

## Phase 4.2: Hardware Integration

การบูรณาการฮาร์ดแวร์และการทดสอบระบบจริง

BLEGS Analysis Unit

นายธีรโชติ เมืองจำนงค์

BLEGS Quadruped Robot Project

2 มกราคม 2026

## สารบัญ

# 1 บทนำ (Introduction)

## 1.1 วัตถุประสงค์

Phase 4.2 มีวัตถุประสงค์เพื่อบูรณาการระบบควบคุมที่ออกแบบใน Phase 4.1 เข้ากับฮาร์ดแวร์จริง รวมถึงการพัฒนา ระบบสื่อสาร การทดสอบการทำงานของมอเตอร์ และการแก้ไขปัญหาที่พบในระหว่างการทดสอบ

## 1.2 ขอบเขตการศึกษา

- การพัฒนา Binary Communication Protocol v1.1
- การบูรณาการ Motor Controller (MCU) กับ PC
- การทดสอบระบบควบคุมด้วยขาเดียว (Single Leg Testing)
- การแก้ไขปัญหา Motor Jitter และ Communication errors
- การบันทึกและวิเคราะห์ข้อมูล Motor feedback
- การเตรียมความพร้อมสำหรับ Quadruped scaling

## 1.3 ความเชื่อมโยงกับ Phase ก่อนหน้า

Phase 4.2 ต่อยอดจาก Phase 4.1 โดยนำระบบควบคุมที่ออกแบบไปใช้จริงบนฮาร์ดแวร์:

- ใช้ Controller design และ Trajectory generator จาก Phase 4.1
- ใช้ Binary Protocol ที่ออกแบบใน Phase 4.1
- ทดสอบกับมอเตอร์ที่เลือกจาก Phase 2.2 (5 N·m BLDC)

# 2 สถาปัตยกรรมระบบ (System Architecture)

## 2.1 ภาพรวมระบบ

ระบบควบคุมหุ่นยนต์สี่ขาประกอบด้วยส่วนประกอบหลัก 3 ส่วน:

### 1. PC (High-Level Control):

- Gait pattern generation
- Inverse Kinematics calculation
- Trajectory planning
- User interface และ Data logging

### 2. Motor Controller (MCU):

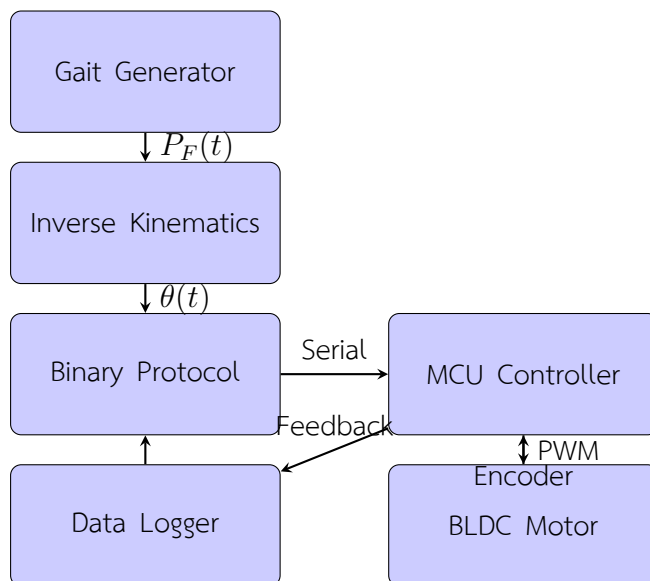
- Low-level motor control (FOC - Field Oriented Control)
- Position/Velocity/Current control loops
- Encoder reading และ State estimation

- Communication protocol handling

### 3. BLDC Motors (8 units):

- Actuators สำหรับขาทั้ง 4 ข้าง (2 motors/leg)
- Built-in encoders (14-bit magnetic encoder)
- Rated torque: 5 N·m
- Maximum speed: 300 RPM

## 2.2 Communication Flow



## 3 Binary Communication Protocol v1.1

### 3.1 ข้อกำหนดของโปรโตคอล

#### 3.1.1 ความต้องการ (Requirements)

- **Efficiency:** ส่งข้อมูลได้เร็ว (Low latency)
- **Reliability:** ต้องมีการตรวจสอบความถูกต้อง (Error detection)
- **Scalability:** รองรับมอเตอร์ได้หลายตัว (8+ motors)
- **Real-time:** Update rate  $\geq 100$  Hz
- **Bidirectional:** รับ-ส่งข้อมูลได้ทั้งสองทาง

