



SIMULATION TRIPOD ROBOT FROM THE MOVIE WAR OF THE WORLDS: MOVEMENT STAND-UP

จัดทำโดย

นายคณพล	กาจธัญกิจ	รหัสนักศึกษา	65340500005
นายธนศพล	หีบแก้ว	รหัสนักศึกษา	65340500027

สถาบันวิทยาการหุ่นยนต์ภาคสนาม

สาขาวิศวกรรมหุ่นยนต์และระบบอัตโนมัติ

ที่ปรึกษา

นายธนวัฒน์ พาวันทา

รายงานนี้เป็นส่วนหนึ่งของรายวิชา FRA333 Kinematics of robotics system

สถาบันวิทยาการหุ่นยนต์ภาคสนาม มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

ภาคเรียนที่ 1 ปีการศึกษา 2567

บทคัดย่อ

โครงงานนี้เป็นโปรเจกสำหรับจัดทำ Simulation หุ่นยนต์ Tripod จากภาพยนตร์ War of the worlds ให้สามารถยืนขึ้นในแนวดิ่งและเงยหน้าได้ โดยทราบความยาวของขา (Fixed Link) เพื่อคำนวณหาตำแหน่งและองศาของหัว (end effector) จาก Forward Kinematic และใช้ Inverse Kinematic เพื่อคำนวณหา Configuration Space ที่ทำให้หุ่นยนต์ Tripod ยืนขึ้นในแนวดิ่งและเงยหน้าตามที่ต้องการ โดยใช้ DH-parameter และมีการแสดงผลเป็นภาพเพื่อให้สามารถง่ายต่อการสังเกต

คำสำคัญ: DH-parameter, Tripod, War of the worlds

สารบัญ

เรื่อง	หน้า
บทคัดย่อ.....	ก
สารบัญ	ข
บทที่ 1 บทนำ.....	1
บทที่ 2 ทบทวนวรรณกรรม และทฤษฎีที่เกี่ยวข้อง	2
บทที่ 3 วิธีการดำเนินงาน.....	8
บทที่ 4 ผลการดำเนินงาน	24
บทที่ 5 สรุป อภิปรายผล และข้อเสนอแนะ	29
เอกสารอ้างอิง	31
ภาคผนวก	32

บทที่ 1 บทนำ

1.1 ที่มาและความสำคัญ

คณะผู้จัดทำมีความสนใจหุ่นยนต์ Tripod จากหนังเรื่อง War of the world ที่มีความสามารถในการเคลื่อนไหวอย่างมั่นคงแม้จะมีโครงสร้างขาเพียงสามข้าง การศึกษานี้มีความสำคัญในการเข้าใจการควบคุมและการคำนวณตำแหน่งของหัวใน Task Space ของหุ่นยนต์แบบ Tripod ผ่านการใช้ Forward Kinematic และ Inverse Kinematic ซึ่งเป็นพื้นฐานในการพัฒนาหุ่นยนต์ในสถานะที่ต้องการความมั่นคง

1.2 วัตถุประสงค์

- 1) เพื่อศึกษาขาของหุ่นยนต์ Tripod แบบ revolute จากทำยีนในปริภูมิ 2 มิติ
- 2) เพื่อศึกษาการคำนวณ Forward Kinematic และ Inverse Kinematic เพื่อตรวจสอบตำแหน่งและองศาของหัว หุ่นยนต์ใน Task Space
- 3) เพื่อพัฒนาระบบแสดงผลการเคลื่อนไหวของหุ่นยนต์ในเชิงภาพเพื่อให้เข้าใจการทำงานของแต่ละขา ระบบได้ง่ายขึ้น

1.3 ขอบเขต

- 1) คำนวณเฉพาะ Kinematic เท่านั้น
- 2) กำหนดให้หุ่นยนต์มี 3 ขา และแต่ละขาของหุ่นยนต์ มี 2 Joint เท่านั้น
- 3) กำหนดให้ตำแหน่งปลายขาของหุ่นยนต์ ไม่มีการเปลี่ยนแปลง แต่สามารถเปลี่ยนการวางแนว (orientation)
- 4) กำหนดให้ในการคำนวณทุก Link ของขาเป็น Rigid body ที่มีเพียงความยาวของ link หน่วยเมตรเท่านั้น
- 5) กำหนดให้ท่าเริ่มต้น อยู่ในสถานะสมดุล (Task Space เริ่มต้น)
- 6) คำนวณการยืงขึ้นในแนวดิ่งและการเงยหน้า โดยพิจารณาเฉพาะตำแหน่งและความเร็ว
- 7) ค่าจาก Input เป็นจำนวนเต็มเท่านั้น

1.4 นิยามศัพท์เฉพาะ

War of the worlds = นิยายวิทยาศาสตร์เกี่ยวกับการรุกรานโลกจากต่างดาว

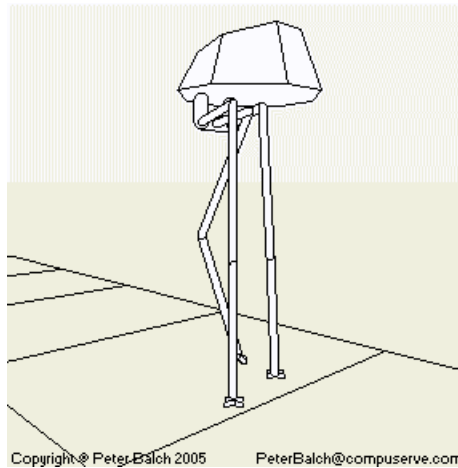
Tripod = หุ่นยนต์ที่มีลักษณะ 3 ขามาจากภาพยนตร์เรื่อง War of the worlds

