

Phase 5.1: Quadruped Scaling

การขยายระบบจากขาเดียวเป็นหุ่นยนต์สี่ขาแบบสมมาตร

BLEGS Analysis Unit

นายธีรโชติ เมืองจันทน์

BLEGS Quadruped Robot Project

2 มกราคม 2026

สารบัญ

1 บทนำ (Introduction)

1.1 วัตถุประสงค์

Phase 5.1 มีวัตถุประสงค์เพื่อขยายระบบควบคุมจากขาเดียว (Single Leg) เป็นหุ่นยนต์สี่ขาแบบสมบูรณ์ (Full Quadruped) โดยพัฒนาระบบ Motor indexing, Mirror kinematics สำหรับขาซ้าย-ขวา และ Multi-leg synchronization เพื่อให้หุ่นยนต์สามารถเดินได้จริงบนฮาร์ดแวร์

1.2 ขอบเขตการศึกษา

- การออกแบบระบบ Motor Indexing สำหรับ 8 มอเตอร์
- การพัฒนา Mirror Kinematics สำหรับขาซ้าย-ขวา
- การออกแบบ Gait Pattern สำหรับ Trot gait
- การพัฒนา Multi-leg Synchronization ด้วย Threading
- การทดสอบระบบ IK ทั้ง 4 ขา (Quadruped IK Testing)
- การทดสอบหุ่นยนต์สี่ขาแบบสมบูรณ์บนฮาร์ดแวร์จริง

1.3 ความเชื่อมโยงกับ Phase ก่อนหน้า

Phase 5.1 สร้างต่อจาก Phase 4.2 โดยนำระบบควบคุมขาเดียวมาขยายเป็น 4 ขา:

- ใช้ Binary Protocol v1.1 จาก Phase 4.2
- ใช้ Controller design และ Trajectory generator จาก Phase 4.1
- ใช้ Gait pattern จากการจำลอง Phase 3.1
- นำบทเรียนจากการแก้ Motor jitter มาปรับใช้ทั้ง 8 มอเตอร์

2 Motor Indexing System

2.1 ระบบการตั้งชื่อขา (Leg Naming Convention)

หุ่นยนต์สี่ขามีขาทั้งหมด 4 ข้าง ตั้งชื่อตามตำแหน่ง:

- FL (Front-Left): ขาซ้ายหน้า
- FR (Front-Right): ขาขวาน้ำ
- RL (Rear-Left): ขาซ้ายหลัง
- RR (Rear-Right): ขาขวาหลัง

2.2 Motor Indexing Scheme

แต่ละขามีมอเตอร์ 2 ตัว (Thigh และ Shank) รวมเป็น 8 มอเตอร์ ตั้งชื่อดังนี้:

Leg	Position	Thigh Motor	Shank Motor	Motor IDs
FL	Front-Left	Motor A	Motor B	1, 2
FR	Front-Right	Motor A	Motor B	3, 4
RL	Rear-Left	Motor A	Motor B	5, 6
RR	Rear-Right	Motor A	Motor B	7, 8

ตารางที่ 1: Motor Indexing Scheme สำหรับหุ่นยนต์สี่ขา

2.3 การคำนวณ Motor ID

สูตรคำนวณ Motor ID จาก Leg index และ Joint type:

$$\text{Thigh Motor ID} = 2 \times \text{Leg_index} + 1 \quad (1)$$

$$\text{Shank Motor ID} = 2 \times \text{Leg_index} + 2 \quad (2)$$

โดย Leg index คือหมายเลขประจำขาแต่ละข้าง และ Motor IDs คือรหัสมอเตอร์ที่คำนวณได้จากสูตรข้างต้น:

