a complete robot control system.

The project's main features include: no applications to install—control the robot directly through the browser; music synchronization capability allowing the robot to dance to music beats; virtual piano interface where playing notes triggers corresponding robot movements; rich preset action library supporting user-defined action sequences. The entire system is highly modular with clear code structure, making it an excellent learning case for combining robotics development with web technologies.

What makes this project special?

First, no software installation required. Everything runs in the browser—you control the robot by simply visiting a web page. This means you can operate OTTO from any device: laptop, tablet, or even your phone.

Second, the music synchronization feature transforms OTTO from a simple robot into an entertainment star. Load your favorite song, enable dance mode, and watch OTTO move to the beat. The robot analyzes the music and executes choreographed movements matching the rhythm.

Third, the virtual piano interface creates a unique musical experience. When you play notes on the piano keyboard, each note triggers a corresponding robot motion. Low notes produce slow, graceful movements while high notes cause energetic jumps. This transforms music playback into an interactive performance.

Fourth, the extensive action library includes over 14 different movements—from basic walking and turning to complex dance moves like moonwalking. These can be combined into custom sequences for endless entertainment possibilities.



OTTO Robot Front View

Figure 2: OTTO front view showing symmetric leg structure

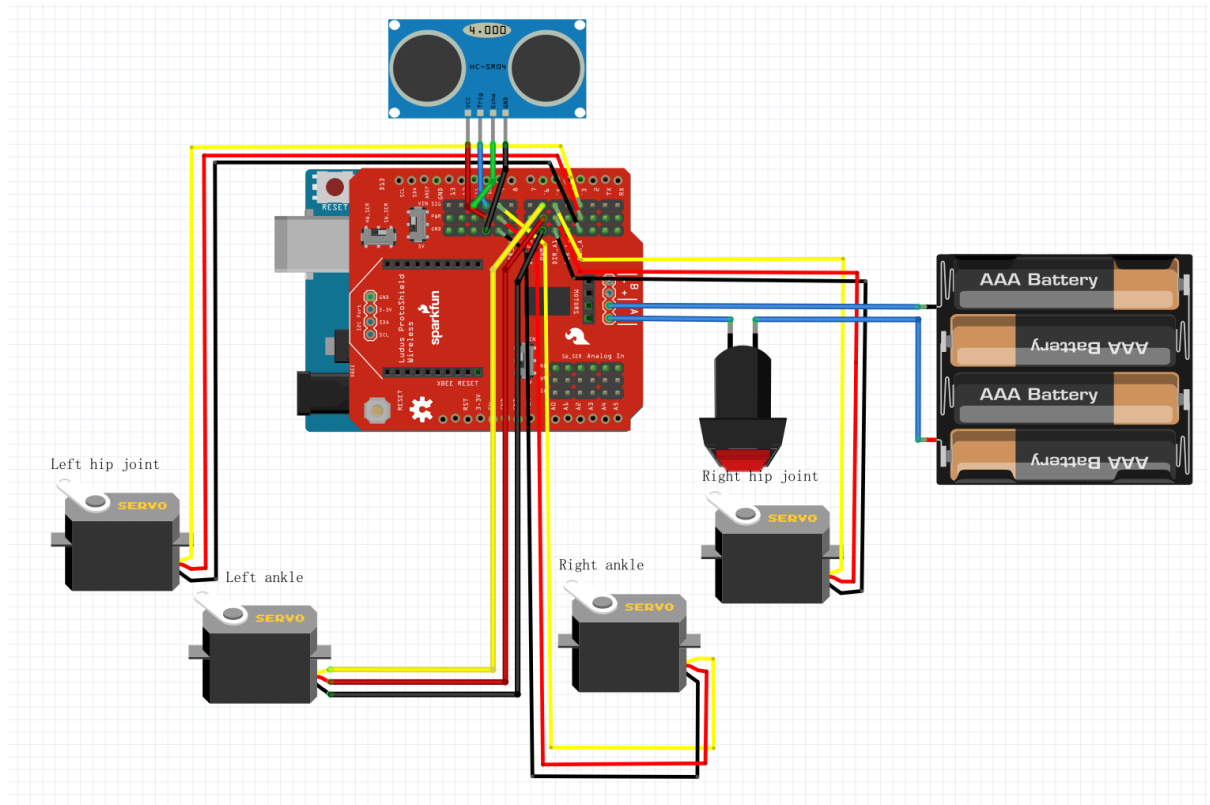
Circuit Diagram & Pin Configuration

Understanding pin configuration is essential for proper assembly and troubleshooting. OTTO uses a four-servo structure to achieve bipedal locomotion, with each servo responsible for one joint movement.