บทที่ 2 ทบทวนวรรณกรรม และทฤษฎีที่เกี่ยวข้อง

2.1 WAR OF THE WORLDS - TRIPOD GAIT (Peter Balch., 2005) [1]

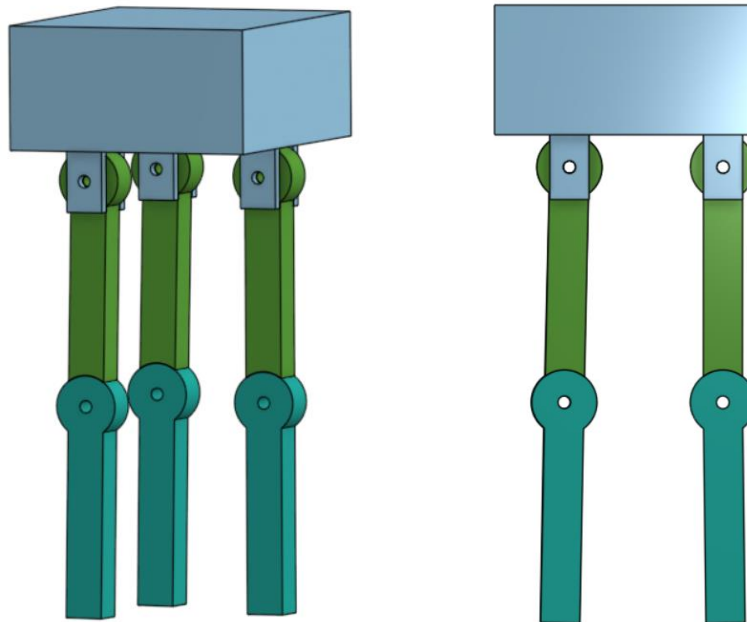
Configuration tripod robot

จากคำบรรยายลักษณะของหุ่นยนต์ในนิยาย คุณ Peter Balch ได้นำมาออกแบบเป็นหุ่นยนต์ได้ ดังนี้

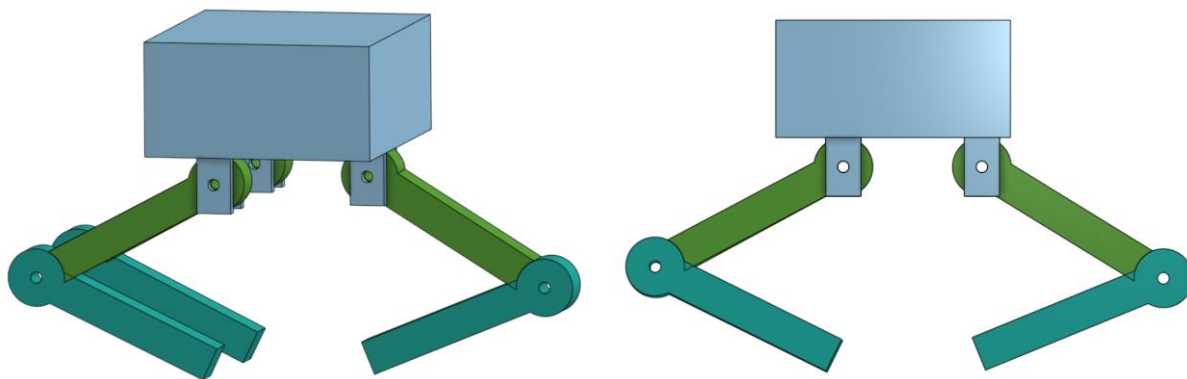


รูปที่ 1 Tripod robot โดยคุณ Peter Balch

เพื่อให้สะดวกต่อการคำนวณ Kinematic และเห็นองค์ประกอบของหุ่นยนต์ เช่น จำนวนและตำแหน่งของ Joint คณะผู้จัดทำจึงได้ออกแบบเพิ่มเติมให้มีลักษณะ ดังนี้



รูปที่ 2 Tripod robot แบบยี่น โดยคณะผู้จัดทำ



รูปที่ 3 Tripod robot แบบนั่ง โดยคณะผู้จัดทำ

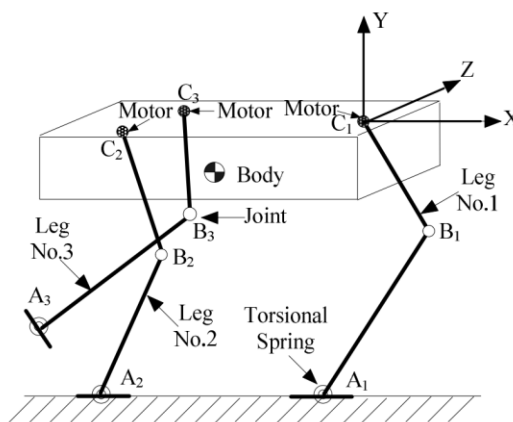
จากการออกแบบเพิ่มเติมทำให้ทราบว่า หุ่นยนต์นี้ มี 3 ขา ในแต่ละขาจะมี 2 Joint และ 2 Link โดย Joint ทั้งหมดเป็นแบบ Revolute Joint

2.2 A NOVEL BIOLOGICALLY INSPIRED TRIPOD WALKING ROBOT (Marco Ceccarelli, Conghui Liang, Giuseppe Carbone., 2009) [3]

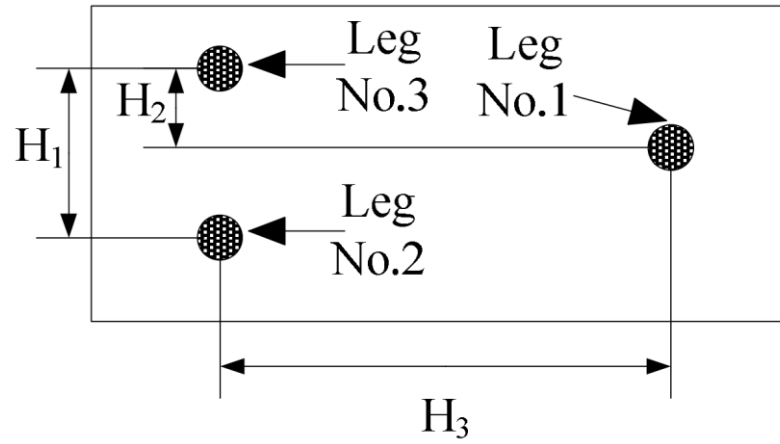
ในวิทยานิพนธ์นี้ นักวิจัยได้พัฒนาหุ่นยนต์เดินสามขา โดยใช้ลักษณะการเดินจากสิ่งมีชีวิต เช่น คนที่เดินด้วยไม้เท้าค้ำยัน, การใช้หางของจิงโจ้ในการเคลื่อนที่ มาเป็นต้นแบบของหุ่นยนต์

