Phase 4.1: Controller Design

การออกแบบระบบควบคุมตำแหน่งสำหรับหุ่นยนต์สี่ขา

BLEGS Analysis Unit

นายธีรโชติ เมืองจำนงค์

BLEGS Quadruped Robot Project

2 มกราคม 2026

สารบัญ

1 บทนำ (Introduction)

1.1 วัตถุประสงค์

Phase 4.1 มีวัตถุประสงค์เพื่อออกแบบและพัฒนาระบบควบคุมตำแหน่งสำหรับมอเตอร์ของหุ่นยนต์สี่ขา โดยใช้วิธีการควบคุมแบบ Direct Position Control ร่วมกับ S-Curve Motion Profiling เพื่อให้การเคลื่อนที่ของหุ่นยนต์มีความนุ่มนวลและแม่นยำ

1.2 ขอบเขตการศึกษา

- การออกแบบระบบควบคุมตำแหน่งแบบ Direct Position Control
- การพัฒนา S-Curve Motion Profile Generator
- การบูรณาการระบบควบคุมกับ Inverse Kinematics
- การทดสอบระบบควบคุมกับขาเดียว (Single Leg Testing)
- การวิเคราะห์ประสิทธิภาพและความแม่นยำของระบบควบคุม

1.3 ความเชื่อมโยงกับ Phase ก่อนหน้า

Phase 4.1 ใช้ความรู้และผลลัพธ์จาก Phase ก่อนหน้าดังนี้:

- **Phase 1 (Kinematics):** ใช้ฟังก์ชัน IK analytical สำหรับแปลงตำแหน่งปลายเท้าเป็นมุมมอเตอร์
- **Phase 2 (Dynamics):** ใช้ข้อมูล Torque analysis เพื่อเลือกมอเตอร์ที่เหมาะสม (5 N·m)
- **Phase 3 (Simulation):** ใช้ Trajectory pattern จากการจำลองมาประยุกต์ในฮาร์ดแวร์จริง

2 หลักการควบคุมตำแหน่ง (Position Control Theory)

2.1 Direct Position Control

Direct Position Control คือวิธีการควบคุมที่ส่งคำสั่งตำแหน่งเป้าหมายไปยังมอเตอร์โดยตรง โดยมอเตอร์จะมีระบบควบคุมภายใน (Internal Controller) ที่รับผิดชอบในการเคลื่อนที่ไปยังตำแหน่งเป้าหมาย

2.1.1 ข้อดี

- **ความเรียบง่าย:** ไม่ต้องพัฒนา Controller Loop ที่ซับซ้อน
- **ความเสถียร:** ระบบควบคุมภายในของมอเตอร์ได้รับการปรับจูนแล้ว
- **ความแม่นยำ:** มอเตอร์ใช้ Encoder feedback เพื่อรักษาความแม่นยำ
- **การตอบสนอง:** Update rate สูง (100-1000 Hz) ภายในมอเตอร์

2.1.2 ข้อจำกัด

- ต้องอาศัยคุณภาพของ Internal Controller ของมอเตอร์
- ควบคุม Torque และ Velocity โดยตรงได้ยากกว่า
- ต้องระวังการส่งคำสั่งที่เร็วเกินไป (Command saturation)

2.2 ระดับของระบบควบคุม (Control Hierarchy)

ระบบควบคุมของหุ่นยนต์สี่ขาแบ่งเป็น 3 ระดับ:

1. **High-Level Controller (Gait Controller):**
   สร้าง Trajectory ของปลายเท้าในพื้นที่ Cartesian (x, y)
   - Input: Gait parameters (step length, lift height, cycle time)
   - Output: $\mathbf{P}_F(t) = [x_F(t), y_F(t)]^T$ สำหรับแต่ละขา
2. **Mid-Level Controller (Inverse Kinematics):**
   แปลงตำแหน่งปลายเท้าเป็นมุมมอเตอร์
   - Input: $\mathbf{P}_F(t)$ จาก Gait Controller
   - Process: Analytical IK (Phase 1.2)
   - Output: $\boldsymbol{\theta}(t) = [\theta_A(t), \theta_B(t)]^T$
3. **Low-Level Controller (Motor Controller):**
   ส่งคำสั่งตำแหน่งไปยังมอเตอร์
   - Input: $\boldsymbol{\theta}(t)$ จาก IK
   - Process: Communication protocol (Binary/Serial)
   - Output: Motor position commands

3 S-Curve Motion Profiling

3.1 หลักการ S-Curve Profile

S-Curve Motion Profile เป็นวิธีการสร้าง Trajectory ที่มีความเร่งเปลี่ยนแปลงอย่างต่อเนื่อง (Continuous Acceleration) เพื่อลดแรงกระแทก (Jerk) และทำให้การเคลื่อนที่นุ่มนวล

3.1.1 พารามิเตอร์

- p_0 = ตำแหน่งเริ่มต้น (Start position)
- p_f = ตำแหน่งเป้าหมาย (Final position)
- v_{max} = ความเร็วสูงสุด (Maximum velocity)
- a_{max} = ความเร่งสูงสุด (Maximum acceleration)
- j_{max} = Jerk สูงสุด (Maximum jerk)
- t_f = เวลารวม (Total time)

3.2 สมการ S-Curve Profile

S-Curve Profile แบ่งการเคลื่อนที่เป็น 7 ขั้นตอน แต่ในการประยุกต์ใช้จริง เรา simplify เป็น 3 ขั้นตอนหลัก:

3.2.1 ขั้นที่ 1: Acceleration Phase (0 ≤ t ≤ t□)

ความเร่งเพิ่มขึ้นแบบ smooth จาก 0 ถึง a_{max}

$$a(t) = a_{max} \sin\left(\frac{\pi t}{2t_1}\right) \tag{1}$$

$$v(t) = \frac{2a_{max}t_1}{\pi}\left[1 - \cos\left(\frac{\pi t}{2t_1}\right)\right] \tag{2}$$

$$p(t) = p_0 + \frac{2a_{max}t_1}{\pi}\left[t - \frac{2t_1}{\pi}\sin\left(\frac{\pi t}{2t_1}\right)\right] \tag{3}$$

3.2.2 ขั้นที่ 2: Constant Velocity Phase (t□ ≤ t ≤ t□)

เคลื่อนที่ด้วยความเร็วคงที่

$$a(t) = 0 \tag{4}$$

$$v(t) = v_{cruise} \tag{5}$$

$$p(t) = p_1 + v_{cruise}(t - t_1) \tag{6}$$

3.2.3 ขั้นที่ 3: Deceleration Phase (t□ ≤ t ≤ t_f)

ความเร่งลดลงแบบ smooth จาก $-a_{max}$ ถึง 0

$$a(t) = -a_{max} \sin\left(\frac{\pi(t - t_2)}{2t_3}\right) \tag{7}$$

$$v(t) = v_{cruise} - \frac{2a_{max}t_3}{\pi}\left[1 - \cos\left(\frac{\pi(t - t_2)}{2t_3}\right)\right] \tag{8}$$

$$p(t) = p_2 + v_{cruise}(t - t_2) - \frac{2a_{max}t_3}{\pi}\left[(t - t_2) - \frac{2t_3}{\pi}\sin\left(\frac{\pi(t - t_2)}{2t_3}\right)\right] \tag{9}$$

3.3 การประยุกต์ใช้ในระบบจริง

ในการพัฒนาระบบควบคุมหุ่นยนต์ เราใช้วิธี S-Curve แบบง่าย โดยใช้ **Sinusoidal Velocity Profile**:

$$v(t) = v_{max} \sin\left(\frac{\pi t}{t_f}\right) \tag{10}$$

ตำแหน่งจะได้จากการ integrate:

$$p(t) = p_0 + (p_f - p_0)\left[\frac{t}{t_f} - \frac{1}{2\pi}\sin\left(\frac{2\pi t}{t_f}\right)\right] \tag{11}$$