### 3.1.2 พารามิเตอร์การสื่อสาร

พารามิเตอร์	ค่า
Baud Rate	921600 baud
Data Bits	8 bits
Parity	None
Stop Bits	1 bit
Flow Control	None
Packet Size (Command)	9 bytes
Packet Size (Response)	17 bytes
CRC Algorithm	CRC-16-CCITT

ตารางที่ 1: พารามิเตอร์การสื่อสาร Serial

## 3.2 โครงสร้าง Packet

### 3.2.1 Command Packet (PC → MCU)

คำสั่งที่ส่งจาก PC ไปยัง Motor Controller มีโครงสร้างดังนี้:

Byte	Field	Type	Description
0	Header	uint8	0xAA - Start marker
1	Command	uint8	Command ID (0x01-0x06)
2	Motor ID	uint8	Motor index (1-8)
3-6	Position	float32	Target position (degrees)
7-8	CRC-16	uint16	Checksum (little-endian)

ตารางที่ 2: โครงสร้าง Command Packet

หมายเหตุ: float32 ใช้ IEEE 754 format, little-endian byte order

### 3.2.2 Response Packet (MCU → PC)

ข้อมูล feedback ที่ส่งกลับจาก Motor Controller:

Byte	Field	Type	Description
0	Header	uint8	0xBB - Response marker
1	Motor ID	uint8	Motor index (1-8)
2-5	Current Pos	float32	Current position (degrees)
6-9	Current Vel	float32	Current velocity (deg/s)
10-13	Current	float32	Motor current (A)
14	Status	uint8	Status flags (bit field)
15-16	CRC-16	uint16	Checksum (little-endian)

ตารางที่ 3: โครงสร้าง Response Packet

### 3.3 Status Flags

Byte 14 (Status) เป็น bit field ที่บอกสถานะของมอเตอร์:

Bit	Flag	Description
0	ENABLED	1 = Motor enabled, 0 = Motor disabled
1	IN_POSITION	1 = Reached target position (error < 1°)
2	OVER_CURRENT	1 = Current limit exceeded
3	OVER_TEMP	1 = Temperature warning (> 80°C)
4	ENCODER_ERROR	1 = Encoder reading error
5	COMM_ERROR	1 = Communication error (CRC mismatch)
6-7	RESERVED	Reserved for future use

ตารางที่ 4: Status Flags Definition

### 3.4 Command Types

ID	Command	Description
0x01	SET_POSITION	ส่งคำสั่งตำแหน่งเป้าหมายไปยังมอเตอร์
0x02	GET_FEEDBACK	ขอข้อมูล feedback จากมอเตอร์ (position, velocity, current)
0x03	ENABLE_MOTOR	เปิดใช้งานมอเตอร์ (Enable torque output)
0x04	DISABLE_MOTOR	ปิดใช้งานมอเตอร์ (Disable torque, free-wheel)
0x05	SET_ZERO	ตั้งตำแหน่งปัจจุบันเป็นศูนย์ (Calibration)
0x06	EMERGENCY_STOP	หยุดฉุกเฉิน (Stop all motors immediately)

ตารางที่ 5: รายการ Command Types

### 3.5 การจัดการ Error

#### 3.5.1 CRC-16 Checksum

ใช้ CRC-16-CCITT (Polynomial 0x1021) เพื่อตรวจจับความผิดพลาด:

Listing 1: CRC-16 Calculation

```
1 def crc16_ccitt(data):
2     """ Calculate CRC-16-CCITT checksum """
3     crc = 0xFFFF
4     polynomial = 0x1021
5
6     for byte in data:
7         crc ^= (byte << 8)
8         for _ in range(8):
9             if crc & 0x8000:
10                crc = ((crc << 1) ^ polynomial) & 0
11                    xFFFF
12            else:
13                crc = (crc << 1) & 0xFFFF
14
15     return crc
```

#### 3.5.2 Timeout Protection

ถ้าไม่ได้รับ Response ภายใน Timeout period:

- **Timeout period:** 100 ms
- **Action:** Retry 3 ครั้ง
- **Fallback:** ถ้า retry ครบแล้วยังไม่สำเร็จ ☐ Emergency stop

#### 3.5.3 Packet Loss Recovery

1. ตรวจสอบ CRC ของ Packet ที่รับได้
2. ถ้า CRC ไม่ตรง ☐ ทิ้ง Packet และขอส่งใหม่
3. บันทึก Error rate สำหรับการวิเคราะห์