- **FL (Leg index = 0):** คำนวณได้ Thigh Motor ID = $2 \times 0 + 1 = 1$ และ Shank Motor ID = $2 \times 0 + 2 = 2$ ดังนั้นขา FL ใช้ Motors 1 และ 2
- **FR (Leg index = 1):** คำนวณได้ Thigh Motor ID = $2 \times 1 + 1 = 3$ และ Shank Motor ID = $2 \times 1 + 2 = 4$ ดังนั้นขา FR ใช้ Motors 3 และ 4
- **RL (Leg index = 2):** คำนวณได้ Thigh Motor ID = $2 \times 2 + 1 = 5$ และ Shank Motor ID = $2 \times 2 + 2 = 6$ ดังนั้นขา RL ใช้ Motors 5 และ 6
- **RR (Leg index = 3):** คำนวณได้ Thigh Motor ID = $2 \times 3 + 1 = 7$ และ Shank Motor ID = $2 \times 3 + 2 = 8$ ดังนั้นขา RR ใช้ Motors 7 และ 8

2.4 ตัวอย่างโค้ด Python

Listing 1: Motor Indexing Helper Functions

```
1 class LegIndex :
2     """ Leg index enumeration """
3     FL = 0    # Front - Left
4     FR = 1    # Front - Right
5     RL = 2    # Rear - Left
6     RR = 3    # Rear - Right
7
8     def get_motor_ids ( leg_index ) :
```

```

9      """
10     Get motor IDs for a specific leg
11
12     Parameters :
13     -----
14     leg_index : int
15         Leg index (0-3)
16
17     Returns :
18     -----
19     thigh_id : int
20         Thigh motor ID
21     shank_id : int
22         Shank motor ID
23     """
24
25     thigh_id = 2 * leg_index + 1
26     shank_id = 2 * leg_index + 2
27     return thigh_id , shank_id
28
29 # Example usage
30 fl_thigh , fl_shank = get_motor_ids ( LegIndex . FL )    #
    Returns (1 , 2)
    fr_thigh , fr_shank = get_motor_ids ( LegIndex . FR )    #
    Returns (3 , 4)

```

3 Mirror Kinematics

3.1 ความแตกต่างระหว่างขาซ้ายและขาขวา

กลไก 5-Bar Linkage ของขาซ้ายและขาขวามีความแตกต่างกัน:

3.1.1 Left Legs (FL, RL)

- Motor A position: $(-42.5, 0)$ mm
- Motor B position: $(+42.5, 0)$ mm
- Coordinate frame: Standard (same as Phase 1)

3.1.2 Right Legs (FR, RR)

- Motor A position: $(+42.5, 0)$ mm (Mirrored)
- Motor B position: $(-42.5, 0)$ mm (Mirrored)
- Coordinate frame: X-axis flipped

3.2 Trajectory Mirroring

เพื่อให้ขาขวาเคลื่อนที่สมมาตรกับขาซ้าย ต้องทำ Mirror transformation:

3.2.1 สมการ Mirroring

สำหรับ Trajectory $P_F(x, y)$ ของขาซ้าย:

$$P_{F, right} = \begin{bmatrix} -x \\ y \end{bmatrix} \quad (3)$$

หมายเหตุ: Mirror เฉพาะพิกัด X เท่านั้น (Y ไม่เปลี่ยน)

3.2.2 ตัวอย่าง

Left Leg	Right Leg	Interpretation
(+30, -200)	(-30, -200)	ก้าวไปข้างหน้า (Forward)
(-30, -200)	(+30, -200)	ก้าวไปข้างหลัง (Backward)
(0, -170)	(0, -170)	ยกขาขึ้น (Lift)

ตารางที่ 2: ตัวอย่าง Trajectory Mirroring

3.3 IK Configuration สำหรับขาซ้าย-ขวา

การเลือก IK configuration ต้องพิจารณาความแตกต่างระหว่างขา:

Leg Type	Motor Layout	IK Config	Elbow Direction
Left (FL, RL)	A=-42.5, B=+42.5	Config 1	Down-Down
Right (FR, RR)	A=+42.5, B=-42.5	Config 1	Down-Down

ตารางที่ 3: IK Configuration สำหรับแต่ละขา

หมายเหตุ: ใช้ Config 1 (Down-Down) ทั้งหมดเพราะ:

- Torque สมดุล (จาก Phase 2.1)
- มุมมอเตอร์อยู่ในช่วงปลอดภัย
- Workspace กว้าง

3.4 ตัวอย่างโค้ด Python

Listing 2: Mirror Kinematics Implementation

```
1 def mirror_trajectory_for_right_leg(trajectory_left):
2     """
```

```

3     Mirror trajectory from left leg to right leg
4
5     Parameters :
6     -----
7     trajectory_left : array (N, 2)
8         Trajectory for left leg [(x, y), ...]
9
10    Returns :
11    -----
12    trajectory_right : array (N, 2)
13        Mirrored trajectory for right leg
14    """
15    trajectory_right = trajectory_left.copy()
16    trajectory_right[:, 0] = -trajectory_right[:, 0]
17    # Mirror X
18    return trajectory_right
19
20 def calculate_ik_for_leg (leg_index , trajectory):
21     """
22     Calculate IK for a specific leg
23
24     Parameters :
25     -----
26     leg_index : int
27         Leg index (0-3: FL, FR, RL, RR)
28     trajectory : array (N, 2)
29         Foot trajectory [(x, y), ...]
30
31     Returns :
32     -----
33     theta_A : array
34         Motor A angles (degrees)
35     theta_B : array
36         Motor B angles (degrees)
37     """
38     # Check if right leg (FR or RR)
39     if leg_index == LegIndex.FR or leg_index ==
40         LegIndex.RR:
41         # Mirror trajectory for right legs
42         trajectory = mirror_trajectory_for_right_leg(
43             trajectory)
44
45     # Calculate IK (Config 1 for all legs)

```

```

43     n_points = len(trajectory)
44     theta_A = np.zeros(n_points)
45     theta_B = np.zeros(n_points)
46
47     for i in range(n_points):
48         P_F = trajectory[i]
49         theta_A[i], theta_B[i] =
            calculate_ik_analytical(P_F, config=1)
50
51     return theta_A, theta_B

```

4 Gait Pattern Design

4.1 Trot Gait Pattern

Trot gait เป็น Gait pattern ที่เหมาะสำหรับหุ่นยนต์สี่ขา โดยมีลักษณะ:

- **Diagonal pairs:** ขาทแยงมุมเคลื่อนที่พร้อมกัน
- **Pair 1:** FR + RL (Front-Right + Rear-Left)
- **Pair 2:** FL + RR (Front-Left + Rear-Right)
- **Phase offset:** 180° (ครึ่งรอบ)

4.2 Gait Timing

Time	FL	FR	RL	RR
0 ms	Stance	Swing	Swing	Stance
300 ms	Swing	Stance	Stance	Swing
600 ms	Stance	Swing	Swing	Stance

ตารางที่ 4: Trot Gait Timing (Cycle = 600 ms)

Stance phase: เท้าสัมผัสพื้น (Supporting)

Swing phase: เท้ายกขึ้น (Transferring)

4.3 Phase Offset Calculation

สูตรคำนวณ Phase offset สำหรับแต่ละขา:

$$\phi_{FL} = 0^\circ \quad (4)$$

$$\phi_{FR} = 180^\circ \quad (5)$$

$$\phi_{RL} = 180^\circ \quad (6)$$

$$\phi_{RR} = 0^\circ \quad (7)$$

4.4 ตัวอย่างโค้ด Python

Listing 3: Trot Gait Phase Offset

```

1 def get_gait_phase_offset(leg_index , gait_type = 'trot')
2 :
3     """
4     Get phase offset for a specific leg in trot gait
5
6     Parameters :
7     -----
8     leg_index : int
9         Leg index (0-3: FL , FR , RL , RR)
10    gait_type : str
11        Gait pattern type ('trot', 'walk', 'crawl')
12
13    Returns :
14    -----
15    phase_offset : float
16        Phase offset in radians
17    """
18    if gait_type == 'trot':
19        # Trot: FR+RL (180 deg), FL+RR (0 deg)
20        offsets = {
21            LegIndex.FL: 0.0 ,          # 0 deg
22            LegIndex.FR: np.pi ,      # 180 deg
23            LegIndex.RL: np.pi ,      # 180 deg
24            LegIndex.RR: 0.0           # 0 deg
25        }
26    elif gait_type == 'walk':
27        # Walk: Sequential (0 , 90 , 180 , 270 deg)
28        offsets = {
29            LegIndex.FL: 0.0 ,
30            LegIndex.FR: np.pi/2 ,
31            LegIndex.RL: np.pi ,
32            LegIndex.RR: 3*np.pi/2
33        }