Pin Assignment:

- **D3 (YL-Left Hip):** Controls left leg forward/backward swing—the primary driver for leg movement during walking
- **D5 (YR-Right Hip):** Controls right leg swing movement. Works with left hip for alternating steps
- **D6 (RL-Left Ankle):** Manages left foot up/down tilt—critical for balance during single-leg support
- **D9 (RR-Right Ankle):** Controls right foot tilt. Moves opposite to hip for center of gravity transfer during walking

Each servo can rotate from 0 to 180 degrees, with 90 degrees representing the neutral (horizontal) position. Servos are connected in symmetric configuration, creating a mirror image between left and right sides.



Circuit Diagram

Figure 3: Servo pin assignment and connection diagram

Hardware Architecture

The system follows a three-tier architecture that separates concerns while maintaining efficient communication:

Layer 1: Web Frontend (Browser)

The user interface runs entirely in the browser, built with HTML5, CSS3, and JavaScript. It provides three main functional areas:

- **Servo Test Page:** Individual slider controls with position presets (45°, 90°, 135°)
- **Virtual Piano:** Interactive keyboard that triggers robot movements based on note frequency

- **Action Control:** Button-based control panel with music playback and dance synchronization

Communication between browser and backend uses WebSocket ([Socket.IO](#)) for real-time, bidirectional messaging.

Layer 2: Python Backend

The Python server handles business logic and motion algorithm computation:

- **main.py:** Web server initialization and API request handling
- **otto_core.py:** Core motion algorithms including oscillator calculations and action generation
- **otto_controller.py:** Hardware interface abstraction and command translation

Layer 3: Arduino Firmware

The microcontroller runs optimized C++ code for real-time servo control:

- Servo angle command execution
- RouterBridge communication protocol handling
- Hardware initialization and safety checks

The key innovation here is separating motion planning (Python) from motion execution (Arduino). Python handles complex algorithm calculations without timing constraints, while Arduino focuses on precise, low-latency servo control.

Feature Description

1. Browser-Based Control

No apps to install, no software to maintain. Open the web interface, connect to your robot, and start controlling. The interface works on any modern browser and adapts to different screen sizes.

2. Virtual Piano with Frequency-Action Mapping

The piano interface maps musical notes to robot actions based on frequency:

- **Low frequencies (C3-G3):** Slow, graceful movements like bending and leg shaking
- **Medium frequencies (C4-B4):** Standard movements including swinging and tiptoeing
- **High frequencies (C5-G5):** Energetic actions like jumping and flapping

This creates an intuitive mapping where musical pitch naturally corresponds to movement intensity.

3. Music Synchronized Dance

Load any of the four pre-loaded songs or upload your own MP3 files. Enable “Dance Mode” and OTTO will perform choreographed movements synchronized to the music. The system parses the audio to detect rhythm and triggers appropriate actions at each beat.

4. Preset Dance Sequences

Three built-in dance routines demonstrate the system’s capabilities:

- **Happy:** Swing → Walk Forward → Walk Backward → Tiptoe Swing → Home
- **Dance:** Moonwalker Left → Moonwalker Right → Crusaito Left → Crusaito Right → Flapping → Home

- **Greeting:** Swing → Bend Left → Bend Right → Walk Forward →

Home

5. Custom Action Sequences

The JSON-based action sequence format allows creating custom dances:

python

```
custom_dance = [  
    {'action': 'walk', 'steps': 2, 'T': 1000, 'direction': 1},  
    {'action': 'moonwalker', 'steps': 3, 'T': 900, 'height': 25,  
    'direction': 1},  
    {'action': 'jump', 'steps': 1, 'T': 1500}  
]
```

Core Algorithms

The core technology of this project lies in enabling stable walking for a bipedal robot with only four servos. This all depends on carefully designed algorithms—center of mass transfer principle and sine wave oscillators.

1. Center of Mass Transfer Principle for Bipedal Robots

The fundamental challenge in bipedal walking is maintaining balance while transitioning between legs. Humans accomplish this through complex vestibular systems and cerebellum coordination, but for robots, we need to simulate this process using mathematical methods. The center of mass transfer principle used by OTTO is one of the most classic and effective solutions in the bipedal robot field.

In human movement biomechanics, walking is understood as a continuous center of mass transfer process. When we step out with our left foot, our body weight shifts toward the left leg, at which point the right foot lifts and swings forward; as the right foot lands, weight gradually transfers to the right leg, and the left foot then lifts. Throughout this process, the center of mass remains within or above the support area (the area formed by both feet), maintaining balance.