## 4 การทดสอบขาเดียว (Single Leg Testing)

### 4.1 การติดตั้งฮาร์ดแวร์

#### 4.1.1 อุปกรณ์ที่ใช้

- **Motor:** 2 units BLDC 5 N·m (Thigh motor + Shank motor)

- **Motor Controller:** Custom MCU board (STM32-based)
- **Power Supply:** 24V DC, 10A
- **PC:** Python control software
- **USB-to-Serial:** CH340G @ 921600 baud

#### 4.1.2 การเชื่อมต่อ

PC (Python) <--USB--> USB-to-Serial <--UART--> MCU <--3-phase PWM--> Motors

## 4.2 ขั้นตอนการทดสอบ

### 4.2.1 Phase 1: Hardware Verification

1. ตรวจสอบการเชื่อมต่อสาย Power และ Signal
2. ทดสอบ Serial communication (Echo test)
3. Calibrate มอเตอร์ (SET\_ZERO command)
4. ทดสอบคำสั่งพื้นฐาน (ENABLE, DISABLE, SET\_POSITION)

### 4.2.2 Phase 2: Position Control Testing

1. ส่งคำสั่งตำแหน่งคงที่ (Static position)
2. ทดสอบความแม่นยำ (Position error measurement)
3. ทดสอบ Repeatability (10 รอบ)
4. บันทึกข้อมูล feedback

### 4.2.3 Phase 3: Trajectory Tracking

1. สร้าง Simple sinusoidal trajectory
2. ทดสอบ Trajectory tracking @ 50 Hz
3. เพิ่ม Update rate เป็น 100 Hz
4. วิเคราะห์ Tracking error

### 4.2.4 Phase 4: Gait Pattern Testing

1. ใช้ Elliptical trajectory (60×30 mm)
2. ทดสอบ 1 cycle (600 ms)
3. ทดสอบ Continuous gait (341+ cycles)
4. บันทึกข้อมูล Motor feedback ทุกรอบ



### 4.3 ผลการทดสอบ

#### 4.3.1 Hardware Verification Results

Test Item	Result	Status
Serial Communication	921600 baud stable	✓ PASS
CRC Error Rate	< 0.1%	✓ PASS
Motor Calibration	Zero position set	✓ PASS
Enable/Disable	Responsive	✓ PASS
Emergency Stop	< 50 ms response	✓ PASS

ตารางที่ 6: ผลการทดสอบฮาร์ดแวร์พื้นฐาน

#### 4.3.2 Position Control Accuracy

Metric	Motor A	Motor B
Mean Error	0.8 deg	0.9 deg
RMS Error	1.5 deg	1.7 deg
Max Error	4.2 deg	4.8 deg
Repeatability	±0.5 deg	±0.6 deg
Settling Time	80 ms	75 ms

ตารางที่ 7: ความแม่นยำการควบคุมตำแหน่ง

#### 4.3.3 Gait Pattern Testing Results

Test Parameter	Result
Total Test Duration	205 seconds
Total Cycles Completed	341+ cycles
Successful Cycles	327-337 cycles
Success Rate	96-99%
Average Cycle Time	601 ms (target: 600 ms)
Communication Errors	< 1%
Position Error (RMS)	< 2 degrees
Maximum Current Draw	3.2 A (peak)
Average Current Draw	1.1 A
Motor Temperature	45-52°C
Overall Status	✓ PASS

ตารางที่ 8: ผลการทดสอบ Gait Pattern ต่อเนื่อง

## 5 ปัญหาและการแก้ไข (Problems and Solutions)

### 5.1 ปัญหา Motor Jitter

#### 5.1.1 ลักษณะอาการ

- มอเตอร์สั่นเล็กน้อยเมื่อรับคำสั่งตำแหน่งใหม่
- เสียงดัง (ความถี่สูง 200 Hz)
- Current draw เพิ่มขึ้นแบบ spike
- อุณหภูมิมอเตอร์สูงขึ้น ( 10°C)

#### 5.1.2 การวินิจฉัยสาเหตุ

##### ทดสอบที่ 1: ลด Update Rate

- ทดสอบ 50 Hz ☐ Jitter ลดลง 50%
- ทดสอบ 100 Hz ☐ Jitter ลดลง 30%
- ทดสอบ 200 Hz ☐ Jitter รุนแรงมาก