ข้อดี:

- สมการเรียบง่าย คำนวณเร็ว
- Jerk ต่ำ (ความเร่งเปลี่ยนแปลงอย่างต่อเนื่อง)
- เหมาะสำหรับ Gait trajectory ที่เป็นวงรอบ (Cyclic motion)

4 การออกแบบ Trajectory Generator

4.1 Elliptical Foot Trajectory

สำหรับการเดินของหุ่นยนต์สี่ขา เราใช้ Trajectory รูปวงรีสำหรับแต่ละขา:

4.1.1 พารามิเตอร์

- x_c, y_c = จุดศูนย์กลางของวงรี (Center position)
- a = กึ่งแกนใหญ่ (Semi-major axis) = Step length / 2
- b = กึ่งแกนเล็ก (Semi-minor axis) = Lift height / 2
- T = Gait cycle time
- ϕ = Phase offset สำหรับแต่ละขา

4.1.2 สมการ Trajectory

$$x_F(t) = x_c + a \cos\left(\frac{2\pi t}{T} + \phi\right) \tag{12}$$

$$y_F(t) = y_c - b \left|\sin\left(\frac{2\pi t}{T} + \phi\right)\right| \tag{13}$$

โดยใช้ $|\sin|$ เพื่อให้วงรีอยู่ด้านล่างเท่านั้น (ไม่ยกขาเหนือจุดศูนย์กลาง)

4.2 Asymmetric Trajectory (Advanced)

สำหรับการเดินที่นุ่มนวลยิ่งขึ้น เราพัฒนา Asymmetric Trajectory ที่แบ่งเวลาไม่เท่ากัน:

- **Stance Phase (65%):** เท้าสัมผัสพื้น - ใช้เวลานาน เพื่อลดแรงกระแทก
- **Swing Phase (35%):** เท้ายกขึ้น - ใช้เวลาสั้น เพื่อเคลื่อนที่เร็ว

4.2.1 สมการ Time Warping

ใช้ฟังก์ชัน $\tau(t)$ เพื่อแปลงเวลาเชิงเส้นเป็น Asymmetric time:

$$\tau(t) = \begin{cases} \frac{t}{0.65T} \cdot \pi & \text{if } 0 \leq t < 0.65T \text{ (Stance)} \\ \pi + \frac{t-0.65T}{0.35T} \cdot \pi & \text{if } 0.65T \leq t < T \text{ (Swing)} \end{cases} \tag{14}$$

แล้วใช้ในสมการ:

$$x_F(t) = x_c + a\cos(\tau(t) + \phi) \tag{15}$$
$$y_F(t) = y_c - b|\sin(\tau(t) + \phi)| \tag{16}$$

5 การบูรณาการกับ Inverse Kinematics

5.1 ขั้นตอนการแปลง Trajectory

1. **Trajectory Generation:**
   สร้าง $\mathbf{P}_F(t) = [x_F(t), y_F(t)]^T$ จาก Gait parameters

2. **Inverse Kinematics:**
   แปลงเป็นมุมมอเตอร์โดยใช้ฟังก์ชัน IK analytical:

$$[\theta_A(t), \theta_B(t)] = \text{IK\_analytical}(\mathbf{P}_F(t), \text{config}) \tag{17}$$

3. **Position Command:**
   ส่งคำสั่งไปยังมอเตอร์:

$$\text{Motor}_A \leftarrow \theta_A(t), \quad \text{Motor}_B \leftarrow \theta_B(t) \tag{18}$$

5.2 การเลือก Configuration

จาก Phase 1.2 เรามี 4 Configurations ที่เป็นไปได้ แต่เราเลือกใช้:

- **Configuration 1 (Down-Down):** เหมาะสำหรับการเดิน
  - Torque สมดุล (จาก Phase 2.1)
  - มุมมอเตอร์อยู่ในช่วงที่ปลอดภัย
  - Workspace กว้าง

5.3 ตัวอย่างโค้ด Python

Listing 1: Trajectory to Motor Angles

```
import numpy as np
from kinematics import calculate_ik_analytical

def trajectory_to_motor_angles(x_traj, y_traj, config
    =1):
```

```
    """
    Convert Cartesian trajectory to motor angles

    Parameters:
    -----------
    x_traj : array-like
        X coordinates of foot trajectory (mm)
    y_traj : array-like
        Y coordinates of foot trajectory (mm)
    config : int
        IK configuration (1, 2, 3, or 4)

    Returns:
    --------
    theta_A : array
        Motor A angles (degrees)
    theta_B : array
        Motor B angles (degrees)
    """
    n_points = len(x_traj)
    theta_A = np.zeros(n_points)
    theta_B = np.zeros(n_points)

    for i in range(n_points):
        P_F = np.array([x_traj[i], y_traj[i]])
        theta_A[i], theta_B[i] =
            calculate_ik_analytical(P_F, config)

    return theta_A, theta_B

# Example: Generate elliptical trajectory
t = np.linspace(0, 1, 100)  # 100 points per cycle
x_center, y_center = 0, -200
step_length, lift_height = 60, 30

x_traj = x_center + (step_length/2) * np.cos(2*np.pi*t)
y_traj = y_center - (lift_height/2) * np.abs(np.sin(2*np.pi*t))

# Convert to motor angles
theta_A, theta_B = trajectory_to_motor_angles(x_traj, y_traj, config=1)
```

6 ข้อกำหนดทางเทคนิค (Technical Specifications)

6.1 พารามิเตอร์การควบคุม

พารามิเตอร์	ค่า	หน่วย
Update Rate	50-100	Hz
Control Loop Time	10-20	ms
Communication Baud Rate	921600	baud
Position Resolution	0.01	degrees
Maximum Velocity	300	deg/s
Maximum Acceleration	1000	deg/s²

ตารางที่ 1: พารามิเตอร์การควบคุมระบบ

6.2 Gait Parameters (Single Leg Testing)

พารามิเตอร์	ค่า	หน่วย
Trajectory Parameters		
Center Position	$(0, -200)$	mm
Step Length	60	mm
Lift Height	30	mm
Timing Parameters		
Gait Cycle Time	600	ms
Update Rate	100	Hz
Points per Cycle	60	points
Test Parameters		
Total Cycles	341+	cycles
Total Test Time	205	seconds
Success Rate	96-99	%

ตารางที่ 2: พารามิเตอร์การทดสอบขาเดียว

7 การทดสอบและผลลัพธ์ (Testing and Results)

7.1 Single Leg Testing

การทดสอบขาเดียวดำเนินการกับขาซ้ายหน้า (Front-Left) โดยใช้ Binary Protocol v1.1

7.1.1 ขั้นตอนการทดสอบ

1. เริ่มต้นที่ Home position: $P_F = (0, -200)$ mm
2. สร้าง Elliptical trajectory (60×30 mm)
3. แปลงเป็นมุมมอเตอร์ด้วย IK (Config 1)
4. ส่งคำสั่งไปยังมอเตอร์ @ 100 Hz
5. บันทึกข้อมูล feedback จากมอเตอร์
6. วิเคราะห์ความแม่นยำและ error

7.1.2 ผลการทดสอบ

เกณฑ์	**ผลลัพธ์**
Total Cycles Tested	341+ cycles
Success Rate	96-99%
Position Error (RMS)	< 2 degrees
Maximum Error	< 5 degrees
Communication Errors	$< 1\%$
Motor Response Time	< 10 ms
สถานะ	**✓PASS**