Mechanism

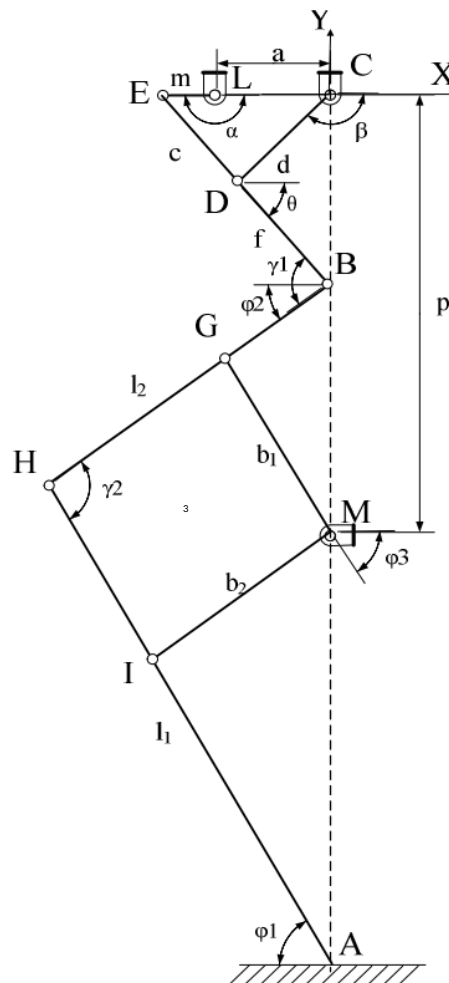
หุ่นยนต์นี้จะมี 3 ขา ในแต่ละขาจะมี 2 Joint และ 2 Link Joint ทั้งหมดเป็นแบบ Revolute Joint และมีข้อต่อแบบอิสระเชื่อมโยงแต่ละขา ทำให้แต่ละขามีความสัมพันธ์กันในเวลาเดิน อีกทั้งบริเวณเท้าจะมีสปริง เพื่อทำให้เท้าหุ่นยนต์ระนาบไปกับพื้นเวลาวางเท้า



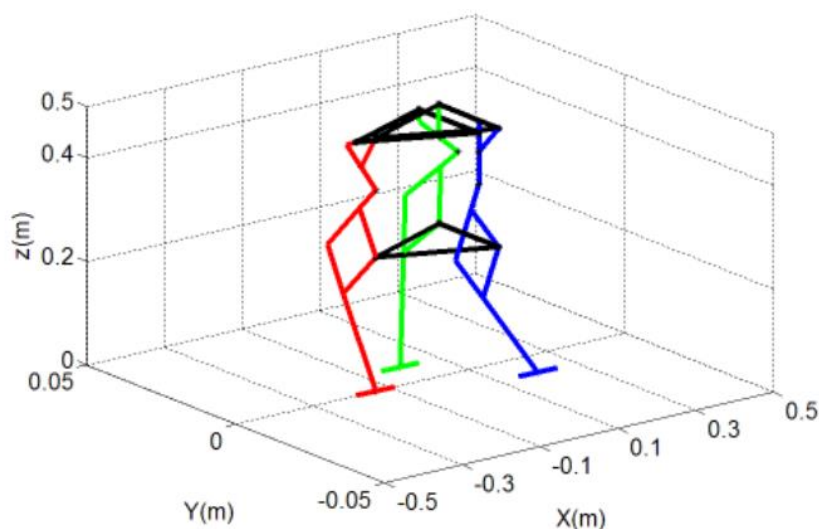
รูปที่ 4 three legs configuration



รูปที่ 5 contacts of three legs in horizontal plane



รูปที่ 6 A scheme for the proposed leg mechanism with design parameters for a tripod walking robot



รูปที่ 7 Simulation in the MATLAB environment

Kinematic Analysis

ในวิทยานิพนธ์นี้ใช้กฎมือซ้าย ในการคำนวณโดยเริ่มต้นคำนวณหาตำแหน่งของ B ที่เป็นข้อต่อที่ 2 ของขา
ได้เป็นสมการ

$$X_B = -a + m \cos \alpha + (c + f) \cos \theta$$

$$Y_B = -m \sin \alpha - (c + f) \sin \theta \quad (1)$$

- โดย X_B คือ ตำแหน่งของจุด B ในแกน x (หน่วย เมตร)
 Y_B คือ ตำแหน่งของจุด B ในแกน y (หน่วย เมตร)
 a คือ ระยะจากข้อต่อแบบอสิระ L ไปข้อต่อที่คุมได้ C (หน่วย เมตร)
 m คือ ระยะจากจุด E ที่เป็นตัวเชื่อมโยงกับขาอื่น ไปข้อต่ออสิระ L (หน่วย เมตร)
 α คือ มุมของข้อต่อแบบอสิระ L เทียบกับแกน x (หน่วย องศา)
 c คือ ระยะจากจุด E ไปจุด D (หน่วย เมตร)
 f คือ ระยะจากจุด D ไปจุด B (หน่วย เมตร)
 θ คือ มุมของจุด D เทียบกับแกน x (หน่วย องศา)

โดย θ หาได้จากสมการ

$$\theta = 2 \tan^{-1} \left(\frac{-B + (B^2 - 4AC)^{1/2}}{2A} \right) \quad (2)$$

และ

$$A = a^2 - d^2 + m^2 + c^2 + 2 a c - 2 m (c + a) \cos \alpha$$

$$B = 4 m c \sin \alpha$$

$$C = a^2 - d^2 + m^2 + c^2 - 2 a c + 2 m (c - a) \cos \alpha \quad (3)$$

โดย d คือ ระยะจากจุด C ไปจุด D (หน่วย เมตร)