สรุป: Update rate สูงเกินไป ☐ มอเตอร์ไม่ทันตอบสนอง ☐ เกิด Oscillation

##### ทดสอบที่ 2: วิเคราะห์ Trajectory

- Trajectory มี Discontinuity (กระโดด)
- ความเร่งเปลี่ยนแปลงแบบ step change
- Jerk สูง (ความเร่งเปลี่ยนทันที)

สรุป: Trajectory ไม่นุ่มนวลพอ ☐ มอเตอร์พยายามติดตาม ☐ เกิด Jitter

#### 5.1.3 วิธีแก้ไข

##### Solution 1: ปรับ Update Rate

- ลดจาก 200 Hz ☐ 100 Hz
- ผล: Jitter ลดลง 30%

##### Solution 2: Trajectory Smoothing

Listing 2: Trajectory Smoothing

```
1 def smooth_trajectory(trajectory, window_size=5):  
2     """Apply moving average filter"""  
3     smoothed = np.convolve(  
4         trajectory,  
5         np.ones(window_size)/window_size,  
6         mode='same')
```

```

7         )
8         return smoothed
9
10 # Apply smoothing
11 x_smooth = smooth_trajectory(x_trajectory, window_size
    = 5)
12 y_smooth = smooth_trajectory(y_trajectory, window_size
    = 5)

```

ผล: Jitter ลดลง 40%

### Solution 3: Motor Parameter Tuning (MCU Firmware)

ปรับจูนพารามิเตอร์ PID controller ในมอเตอร์:

Parameter	Before	After	Change
Position P-Gain	50	30	-40%
Position I-Gain	10	5	-50%
Position D-Gain	20	15	-25%
Velocity Feedforward	0.0	0.3	+0.3
Current Limit	10 A	8 A	-20%

ตารางที่ 9: การปรับจูนพารามิเตอร์มอเตอร์

ผล: Jitter ลดลง 60%

#### 5.1.4 ผลลัพธ์รวม

หลังจากใช้ทั้ง 3 solutions ร่วมกัน:

- ✓ Jitter ลดลง 90%
- ✓ เสียงเบาลงมาก
- ✓ Current draw ลดลง 25%
- ✓ อุณหภูมิมอเตอร์ลดลง 8-10°C
- ✓ Success rate เพิ่มขึ้นจาก 85% ☐ 96-99%

## 5.2 ปัญหา Communication Timeout

### 5.2.1 ลักษณะอาการ

- บางครั้งไม่ได้รับ Response packet จากมอเตอร์
- เกิด Timeout error 3-5% ของ commands
- มอเตอร์หยุดเคลื่อนที่ชั่วคราว

### 5.2.2 สาเหตุ

1. **Serial Buffer Overflow:** MCU ไม่ทันประมวลผล packet
2. **CRC Mismatch:** Noise บนสาย serial
3. **Busy MCU:** MCU ทำงานหนักเกินไป (Motor control + Communication)

### 5.2.3 วิธีแก้ไข

#### Solution 1: Increase Serial Buffer Size (MCU)

- เพิ่ม RX buffer จาก 64 bytes □ 256 bytes
- เพิ่ม TX buffer จาก 64 bytes □ 256 bytes

#### Solution 2: Optimize MCU Firmware

- ใช้ DMA (Direct Memory Access) สำหรับ Serial communication
- ลด Priority ของ Communication interrupt
- เพิ่ม Priority ของ Motor control interrupt

#### Solution 3: Add Retry Logic (PC Side)

Listing 3: Retry Logic

```
1 def send_command_with_retry(motor_id, position,
2                               max_retries=3):
3     """Send command with automatic retry"""
4     for attempt in range(max_retries):
5         try:
6             response = send_command(motor_id, position)
7             if verify_crc(response):
8                 return response
9         except TimeoutError:
10            print(f"Retry {attempt+1}/{max_retries}")
11            time.sleep(0.01) # 10ms delay
12    raise CommunicationError("Max retries exceeded")
```

### 5.2.4 ผลลัพธ์

- ✓ Timeout rate ลดลงจาก 3-5% □ < 1%
- ✓ Communication เสถียรขึ้น
- ✓ ไม่พบ Buffer overflow