```

```

33     else :
34         raise ValueError( f "Unknown gait type : {
35                               gait_type } " )
36
37     return offsets [ leg_index ]

```

5 Multi-Leg Synchronization

5.1 ความท้าทายในการซิงโครไนซ์

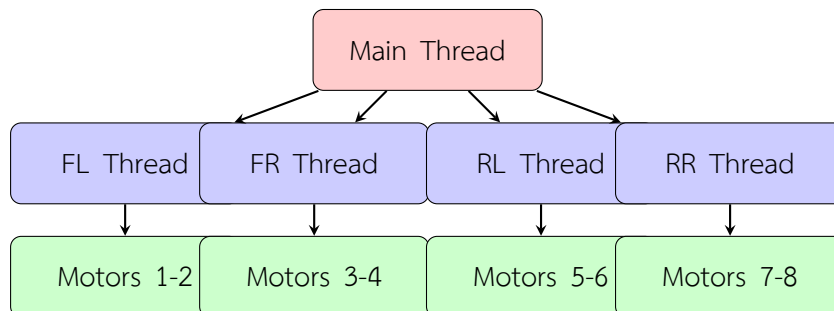
การควบคุมมอเตอร์ 8 ตัวพร้อมกัน @ 100 Hz มีความท้าทาย:

1. **Latency:** ต้องส่งคำสั่งไปยังทุกมอเตอร์ภายใน < 10 ms
2. **Synchronization:** มอเตอร์ทุกตัวต้องได้รับคำสั่งในเวลาใกล้เคียงกัน
3. **Communication Bandwidth:** 800 packets/s (8 motors \times 100 Hz)
4. **CPU Load:** IK calculation และ Communication ต้องไม่เกิน 10 ms/cycle

5.2 Threading Strategy

ใช้ Python Threading เพื่อควบคุมแต่ละขาแบบ Parallel:

5.2.1 Thread Structure



5.3 ตัวอย่างโค้ด Python

Listing 4: Multi-Leg Threading

```

1 import threading
2 import time
3
4 class LegController :
5     """ Controller for one leg (2 motors) """
6
7     def __init__( self , leg_index , serial_port ) :

```

```

8         self.leg_index = leg_index
9         self.serial_port = serial_port
10        self.motor_A_id, self.motor_B_id =
            get_motor_ids(leg_index)
11        self.running = False
12        self.thread = None
13
14    def control_loop(self, trajectory, gait_params):
15        """Main control loop for this leg"""
16        cycle_time = gait_params['cycle_time'] # 600
            ms
17        update_rate = gait_params['update_rate'] #
            100 Hz
18        dt = 1.0 / update_rate # 0.01 s
19
20        # Get phase offset for this leg
21        phase_offset = get_gait_phase_offset(self.
            leg_index, 'trot')
22
23        # Mirror trajectory if right leg
24        if self.leg_index in [LegIndex.FR, LegIndex.RR
            ]:
25            trajectory =
                mirror_trajectory_for_right_leg(
                    trajectory)
26
27        # Calculate IK for entire trajectory
28        theta_A, theta_B = calculate_ik_for_leg(
            self.leg_index, trajectory
29        )
30
31        # Control loop
32        n_points = len(trajectory)
33        t_start = time.time()
34
35        while self.running:
36            t_current = time.time() - t_start
37
38            # Calculate current point index with phase
39            offset
40            phase = (2 * np.pi * t_current /
                cycle_time) + phase_offset