ซึ่งสามารถนำ θ มาคำนวณหา β ที่เป็นองศาของข้อต่อจุด C ได้ ซึ่งเขียนเป็นสมการได้ดังนี้

$$\beta = \cos^{-1} \left(\frac{-a + m \cos \alpha + c \cos \theta}{d} \right) \quad (4)$$

จาก pantograph mechanism ที่เคยมีการคำนวณไว้ในเอกสารอ้างอิง [4] นำมาหาดำแหน่งของ A ได้สมการ

$$X_A = X_B - l_2 \cos \varphi_2 + l_3 \cos \varphi_3$$

$$Y_A = Y_B - l_2 \sin \varphi_2 + l_3 \sin \varphi_3 \quad (5)$$

โดย X_A คือ ตำแหน่งของจุด A ในแกน x (หน่วย เมตร)

Y_A คือ ตำแหน่งของจุด A ในแกน y (หน่วย เมตร)

l_2 คือ ระยะจากจุด B ไปจุด H (หน่วย เมตร)

l_3 คือ ระยะจากจุด A ไปจุด H (หน่วย เมตร)

φ_3 คือ มุมของข้อต่ออิสระ M ที่เป็นตัวเชื่อมโยงกับขาอื่น เทียบกับแกน x (หน่วย องศา)

และสามารถเขียนสมการ close-loop คำนวณหาองศาของ Link เทียบกับระนาบพื้นหุ่นยนต์ได้ ดังนี้

$$\begin{aligned}\varphi_2 &= 2\tan^{-1}\left(\frac{-E - (E^2 - 4DF)^{1/2}}{2D}\right) \\ \varphi_3 &= \cos^{-1}\left(\frac{-X_B + (l_2 - b_2)\cos\varphi_2}{b_1}\right)\end{aligned}\quad (6)$$

และคำนวณตำแหน่ง จุดต่างๆ จากสมการ

$$\begin{aligned}D &= X_B^2 - b_1^2 + (l_2 - b_2)^2 + Y_B^2 + p^2 + 2(l_2 - b_2)X_B + 2Y_Bp \\ E &= -4(Y_B + p(l_2 - b_2)) \\ F &= X_B^2 - b_1^2 + (l_2 - b_2)^2 + Y_B^2 + p^2 - 2(l_2 - b_2)X_B + 2Y_Bp\end{aligned}\quad (7)$$

โดย φ_2 คือ มุมของข้อต่อ B เทียบกับแกน x (หน่วย องศา)
 b_1 คือ ระยะจากจุด G ไปจุด M (หน่วย เมตร)
 b_2 คือ ระยะจากจุด I ไปจุด M (หน่วย เมตร)
 p คือ ความสูงจากจุด C ไปจุด M (หน่วย เมตร)

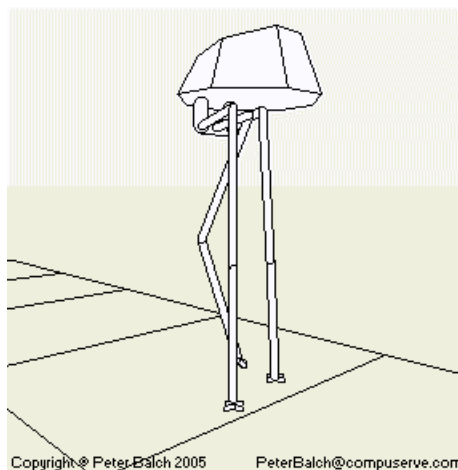
บทที่ 3 วิธีการดำเนินงาน

3.1 การออกแบบหุ่นยนต์จากภาพยนตร์

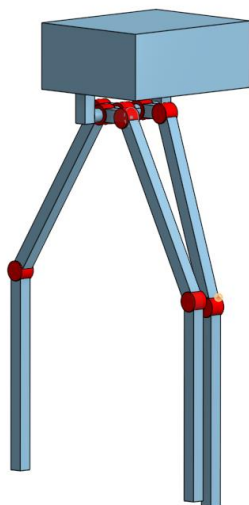
หุ่นยนต์ Tripod จากภาพยนตร์ War of the worlds มีรูปร่างที่แปลกประหลาด ซึ่งส่งผลให้การนำรูปร่างนั้นมาคำนวณจลนศาสตร์ในความเป็นจริง เป็นเรื่องที่ยากเป็นอย่างมาก ทางคณะผู้จัดทำจึงปรับปรุงและพัฒนาการออกแบบเพิ่มเติม



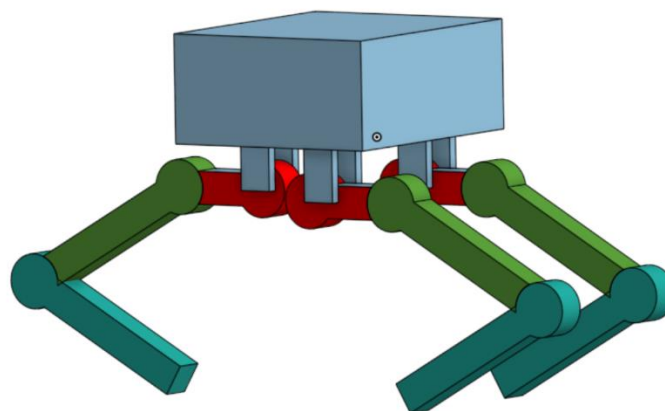
รูปที่ 8 หุ่นยนต์จากภาพยนตร์



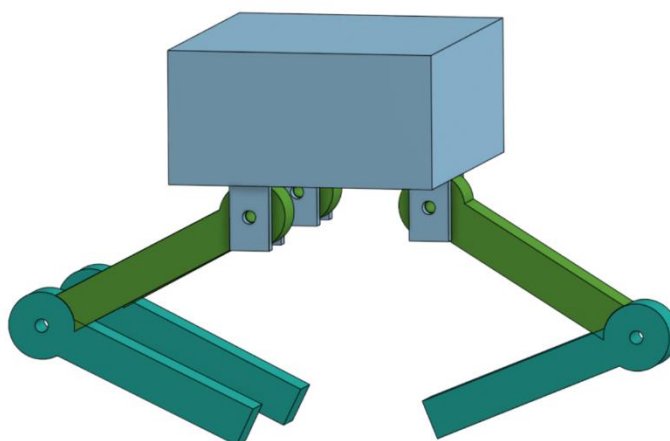
รูปที่ 9 หุ่นยนต์โดยคุณ Peter Balch [1]