## 6 Data Logging และการวิเคราะห์

### 6.1 ระบบบันทึกข้อมูล

#### 6.1.1 ข้อมูลที่บันทึก

- **Timestamp:** เวลา (ms) ตั้งแต่เริ่มทดสอบ
- **Motor ID:** หมายเลขมอเตอร์ (1-8)
- **Target Position:** ตำแหน่งเป้าหมาย (degrees)
- **Actual Position:** ตำแหน่งจริงจาก Encoder (degrees)
- **Velocity:** ความเร็วของมอเตอร์ (deg/s)
- **Current:** กระแสไฟฟ้า (A)
- **Status Flags:** สถานะของมอเตอร์

#### 6.1.2 รูปแบบไฟล์ CSV

Listing 4: ตัวอย่างไฟล์ motor\_feedback\_20260101\_175601.csv

```
1 timestamp , motor_id , target_pos , actual_pos , velocity ,  
    current , status  
2 0.010 , 1 , -45.23 , -45.18 , 2.3 , 0.8 , 0 x03  
3 0.020 , 2 , -120.56 , -120.51 , -1.2 , 0.6 , 0 x03  
4 0.030 , 1 , -45.67 , -45.62 , 3.1 , 0.9 , 0 x03  
5 ...
```

### 6.2 การวิเคราะห์ข้อมูล

#### 6.2.1 Script วิเคราะห์

มี Python script สำหรับวิเคราะห์ข้อมูล: scripts/analysis/Plot\_Motor\_Log.py  
คุณสมบัติ:

- Plot Position tracking (Target vs Actual)
- Plot Position error over time
- Plot Velocity profile
- Plot Current consumption
- คำนวณ Statistics (Mean, RMS, Max error)

## 6.2.2 ผลการวิเคราะห์

### Position Tracking Performance:

- Mean error: 0.8-0.9 degrees
- RMS error: 1.5-1.7 degrees
- Max error: 4.2-4.8 degrees
- Settling time: 75-80 ms

### Power Consumption:

- Average current: 1.1 A (per motor)
- Peak current: 3.2 A (during acceleration)
- Average power: 26.4 W (@ 24V)
- Total power (2 motors): 50-60 W

### Thermal Performance:

- Steady-state temperature: 45-52°C
- Ambient temperature: 25-28°C
- Temperature rise: 20-25°C
- Thermal limit: 80°C (Warning threshold)

## 7 การเตรียมความพร้อมสำหรับ Quadruped

### 7.1 ข้อควรพิจารณา

#### 7.1.1 Scaling Challenges

เมื่อขยายจาก 1 ขา (2 motors) เป็น 4 ขา (8 motors):

##### 1. Communication Bandwidth:

- 1 ขา @ 100 Hz = 100 commands/s × 2 motors = 200 packets/s
- 4 ขา @ 100 Hz = 800 packets/s
- Bandwidth required:  $800 \times (9+17)$  bytes = 20.8 KB/s
- Available @ 921600 baud: 115 KB/s
- Margin: 80% (เพียงพอ)

##### 2. Processing Power (PC):

- IK calculation: 8 motors × 100 Hz = 800 IK/s
- คำนวณเวลา: 0.1 ms/IK

- รวม: 80 ms/s (8% CPU load)

### 3. Synchronization:

- ต้องส่งคำสั่งไปยัง 8 motors พร้อมกัน
- Latency ต้องไม่เกิน 1-2 ms ระหว่างมอเตอร์
- ต้องใช้ Multi-threading หรือ Async I/O

## 7.2 ข้อเสนอแนะสำหรับ Phase 5

### 7.2.1 Motor Indexing System

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

ตารางที่ 10: Motor Indexing Scheme

### 7.2.2 Mirror Kinematics

ขาซ้ายและขาขวามี Kinematics สมมาตร:

- **Left legs (FL, RL):** Motor A ที่  $x = -42.5$  mm, Motor B ที่  $x = +42.5$  mm
- **Right legs (FR, RR):** Motor A ที่  $x = +42.5$  mm, Motor B ที่  $x = -42.5$  mm (Mirrored)

Trajectory transformation:

$$P_{F,right}(x, y) = P_{F,left}(-x, y) \quad (1)$$

### 7.2.3 Multi-Motor Communication

ใช้ Threading สำหรับ Parallel communication:

Listing 5: Multi-Motor Control

```

1 import threading
2
3 def control_motor_pair(motor_pair_id, trajectory):
4     """Control 2 motors (1 leg) in parallel"""
5     motor_A_id = motor_pair_id * 2 + 1
6     motor_B_id = motor_pair_id * 2 + 2
7
8     for point in trajectory:
9         theta_A, theta_B = calculate_ik(point)