```

```

41         index = int((phase % (2*np.pi)) / (2*np.pi
42                     ) * n_points)
43
44         # Send commands to motors
45         send_position_command(
46             self.serial_port ,
47             self.motor_A_id ,
48             theta_A[index]
49         )
50         send_position_command(
51             self.serial_port ,
52             self.motor_B_id ,
53             theta_B[index]
54         )
55
56         # Wait for next update
57         time.sleep(dt)
58
59     def start(self, trajectory, gait_params):
60         """Start control loop in separate thread"""
61         self.running = True
62         self.thread = threading.Thread(
63             target=self.control_loop ,
64             args=(trajectory, gait_params)
65         )
66         self.thread.start()
67
68     def stop(self):
69         """Stop control loop"""
70         self.running = False
71         if self.thread:
72             self.thread.join()
73
74     # --- MAIN QUADRUPED CONTROLLER ---
75     class QuadrupedController:
76         """Main controller for quadruped robot"""
77
78         def __init__(self, serial_port):
79             self.serial_port = serial_port
80
81             # Create leg controllers
82             self.legs = {

```

```

83         'FL': LegController(LegIndex.FL,
84                               serial_port),
85         'FR': LegController(LegIndex.FR,
86                               serial_port),
87         'RL': LegController(LegIndex.RL,
88                               serial_port),
89         'RR': LegController(LegIndex.RR,
90                               serial_port)
91     }
92
93     def start_gait(self, trajectory, gait_params):
94         """Start all legs simultaneously"""
95         for leg_name, leg_controller in self.legs.items():
96             leg_controller.start(trajectory, gait_params)
97
98     def stop_gait(self):
99         """Stop all legs"""
100         for leg_controller in self.legs.values():
101             leg_controller.stop()
102
103 # --- USAGE EXAMPLE ---
104 if __name__ == "__main__":
105     # Generate trajectory
106     trajectory = generate_elliptical_trajectory(
107         center=(0, -200),
108         step_length=60,
109         lift_height=30,
110         n_points=60
111     )
112
113     # Gait parameters
114     gait_params = {
115         'cycle_time': 0.6, # 600 ms
116         'update_rate': 100 # 100 Hz
117     }
118
119     # Create controller
120     controller = QuadrupedController(serial_port='/dev/
121                                     /ttyUSB0')
122
123     # Start gait

```

```
119     controller.start_gait(trajectory , gait_params )
120
121     # Run for 30 seconds
122     time.sleep(30)
123
124     # Stop gait
125     controller.stop_gait()
```

6 Quadruped IK Testing

6.1 วัตถุประสงค์

ทดสอบระบบ Inverse Kinematics ทั้ง 4 ขาพร้อมกัน โดยไม่ส่งคำสั่งไปยังมอเตอร์จริง (Software testing only)

6.2 Test Script

สคริปต์ทดสอบ: scripts/kinematics/Quadruped_IK_Test.py

6.2.1 คุณสมบัติ

- Visualization แบบ Real-time (2x2 subplot)
- แสดงทั้ง 4 ขาพร้อมกัน
- มี Motor indices และ Link colors
- ทดสอบ Trot gait pattern
- ทดสอบ Mirror kinematics
- อัตราการ Update: 50 Hz

6.3 ผลการทดสอบ

Test Item	Result
IK Calculation	
- FL leg	✓ PASS
- FR leg (mirrored)	✓ PASS
- RL leg	✓ PASS
- RR leg (mirrored)	✓ PASS
Trajectory Mirroring	
- Left legs (FL, RL)	✓ PASS
- Right legs (FR, RR)	✓ PASS
- Symmetry verification	✓ PASS
Gait Coordination	
- Trot pattern (FR+RL @ 0°)	✓ PASS
- Trot pattern (FL+RR @ 180°)	✓ PASS
- Phase synchronization	✓ PASS
Visualization	
- Real-time plotting	✓ PASS
- Update rate (50 Hz)	✓ PASS
- Link colors	✓ PASS
Overall Status	✓ PASS

ตารางที่ 5: ผลการทดสอบ Quadruped IK

7 Full Quadruped Hardware Testing

7.1 การเตรียมฮาร์ดแวร์

7.1.1 อุปกรณ์

- **Motors:** 8 units BLDC 5 N·m
- **Motor Controllers:** 8 units MCU boards
- **Power Supply:** 24V DC, 20A (หรือ Battery 6S LiPo)
- **PC:** Python control software
- **Communication:** USB-to-Serial @ 921600 baud