ตารางที่ 3: ผลการทดสอบการควบคุมขาเดียว

7.2 การแก้ปัญหา Motor Jitter

7.2.1 ปัญหาที่พบ

ในช่วงแรกของการทดสอบ พบว่ามอเตอร์มีอาการสั่น (Jitter) เมื่อรับคำสั่งตำแหน่งใหม่

7.2.2 สาเหตุ

- ส่งคำสั่งเร็วเกินไป (Update rate สูงเกินจำเป็น)
- คำสั่งตำแหน่งมีการกระโดดแบบ step change
- ระบบควบคุมภายในมอเตอร์ตอบสนองแบบ aggressive

7.2.3 วิธีแก้ไข

1. ลด Update rate จาก 200 Hz เหลือ 100 Hz
2. เพิ่ม S-Curve profiling ให้กับ trajectory
3. ปรับจูนพารามิเตอร์ของมอเตอร์ (ใน MCU firmware):

- ลด Position P-gain
- เพิ่ม Velocity feedforward
- ปรับ Current limit

4. ใช้ Trajectory smoothing filter (Moving average)

7.2.4 ผลลัพธ์หลังแก้ไข

- Motor jitter ลดลงเกือบหมด
- การเคลื่อนที่นุ่มนวลและต่อเนื่อง
- Success rate เพิ่มขึ้นจาก 85% เป็น 96-99%

8 การพัฒนา Binary Protocol

8.1 ภาพรวม Communication Protocol

Binary Protocol ถูกพัฒนาเพื่อการสื่อสารที่มีประสิทธิภาพระหว่าง PC และ Motor Controller (MCU)

8.1.1 คุณสมบัติ

- **Efficiency:** ใช้ Binary format (compact กว่า ASCII)
- **Reliability:** มี CRC-16 checksum
- **Speed:** Baud rate 921600 (ส่งข้อมูลได้เร็ว)
- **Flexibility:** รองรับคำสั่งหลายประเภท

8.2 โครงสร้าง Packet

8.2.1 Command Packet (PC □ MCU)

Field	Size (bytes)	Type	Description
Header	1	uint8	0xAA (Start marker)
Command	1	uint8	Command ID
Motor ID	1	uint8	Motor index (1-8)
Position	4	float32	Target position (degrees)
CRC-16	2	uint16	Checksum
Total	**9**		

ตารางที่ 4: โครงสร้าง Command Packet

8.2.2 Response Packet (MCU → PC)

Field	Size (bytes)	Type	Description
Header	1	uint8	0xBB (Start marker)
Motor ID	1	uint8	Motor index (1-8)
Current Pos	4	float32	Current position (degrees)
Current Vel	4	float32	Current velocity (deg/s)
Current	4	float32	Motor current (A)
Status	1	uint8	Status flags
CRC-16	2	uint16	Checksum
Total	**17**		

ตารางที่ 5: โครงสร้าง Response Packet

8.3 Command Types

ID	Command	Description
0x01	SET_POSITION	ส่งคำสั่งตำแหน่งเป้าหมาย
0x02	GET_FEEDBACK	ขอข้อมูล feedback จากมอเตอร์
0x03	ENABLE_MOTOR	เปิดใช้งานมอเตอร์
0x04	DISABLE_MOTOR	ปิดใช้งานมอเตอร์
0x05	SET_ZERO	ตั้งตำแหน่งปัจจุบันเป็นศูนย์
0x06	EMERGENCY_STOP	หยุดฉุกเฉิน