รูปที่ 10 หุ่นยนต์โดยคณะผู้จัดทำแบบที่ 1



รูปที่ 11 หุ่นยนต์โดยคณะผู้จัดทำแบบที่ 2



รูปที่ 12 หุ่นยนต์โดยคณะผู้จัดทำแบบที่ 3

การออกแบบและพัฒนา

หุ่นยนต์แบบที่ 1

ลักษณะ : มี 3 ขา โดยแต่ละขามี 3 Joint และ 3 Link โดยอ้างอิงจากคุณ Peter Balch [1]

ข้อสังเกต : ขาดสมดุลในการยืน เพราะขาทั้ง 3 อยู่ในแนวเดียวกัน และไม่อยู่ตรงศูนย์กลางของหุ่นยนต์

หุ่นยนต์แบบที่ 2

ลักษณะ : มี 3 ขา โดยแต่ละขามี 3 Joint และ 3 Link

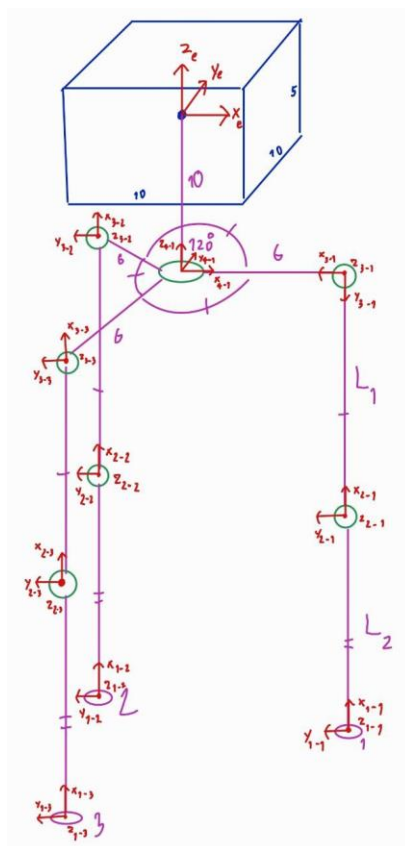
ข้อสังเกต : ในการยืนขึ้นการมี 3 Joint และ 3 Link อาจจะยังไม่จำเป็น

เนื่องจาก มี 2 Joint และ 2 Link ก็เพียงพอแล้ว

หุ่นยนต์แบบที่ 3

ลักษณะ : มี 3 ขา โดยแต่ละขามี 2 Joint และ 2 Link

3.2 Forward Kinematics



รูปที่ 13 Frame ของแต่ละ Joint

ตารางที่ 1 DH-Parameter Leg1

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	$\pi/2$	0	$\pi/2$
2	L_2	0	0	0
3	L_1	0	0	$\pi/2$
4	6	$\pi/2$	0	π
e	0	0	10	0

ตารางที่ 2 DH-Parameter Leg2

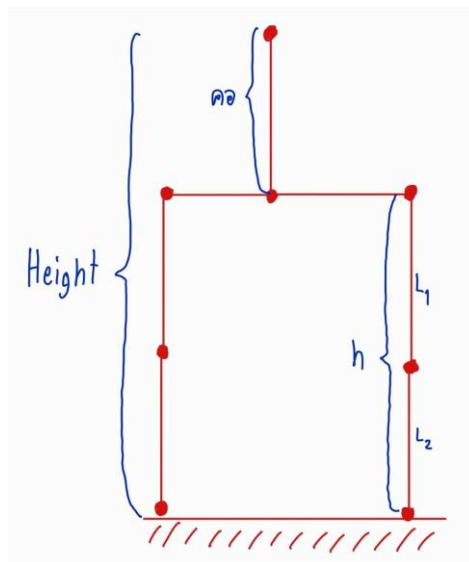
i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	$\pi/2$	0	$\pi/2$
2	L_2	0	0	0
3	L_1	0	0	$\pi/2$
4	0	$-\pi/3$	0	$-\pi/2$
5	6	$-\pi/2$	0	$\pi/3$
e	0	0	10	0

ตารางที่ 3 DH-Parameter Leg3

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	$\pi/2$	0	$\pi/2$
2	L_2	0	0	0
3	L_1	0	0	$\pi/2$
4	0	$\pi/3$	0	$-\pi/2$
5	6	$-\pi/2$	0	$-\pi/3$
e	0	0	10	0