For bipedal robots like OTTO, we need to implement a similar balancing mechanism in a simplified mechanical structure with only four servos. The core idea of center of mass transfer can be summarized in four steps:

1. **Weight transfer:** Shift body weight to the supporting leg, ensuring no tipping during single-foot support
2. **Leg lift:** Swing the other leg forward, lifting it off the ground
3. **New support foot landing:** Transfer weight to the new supporting leg before the swinging foot lands
4. **Repeat:** The original supporting leg lifts, entering the next gait cycle

In OTTO' s implementation, center of mass transfer is accomplished by coordinating the movements of four servos. The key is the **90-degree phase difference** between hip and ankle joints, which ensures that when one leg swings forward, the same-side foot tilts appropriately to transfer the center of mass to the other supporting leg.

Here' s the walking action implementation code extracted from `otto_core.py`:

python

```
def walk(self, steps: int = 2, T: int = 1000, direction: int = FORWARD) -> List[ServoAngles]:
    """
    Walking action - stable walking using center of mass transfer principle

    Parameters:
    - steps: Number of steps (complete gait cycle count)
    - T: Cycle time in milliseconds (600-1400, larger = slower)
    - direction: Walking direction, FORWARD(1) or BACKWARD(-1)
    """
    # Amplitude configuration: [left hip, right hip, left ankle, right ankle]
    # Hip amplitude of 30 degrees determines leg swing range
    # Ankle amplitude of 15 degrees determines foot tilt extent
    A = [30, 30, 15, 15]
```

```

    # Offset configuration for fine-tuning ankle initial position
    # 4-degree deviation between left and right ankles compensates
    for mechanical asymmetry
    O = [0, 0, 4, -4]

    # Phase difference configuration - key to implementing center of
    mass transfer
    # -90 degree phase difference between ankle and hip ensures
    coordinated movement
    phase_diff = [0, 0, deg2rad(direction * -90), deg2rad(direction *
    -90)]

    return self._execute_oscillation(A, O, T, phase_diff, steps)

```

The physical meaning of walking parameters:

Amplitude determines the movement range of each servo:

- A[0] = 30: Left hip joint swing range ± 30 degrees
- A[1] = 30: Right hip joint swing range ± 30 degrees
- A[2] = 15: Left ankle up/down tilt range ± 15 degrees
- A[3] = 15: Right ankle up/down tilt range ± 15 degrees

Phase Difference is the essence of the entire gait algorithm:

- phase_diff[0] = 0: Left hip phase as reference
- phase_diff[1] = 0: Right hip in phase with left hip
- phase_diff[2] = $-\pi/2$: Left ankle lags left hip by 90 degrees
- phase_diff[3] = $-\pi/2$: Right ankle lags right hip by 90 degrees

2. Sine Wave Oscillator Algorithm

Human and animal movements are often characterized by periodicity and smoothness because biological musculoskeletal systems are essentially precision spring-damper systems. When we wave or walk,

the trajectory of arm or leg movement can be well approximated using sine functions.

Choosing sine functions for several important reasons:

1. **Infinite differentiability:** Sine functions are infinitely differentiable, with continuous velocity (first derivative) and acceleration (second derivative) changes—no sudden jumps. This makes movement trajectories smooth and natural.
2. **Periodicity:** Sine functions are inherently periodic, perfect for representing repetitive periodic movements like walking and swinging.
3. **Mathematical simplicity:** Sine functions have concise integral and differential forms, facilitating kinematic analysis and parameter adjustment.
4. **Few adjustable parameters:** Only four parameters—amplitude, offset, period, and phase—can generate rich movement patterns.

The mathematical principle of the sine wave oscillator can be expressed as:

```
Position = Amplitude × sin(Current Phase + Initial Phase) + Offset
```

Converted to servo angle:

```
Angle = Position + 90 + Calibration Offset
```

Here' s the complete Oscillator class implementation:

python

```

class Oscillator:
    """Sine wave oscillator class - for generating smooth periodic motion"""

    def __init__(self, trim: float = 0.0):
        self._A = 0.0          # Amplitude, initially 0
        self._O = 0.0          # Offset, initially 0
        self._T = 2000.0       # Period (ms), default 2 seconds
        self._phase0 = 0.0     # Initial phase
        self._trim = trim      # Calibration offset
        self._phase = 0.0      # Current phase, initially 0
        self._pos = 90.0       # Current position, initially center
        self._rev = False      # Reverse flag, default False

        # Sampling: 30ms per sample, approximately 33Hz update rate
        self._TS = 30
        self._update_period_params()

    def refresh(self, current_time: float = None) -> float:
        """Refresh oscillator state and return current position"""
        if current_time is None:
            current_time = time.time() * 1000

        # Calculate current position using sine function
        pos = self._A * math.sin(self._phase + self._phase0) +
self._O

        if self._rev:
            pos = -pos

        # Convert to servo angle (0-180 range, center at 90)
        self._pos = round(pos) + 90 + self._trim

        # Clamp angle to valid range
        self._pos = max(0, min(180, self._pos))

        # Update phase
        self._phase += self._inc

        return self._pos

```

The oscillator runs at a 30ms sampling rate (~33 FPS), which exceeds the human eye's motion continuity threshold (typically 24 fps is sufficient) while not overburdening Arduino communication.