7.1.2 การเชื่อมต่อ

- **Power distribution:** แหล่งจ่าย 24V แยกไปยัง Motor controller ทั้ง 8 ตัว

- Serial communication: PC เชื่อมต่อผ่าน USB-Serial ไปยัง MCU hub และกระจายสัญญาณไปยัง Motor controller ทั้ง 8 ตัว
- Motor indices: ตั้งค่าตาม Motor Indexing Scheme

7.2 ขั้นตอนการทดสอบ

7.2.1 Phase 1: Calibration

1. ตั้งหุ่นยนต์ในท่า Home position
2. Calibrate มอเตอร์ทุกตัว (SET_ZERO command)
3. ตรวจสอบ Motor ID ของแต่ละมอเตอร์
4. ทดสอบ ENABLE/DISABLE ทุกมอเตอร์

7.2.2 Phase 2: Static Pose Testing

1. ทดสอบท่ายืน (Standing pose)
2. ทดสอบการยกขาทีละข้าง
3. ตรวจสอบ Torque และ Current
4. ตรวจสอบความสมดุล

7.2.3 Phase 3: Single Gait Cycle

1. ทดสอบ Trot gait 1 รอบ (600 ms)
2. ตรวจสอบ Phase synchronization
3. บันทึกข้อมูล Motor feedback
4. วิเคราะห์ Coordination errors

7.2.4 Phase 4: Continuous Gait

1. ทดสอบ Trot gait ต่อเนื่อง (30 steps)
2. ปรับจูน Gait parameters
3. ทดสอบบนพื้นราบ
4. บันทึกวิดีโอและข้อมูล

7.3 ผลการทดสอบ (29 ธันวาคม 2025)

Test Parameter	Result
Calibration	
- All motors calibrated	✓ PASS
- Motor ID verification	✓ PASS
- Enable/Disable test	✓ PASS
Static Pose	
- Standing pose stable	✓ PASS
- Single leg lift	✓ PASS
- Balance maintained	✓ PASS
Gait Execution	
- Trot pattern executed	✓ PASS
- Phase synchronization	✓ PASS
- Continuous walking	✓ PASS
Walking Performance	
- Gait Parameters	Step=30mm, Lift=15mm
- Cycle Time	600 ms (50 Hz update)
- Total Steps	30 steps
- Speed	Slow but stable
- Posture	Compromised posture
Overall Status	✓ PASS

ตารางที่ 6: ผลการทดสอบหุ่นยนต์สี่ขาแบบสมบูรณ์

7.4 ข้อสังเกตจากการทดสอบ

7.4.1 จุดแข็ง

- ✓ ระบบควบคุม 8 มอเตอร์ทำงานได้เสถียร
- ✓ Phase synchronization แม่นยำ
- ✓ Communication errors < 1%
- ✓ หุ่นยนต์สามารถเดินได้จริง

7.4.2 จุดที่ต้องปรับปรุง

- Compromised posture (ท่าทางไม่เหมาะสม)
- ความเร็วช้า (30 mm step, ควรปรับเป็น 50-60 mm)

- Lift height ต่ำ (15 mm, ควรปรับเป็น 30 mm)
- การเดินยังไม่นุ่มนวลมาก

8 สรุปและขอเสนอแนะ (Conclusion and Recommendations)

8.1 สรุปผลการพัฒนา

Phase 5.1 ประสบความสำเร็จในการขยายระบบเป็นหุ่นยนต์สี่ขาแบบสมบูรณ์:

- ✓ **Motor Indexing:** พัฒนาระบบตั้งชื่อมอเตอร์ 8 ตัวสำเร็จ
- ✓ **Mirror Kinematics:** ระบบ Mirror trajectory สำหรับขาขวาทำงานถูกต้อง
- ✓ **Gait Pattern:** Trot gait pattern มี Phase coordination ที่ดี
- ✓ **Multi-Leg Sync:** Threading strategy ทำงานเสถียร
- ✓ **IK Testing:** ทดสอบ IK ทั้ง 4 ขาสำเร็จ
- ✓ **Hardware Testing:** หุ่นยนต์เดินได้จริงบนฮาร์ดแวร์