3.3 Inverse Kinematics

ทำการรับค่า Height เข้ามา แล้วนำมาขนาดของ h เพื่อเป็นเป้าหมายในการคำนวณมุมต่าง ๆ



รูปที่ 14 ทำยีนเริ่มต้น

โดย Height คือ ความสูงของหุ่นยนต์ตั้งแต่เท้าถึงหัว

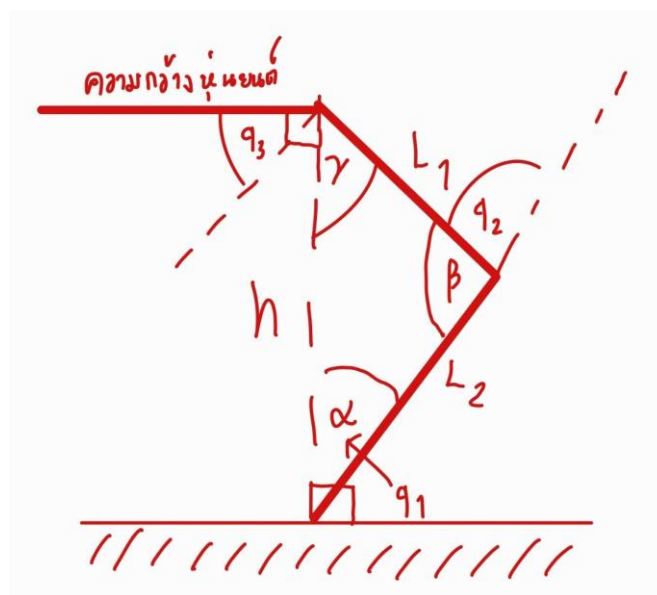
คอ คือ ระยะจากตัวของหุ่นยนต์ไปหัว

h คือ ค่าที่ต้องคำนวณมาเพื่อให้ Height ที่กำหนด

L1 คือ ความยาวของ link ที่ 1

L2 คือ ความยาวของ link ที่ 2

จากนั้นทำการหามุม β โดยใช้กฎของโคไซน์

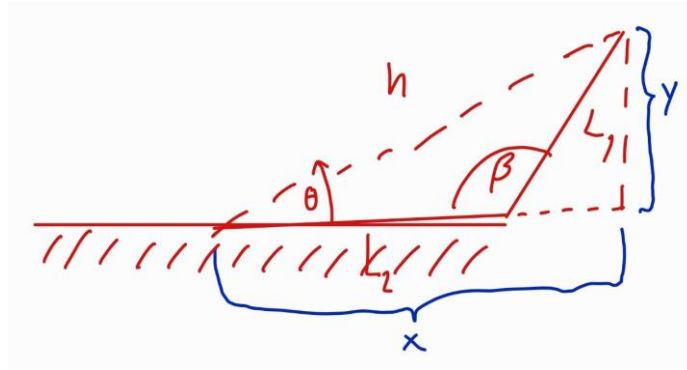


รูปที่ 15 เรขาคณิตของขาหุ่นยนต์ ในการหามุม β

จะได้

$$\beta = \cos^{-1}\left(\frac{L1^2 + L2^2 - h^2}{2 \times L1 \times L2}\right)$$

$$q2 = \pi - \beta$$



รูปที่ 16 เรขาคณิตของขาคู่หุ่นยนต์ ในการหามุม θ

หามุม θ ที่ทำให้ h ตั้งฉากกับพื้นโลก

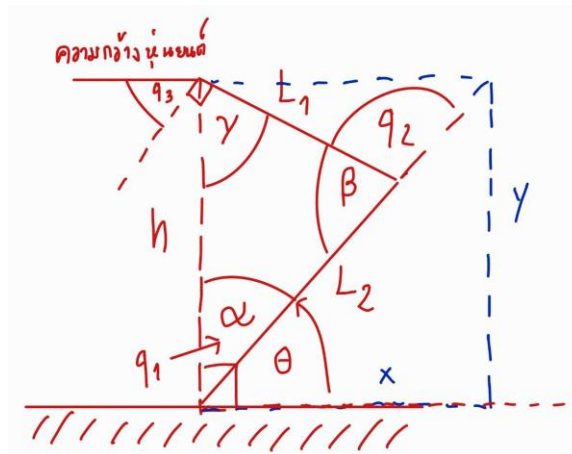
$$x = L2 + (L1 \times \cos(q2))$$

$$y = L1 \times \sin(q2)$$

$$\theta = \arctan2(y, x)$$

$$\alpha = \frac{\pi}{2} - \theta$$

$$q1 = -\alpha$$



รูปที่ 17 เรขาคณิตของขาคู่หุ่นยนต์ ในการหามุม γ

หามุม γ ที่ทำให้หุ่นยนต์ขนานกับพื้นโลก

$$\gamma = \pi - \alpha - \beta$$

$$q3 = -\gamma$$

3.4 Differential Kinematics

ใช้ Jacobian Matrix ในการคำนวณหาความเร็วในการย่นขึ้น โดยอ้างอิงจากความเร็วในการเคลื่อนที่ในแกน z ของหัวหุ่นยนต์

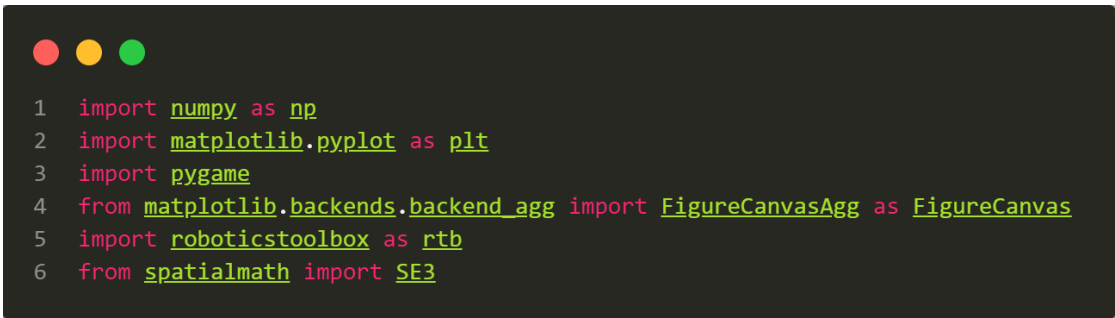
$$Diff(q) = \dot{q}$$

$$\vec{V}_e = J(q) \cdot \dot{q}$$

3.5 Program

3.5.1 ทำการ import library โดยมี library ดังนี้

- numpy
- matplotlib.pyplot
- pygame
- matplotlib.backends.backend_agg
- roboticstoolbox
- spatialmath



```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pygame
4 from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
5 import roboticstoolbox as rtb
6 from spatialmath import SE3
  
```

รูปที่ 18 คำสั่ง import library

3.5.2 ฟังก์ชัน calculate_valid_height_range คำนวณช่วงความสูงที่เหมาะสม (min และ max) ของหุ่นยนต์โดยอิงจากความยาวของ link1 และ link2 จากขาของหุ่นยนต์ โดยมี

อินพุต:

- L1 และ L2: ความยาวของลิงก์
- head_to_waist: ความสูงจากเอวถึงหัว

การทำงาน:

- คำนวณ max_height โดยรวมความยาวของลิงก์และส่วนหัว
- ใช้การวนลูปหาค่าความสูงต่ำสุด (min_height) โดยตรวจสอบว่าองศาที่คำนวณได้จากความสูงนั้นอยู่ในช่วงที่เหมาะสม
- หากเกิดข้อผิดพลาด (เช่น ค่าทางคณิตศาสตร์ไม่สมเหตุสมผล) จะข้ามไปยังค่าถัดไป

ผลลัพธ์: ช่วงความสูง (min_height, max_height)

```

1  def calculate_valid_height_range(L1, L2, head_to_waist):
2      """
3      Calculate the valid height range for the robot based on link lengths.
4
5      Args:
6      L1 (float): Length of first link
7      L2 (float): Length of second link
8      head_to_waist (float): Distance from waist to head
9
10     Returns:
11     tuple: (min_height, max_height)
12     """
13     # Maximum possible height (fully extended)
14     max_height = L1 + L2 + head_to_waist
15
16     # Minimum possible height (finding the lowest point where joint angles
17     # are valid)
18     min_height = head_to_waist # Start with head_to_waist as initial minimum
19
20     for height in range(int(head_to_waist), int(max_height) + 1):
21         try:
22             # Calculate joint angles to check validity
23             beta_test = np.arccos((L1**2 + L2**2 - (height - head_to_waist)**2) / (2 * L1 * L2))
24             q2_test = np.pi - beta_test
25
26             x_h_test = L2 + L1 * np.cos(q2_test)
27             y_h_test = L1 * np.sin(q2_test)
28
29             out_theta_test = np.arctan2(x_h_test, y_h_test)
30             alpha_test = np.pi/2 - out_theta_test
31
32             # Check if angle is within a reasonable range
33             if np.rad2deg(alpha_test) < 90:
34                 min_height = height
35                 break
36         except Exception:
37             # If calculation fails, continue to next iteration
38             continue
39
40     return min_height, max_height

```

รูปที่ 19 ฟังก์ชัน calculate_valid_height_range

3.5.2 ฟังก์ชัน create_robot_legs สร้างโมเดลขา 3 ขาของหุ่นยนต์ โดยใช้พารามิเตอร์ Denavit-Hartenberg (DH-Parameter)

อินพุต: L1 และ L2: ความยาวของลิงก์

การทำงาน:

- นิยามขาแต่ละขาด้วย rtb.DHRobot และกำหนดค่าพารามิเตอร์ DH-Parameter
- ขาแต่ละขามีตำแหน่งฐานที่แตกต่างกัน (Leg1, Leg2, Leg3)

ผลลัพธ์: โมเดลขาทั้งสาม (Leg1, Leg2, Leg3)

```

1 def create_robot_legs(L1, L2):
2     # Leg 1
3     Leg1 = rtb.DHRobot(
4         [
5             rtb.RevoluteMDH(alpha=np.pi/2, offset=np.pi/2),
6             rtb.RevoluteMDH(a=L2),
7             rtb.RevoluteMDH(a=L1, offset=np.pi/2),
8             rtb.RevoluteMDH(a=6, alpha=np.pi/2, offset=np.pi),
9             rtb.RevoluteMDH(),
10            rtb.RevoluteMDH(d=10),
11            rtb.RevoluteMDH(alpha=np.pi/2),
12            rtb.RevoluteMDH(a=5, alpha=-np.pi/2),
13        ], base=SE3(6, 0, 0),
14        name="Leg1"
15    )
16
17    # Leg 2
18    Leg2 = rtb.DHRobot(
19        [
20            rtb.RevoluteMDH(alpha=np.pi/2, offset=np.pi/2),
21            rtb.RevoluteMDH(a=L2),
22            rtb.RevoluteMDH(a=L1),
23            rtb.RevoluteMDH(alpha=-np.pi/3, offset=-np.pi/2),
24            rtb.RevoluteMDH(a=6, alpha=-np.pi/2, offset=-np.pi/3),
25            rtb.RevoluteMDH(d=10),
26            rtb.RevoluteMDH(alpha=np.pi/2),
27            rtb.RevoluteMDH(a=5, alpha=-np.pi/2),
28        ], base=SE3(-np.sin(np.deg2rad(30))*6, np.cos(np.deg2rad(30))*6,
29        0),
30        name="Leg2"
31    )
32
33    # Leg 3
34    Leg3 = rtb.DHRobot(
35        [
36            rtb.RevoluteMDH(alpha=np.pi/2, offset=np.pi/2),
37            rtb.RevoluteMDH(a=L2),
38            rtb.RevoluteMDH(a=L1),
39            rtb.RevoluteMDH(alpha=-np.pi/3, offset=-np.pi/2),
40            rtb.RevoluteMDH(a=6, alpha=-np.pi/2, offset=-np.pi/3),
41            rtb.RevoluteMDH(d=10),
42            rtb.RevoluteMDH(alpha=np.pi/2),
43            rtb.RevoluteMDH(a=5, alpha=-np.pi/2),
44        ], base=SE3(-np.sin(np.deg2rad(30))*6, -np.cos(np.deg2rad(30))*6,
45        0),
46        name="Leg3"
47    )
48
49    return Leg1, Leg2, Leg3

```

รูปที่ 20 ฟังก์ชัน create_robot_legs

3.5.4 ฟังก์ชัน calculate_angles คำนวณองศาของข้อต่อสำหรับความสูงและมุมที่กำหนด

อินพุต:

- Height: ความสูงที่ต้องการ
- L1 และ L2: ความยาวของลิงก์
- Degree_of_head_tilt: องศาการเอียงของหัว

การทำงาน:

- คำนวณองศาข้อต่อ (q1, q2, q3, q7) โดยใช้กฎของโคไซน์และตรีโกณมิติ
- ตรวจสอบว่าความสูงอยู่ในช่วงที่ขาเอื้อมถึงได้