3. Motion Execution Pipeline

The motion execution pipeline is the critical bridge connecting motion algorithms to actual servo control. It receives motion parameters,

generates a sequence of servo angle frames, and sends them to Arduino for execution.

python

```
def _execute_oscillation(self, A: List[float], O: List[float],
                        T: float, phase_diff: List[float],
                        steps: float = 1.0,
                        sample_rate: float = 30) -> List[ServoAngles]:
    """
    Execute synchronized oscillation motion on all four servos
    """
    frames = []

    # Step 1: Configure each oscillator
    for i in range(SERVO_COUNT):
        self._oscillators[i].SetA(A[i])
        self._oscillators[i].SetO(O[i])
        self._oscillators[i].SetT(T)
        self._oscillators[i].SetPh(phase_diff[i])

    # Step 2: Calculate total frames needed
    total_samples = int((T * steps) / sample_rate)

    # Step 3: Generate motion frame sequence
    for i in range(total_samples):
        angles = []
        for j in range(SERVO_COUNT):
            pos = self._oscillators[j].refresh()
            angles.append(int(pos))
        frame = ServoAngles(angles[0], angles[1], angles[2],
angles[3])
        frames.append(frame)

    return frames
```

4. Action Library Deep Dive

The project implements a rich action library, where each action achieves unique movement effects by adjusting oscillator parameters (amplitude, offset, phase difference).

Moonwalker - Michael Jackson' s classic dance move:

python

```
def moonwalker(self, steps: int = 3, T: int = 1000, height: int = 25,
               direction: int = LEFT):
    # Only ankle movement, hip stays stationary
    A = [0, 0, height, height]
```

```
# Ankle offset creates tilt effect
O = [0, 0, height / 2 + 2, -height / 2 - 2]
# Phase difference is key to "sliding" effect
phi = -direction * 90
phase_diff = [0, 0, deg2rad(phi), deg2rad(-60 * direction + phi)]

return self._execute_oscillation(A, O, T, phase_diff, steps)
```

The characteristic of moonwalking is body tilt while feet slide on the ground—only ankle joints move while hips remain stationary. The key is the -60 degree phase difference between left and right ankles, which creates the “sliding” effect.

Flapping - Simulating bird wing flapping:

python

```
def flapping(self, steps: int = 2, T: int = 1000, height: int = 20,
             direction: int = FORWARD):
    # Key: 180 degree phase difference between left and right hips
    A = [12, 12, height, height]
    O = [0, 0, height - 10, -height + 10]
    phase_diff = [0, deg2rad(180), deg2rad(-90 * direction),
deg2rad(90 * direction)]

    return self._execute_oscillation(A, O, T, phase_diff, steps)
```

A 180-degree phase difference means when the left leg goes up, the right leg goes down; when the left leg goes down, the right leg goes up, creating alternating up-down movement.

By understanding these three core concepts—**center of mass transfer**, **sine wave oscillator**, and **phase difference**—you can create your own new actions! All actions follow the same design pattern: generating different movement effects by adjusting the three parameters of amplitude, offset, and phase difference.

Usage Instructions

Initial Setup:

1. Assemble the OTTO robot according to the 3D printed parts guide

2. Connect servos to designated pins (D3, D5, D6, D9)
3. Connect Arduino UNO Q to computer via USB
4. Open Arduino App Lab and load the OttoControlWebVoice project
5. Configure WiFi credentials (password: 12345678)
6. Click "Run" to start the program

Accessing Control Interface:

1. After startup, the console displays a web address (e.g., <http://192.168.1.100>)
2. Open this address in your web browser
3. Click "Connect" to establish communication with the robot

Testing Servos:

1. Navigate to the Servo Test page
2. Use sliders to test each servo individually
3. Verify movement range and check for mechanical binding
4. Use preset buttons (45°, 90°, 135°) for quick testing

Controlling Actions:

1. Go to the Action Control page
2. Click any action button to execute that movement
3. Adjust the speed slider to modify action timing

4. Use the music player to test dance synchronization

Virtual Piano:

1. Navigate to the Piano page
2. Click or tap keys to play notes
3. Each note produces sound AND triggers a robot motion
4. Experiment with different instruments for varied responses