ตารางที่ 6: รายการ Command Types

8.4 CRC-16 Checksum

ใช้ CRC-16-CCITT polynomial สำหรับตรวจจับ error:

$$\text{Polynomial} = x^{16} + x^{12} + x^5 + 1 = 0\text{x}1021 \tag{19}$$

8.4.1 ตัวอย่างโค้ด Python

Listing 2: CRC-16 Implementation

```python
def crc16_ccitt(data):
    """
    Calculate CRC-16-CCITT checksum

    Parameters:
```

```
    -----------
    data : bytes
        Input data

    Returns:
    --------
    crc : int
        CRC-16 checksum (16-bit)
    """
    crc = 0xFFFF
    polynomial = 0x1021

    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
            if crc & 0x8000:
                crc = ((crc << 1) ^ polynomial) & 0
                    xFFFF
            else:
                crc = (crc << 1) & 0xFFFF

    return crc
```

9 สรุปและข้อเสนอแนะ (Conclusion and Recommendations)

9.1 สรุปผลการออกแบบ

Phase 4.1 ได้ออกแบบและพัฒนาระบบควบคุมตำแหน่งสำหรับหุ่นยนต์สี่ขาได้สำเร็จ โดยมีผลลัพธ์ดังนี้:

- ✓ **Controller Design:** Direct Position Control + S-Curve Profiling
- ✓ **Trajectory Generation:** Elliptical และ Asymmetric trajectory
- ✓ **IK Integration:** ใช้ Analytical IK (Config 1) สำเร็จ
- ✓ **Binary Protocol:** รองรับการสื่อสารที่เสถียร (Success rate 96-99%)
- ✓ **Single Leg Testing:** ทดสอบ 341+ cycles สำเร็จ

9.2 จุดเด่นของระบบ

1. **ความนุ่มนวล:** S-Curve profiling ลด Jerk และแรงกระแทก
2. **ความแม่นยำ:** Position error < 2 degrees (RMS)
3. **ความเสถียร:** Success rate สูง (96-99%)

4. **ความเร็ว:** Update rate 100 Hz เพียงพอสำหรับ Gait control
5. **ความยืดหยุ่น:** รองรับ Gait pattern หลายแบบ

9.3 บทเรียนที่ได้รับ

- **Update Rate:** ไม่จำเป็นต้องสูงเกินจำเป็น - 100 Hz เพียงพอ
- **Trajectory Smoothing:** สำคัญมากสำหรับการลด Motor jitter
- **Motor Tuning:** การปรับจูนพารามิเตอร์มอเตอร์ (P-gain, Feedforward) มีผลมาก
- **Error Handling:** ต้องมี Checksum และ Timeout protection

9.4 ข้อเสนอแนะสำหรับ Phase ถัดไป

9.4.1 Phase 4.2: Hardware Integration

- ขยายระบบควบคุมจาก 1 ขา (2 motors) เป็น 4 ขา (8 motors)
- พัฒนา Multi-motor synchronization
- ทดสอบ Trot gait บนฮาร์ดแวร์จริง
- เพิ่ม Emergency stop และ Safety features

9.4.2 Phase 5: Quadruped Scaling

- พัฒนา Motor indexing system
- ออกแบบ Mirror kinematics สำหรับขาซ้าย-ขวา
- ทดสอบ Multi-leg coordination
- ปรับจูน Gait parameters สำหรับการเดินบนพื้นจริง

9.5 ข้อจำกัดที่ควรระวัง

- **IK Singularities:** หลีกเลี่ยงตำแหน่งที่ IK มี multiple solutions หรือ undefined
- **Workspace Limits:** ตรวจสอบว่า Trajectory อยู่ภายใน reachable workspace
- **Communication Latency:** ต้องมีการชดเชย Latency ในระบบ Real-time
- **Power Management:** ต้องมี Battery monitoring และ Low voltage protection

10 เอกสารอ้างอิง (References)

1. Phase 1.2: Inverse Kinematics Analytical - BLEGS Analysis Unit
2. Phase 2.1: Static Torque Analysis - BLEGS Analysis Unit
3. Phase 3.1: Gait Control Simulation - BLEGS Analysis Unit
4. S-Curve Motion Profile Theory and Applications
5. Binary Communication Protocols for Embedded Systems
6. CRC-16-CCITT Checksum Standard