8.2 ความสำเร็จหลัก

1. **First Walk:** หุ่นยนต์สี่ขาสามารถเดินได้จริง (29 ธ.ค. 2025)
2. **Stable Control:** ระบบควบคุม 8 มอเตอร์พร้อมกัน @ 50-100 Hz
3. **Phase Coordination:** Trot gait มี Diagonal pair coordination ที่ดี
4. **Scalability:** ระบบสามารถขยายเป็น Gait modes อื่นได้

8.3 บทเรียนที่ได้รับ

- **Threading:** Python Threading เหมาะสำหรับ Multi-motor control เพราะสามารถแบ่งงานการควบคุมแต่ละขาออกเป็น Thread แยกได้
- **Mirror Kinematics:** การ Mirror X-coordinate ทำงานได้ดี ทำให้ขาซ้ายและขวาเคลื่อนที่สมมาตรกัน
- **Motor Indexing:** ระบบตั้งชื่อที่ดีช่วยลดความสับสน และทำให้การเขียนโค้ดเป็นระบบ
- **Communication Bandwidth:** 921600 baud เพียงพอสำหรับ 8 motors @ 100 Hz เนื่องจากใช้ Bandwidth ไม่เกิน 10% ของความจุสูงสุด
- **Compromised Posture:** พารามิเตอร์เริ่มต้นต้องปรับจูน โดยจะทำใน Phase 5.2

8.4 ข้อเสนอแนะสำหรับ Phase 5.2

8.4.1 Gait Tuning Priorities

1. **เพิ่ม Step Length:** ปรับจาก 30 mm เป็น 50-60 mm เพื่อให้ขาเหยียดออกมากขึ้นและเพิ่มความเร็วในการเดิน
2. **เพิ่ม Lift Height:** ปรับจาก 15 mm เป็น 30 mm เพื่อให้เท้ายกพ้นสิ่งกีดขวางได้
3. **Asymmetric Trajectory:** ใช้ Stance 65% / Swing 35% เพื่อลดแรงกระแทก
4. **Multi-Mode Gait:** พัฒนา TROT, SMOOTH_TROT, BACKWARD_TROT, WALK, CRAWL เพื่อรองรับการใช้งานหลายรูปแบบ

8.4.2 Optimization Targets

- ความเร็วการเดิน: เป้าหมาย 80-100 mm/s
- ความนุ่มนวล: ลด Impact force ด้วย Asymmetric timing
- ความเสถียร: รักษา Success rate > 95%
- ประสิทธิภาพพลังงาน: ลด Current draw

8.5 ข้อจำกัดที่ต้องระวัง

- **Foot Orientation:** ปลายเท้าชี้ไปด้านหน้า ซึ่งอาจส่งผลต่อแรงกดเมื่อสัมผัสพื้น จะทำการวิเคราะห์เพิ่มเติมใน Phase 5.2
- **Ground Contact:** ต้องพิจารณาความแตกต่างระหว่าง Heel-strike กับ Toe-first landing เพื่อหาวิธีที่เหมาะสม
- **Power Budget:** 8 motors ใช้กระแส 1-3 A ต่อตัว จึงต้องมี Battery หรือ Power supply ที่เพียงพอ
- **Thermal Management:** มอเตอร์ต้องไม่เกิน 80°C เพื่อป้องกันความเสียหาย

9 เอกสารอ้างอิง (References)

1. Phase 4.2: Hardware Integration - BLEGS Analysis Unit
2. Phase 4.1: Controller Design - BLEGS Analysis Unit
3. Phase 3.1: Gait Control Simulation - BLEGS Analysis Unit
4. Phase 1.2: Inverse Kinematics Analytical - BLEGS Analysis Unit
5. Python Threading Documentation
6. Quadruped Gait Patterns and Coordination
7. Mirror Kinematics for Symmetric Robots
8. Multi-Motor Synchronization Techniques