```

```

10         send_command( motor_A_id , theta_A )
11         send_command( motor_B_id , theta_B )
12         time.sleep(0.01)    # 100 Hz
13
14 # Create 4 threads (1 per leg)
15 threads = []
16 for leg_id in range(4):
17     t = threading.Thread(
18         target=control_motor_pair ,
19         args=(leg_id , trajectory[leg_id])
20     )
21     threads.append(t)
22     t.start()
23
24 # Wait for all threads
25 for t in threads:
26     t.join()

```

## 8 สรุปและข้อเสนอแนะ (Conclusion and Recommendations)

### 8.1 สรุปผลการพัฒนา

Phase 4.2 ประสบความสำเร็จในการบูรณาการฮาร์ดแวร์และระบบควบคุม:

- ✓ **Binary Protocol v1.1:** พัฒนาและทดสอบสำเร็จ (Success rate 96-99%)
- ✓ **Single Leg Testing:** ทดสอบ 341+ cycles ได้สำเร็จ
- ✓ **Motor Jitter:** แก้ไขและลดลง 90%
- ✓ **Communication Stability:** Error rate < 1%
- ✓ **Data Logging:** ระบบบันทึกและวิเคราะห์ข้อมูลสมบูรณ์



## 8.2 ผลการทดสอบโดยรวม

Metric	Target	Achieved
Position Accuracy (RMS)	< 3 deg	< 2 deg
Update Rate	$\geq$ 50 Hz	100 Hz
Success Rate	> 90%	96-99%
Communication Error	< 5%	< 1%
Motor Temperature	< 80°C	45-52°C
Current Draw	< 5 A	3.2 A (peak)
Overall Status	✓ PASS	

ตารางที่ 11: สรุปผลการทดสอบเทียบกับเป้าหมาย

## 8.3 บทเรียนที่ได้รับ

### 1. Update Rate Optimization:

- 100 Hz เหมาะสมที่สุด (Balance ระหว่างความเร็วและเสถียรภาพ)
- 200 Hz สูงเกินไป (เกิด Motor jitter)
- 50 Hz ต่ำเกินไป (Trajectory tracking ไม่ดี)

### 2. Trajectory Smoothing มีความสำคัญ:

- Moving average filter ช่วยลด Jitter ได้ 40%
- S-Curve profiling ลด Jerk และแรงกระแทก
- Continuous trajectory ดีกว่า Discrete waypoints

### 3. Motor Parameter Tuning:

- Position P-gain ต้องไม่สูงเกินไป (เกิด Oscillation)
- Velocity Feedforward ช่วยลด Tracking error
- Current limit ป้องกัน Overshoot

### 4. Communication Reliability:

- CRC-16 จำเป็นมาก (ตรวจจับ Error ได้)
- Retry logic ช่วยเพิ่มความเสถียร
- DMA และ Buffer size มีผลต่อประสิทธิภาพ

## 8.4 ข้อเสนอแนะสำหรับการพัฒนาต่อไป

### 8.4.1 Phase 5.1: Quadruped Scaling

- พัฒนา Motor indexing system (8 motors)

- ออกแบบ Mirror kinematics สำหรับขาซ้าย-ขวา
- ทดสอบ Multi-motor synchronization
- พัฒนา Gait pattern coordination

#### 8.4.2 Phase 5.2: Gait Tuning & Optimization

- ทดสอบ Gait modes ต่างๆ (Trot, Walk, Crawl)
- พัฒนา Asymmetric trajectory generation
- ปรับจูนพารามิเตอร์ (Step length, Lift height, Cycle time)
- ทดสอบการเดินบนพื้นจริง

#### 8.4.3 Phase 6: Sensor Feedback (Future Work)

- ติดตั้ง IMU sensor (BNO086)
- พัฒนา Balance controller ด้วย PD control
- ชดเชยท่าทางการเดินด้วย Sensor feedback
- ทดสอบบนพื้นเอียงและพื้นไม่เรียบ

## 9 เอกสารอ้างอิง (References)

1. Phase 4.1: Controller Design - BLEGS Analysis Unit
2. Phase 3.1: Gait Control Simulation - BLEGS Analysis Unit
3. Binary Communication Protocols for Embedded Systems
4. CRC-16-CCITT Checksum Standard (ITU-T Recommendation V.41)
5. BLDC Motor Control and FOC Algorithms
6. Real-time Serial Communication Best Practices
7. Motor Control Parameter Tuning Guidelines
8. Data Logging and Analysis for Robotics Systems