ผลลัพธ์: ชุดองศาข้อต่อ (q1, q2, q3, q7)

```

1 def calculate_angles(Height, L1, L2, Degree_of_head_tilt):
2     h = Height - 10 # Head_to_Waist is 10
3     # Avoid domain errors in arccos
4     if h > (L1 + L2) or h < abs(L1 - L2):
5         raise ValueError("Height is out of reachable range for the robot legs.")
6
7     beta = np.arccos((L1**2 + L2**2 - h**2) / (2 * L1 * L2))
8     q2 = np.pi - beta
9
10    x_h = L2 + L1 * np.cos(q2)
11    y_h = L1 * np.sin(q2)
12    out_theta = np.arctan2(x_h, y_h)
13    alpha = np.pi / 2 - out_theta
14    q1 = -(np.pi / 2 - out_theta)
15
16    gamma = np.pi - alpha - beta
17    q3 = -gamma
18    q7 = np.deg2rad(Degree_of_head_tilt)
19
20    return q1, q2, q3, q7

```

รูปที่ 21 ฟังก์ชัน calculate_angles

3.5.5 ฟังก์ชัน `plot_robot` แสดงผลตำแหน่งและท่าทางของหุ่นยนต์ในรูปแบบกราฟ 3D และ 2D

อินพุต: `height`, `degree_of_head_tilt`, `L1`, `L2`

การทำงาน:

- ใช้ฟังก์ชัน `create_robot_legs` เพื่อสร้างขาและ `calculate_angles` เพื่อคำนวณองศา
- คำนวณตำแหน่งของข้อต่อใน 3 มิติ
- แสดงกราฟ:
 - กราฟ 3D: ขาทั้งสามขา
 - กราฟ 2D (XZ และ YZ): โครงร่างขาในระนาบ
- แปลงกราฟเป็น Pygame surface

ผลลัพธ์: พื้นผิวกราฟและข้อมูลขาหุ่นยนต์

```

1 def plot_robot(height, degree_of_head_tilt, L1, L2):
2     Leg1, Leg2, Leg3 = create_robot_legs(L1, L2)
3     q1, q2, q3, q7 = calculate_angles(height, L1, L2, degree_of_head_tilt)
4
5     q_Leg1 = [q1, q2, q3, 0, 0, 0, q7, 0]
6     q_Leg2 = [-q1, -q2, -q3, 0, 0, 0, q7, 0]
7     q_Leg3 = [-q1, -q2, -q3, 0, 0, 0, q7, 0]
8
9     def get_joint_positions(robot, q):
10         T = robot.fkine_all(q)
11         return np.array([T[i].t for i in range(len(T))])
12
13     # Get joint positions for each leg
14     positions_leg1 = get_joint_positions(Leg1, q_Leg1)
15     positions_leg2 = get_joint_positions(Leg2, q_Leg2)
16     positions_leg3 = get_joint_positions(Leg3, q_Leg3)
17
18     # Create figure with multiple subplots
19     fig = plt.figure(figsize=(8, 6))
20
21     # 3D Plot (first subplot)
22     ax = fig.add_subplot(221, projection="3d")
23     ax.set_title("Leg1, Leg2, Leg3 Plots in One Figure")
24
25     # Plot lines for each leg in 3D
26     ax.plot(positions_leg1[:, 0], positions_leg1[:, 1], positions_leg1[:,
27 ], 'o-', Label="Leg1", color='r')
28     ax.plot(positions_leg2[:, 0], positions_leg2[:, 1], positions_leg2[:,
29 ], 'o-', Label="Leg2", color='g')
30     ax.plot(positions_leg3[:, 0], positions_leg3[:, 1], positions_leg3[:,
31 ], 'o-', Label="Leg3", color='b')
32
33     ax.set_xlabel("X")
34     ax.set_ylabel("Y")
35     ax.set_zlabel("Z")
36     ax.legend()
37     ax.grid(True)
38
39     # XZ Plane (second subplot)
40     ax2 = fig.add_subplot(222)
41     ax2.plot(positions_leg1[:, 0], positions_leg1[:, 2], 'o-', Label="Leg
42 1", color='r')
43     ax2.plot(positions_leg2[:, 0], positions_leg2[:, 2], 'o-', Label="Leg
44 2", color='g')
45     ax2.plot(positions_leg3[:, 0], positions_leg3[:, 2], 'o-', Label="Leg
46 3", color='b')
47     ax2.set_title("XZ Plane")
48     ax2.set_xlabel("X (m)")
49     ax2.set_ylabel("Z (m)")
50     ax2.legend()
51     ax2.grid(True)
52     ax2.axis("equal")
53
54     # YZ Plane (third subplot)
55     ax3 = fig.add_subplot(223)
56     ax3.plot(positions_leg1[:, 1], positions_leg1[:, 2], 'o-', Label="Leg
57 1", color='r')
58     ax3.plot(positions_leg2[:, 1], positions_leg2[:, 2], 'o-', Label="Leg
59 2", color='g')
60     ax3.plot(positions_leg3[:, 1], positions_leg3[:, 2], 'o-', Label="Leg
61 3", color='b')
62     ax3.set_title("YZ Plane")
63     ax3.set_xlabel("Y (m)")
64     ax3.set_ylabel("Z (m)")
65     ax3.legend()
66     ax3.grid(True)
67     ax3.axis("equal")
68
69     # Adjust layout
70     plt.tight_layout()
71
72     # Convert to Pygame surface
73     canvas = FigureCanvas(fig)
74     canvas.draw()
75     raw_data = np.frombuffer(canvas.tostring_rgb(), dtype=np.uint8)
76     width, height = canvas.get_width_height()
77     surface = pygame.image.frombuffer(raw_data, (width, height), "RGB")
78     plt.close(fig)
79
80     return surface, Leg1, Leg2, Leg3, q_Leg1, q_Leg2, q_Leg3

```

รูปที่ 22 ฟังก์ชัน plot_robot

3.5.6 ฟังก์ชัน main จัดการแอปพลิเคชันแบบเรียลไทม์สำหรับปรับพารามิเตอร์และแสดงผล

อินพุต: L1, L2, head_to_waist: พารามิเตอร์เริ่มต้น

การทำงาน:

- เรียกใช้ Pygame เพื่อแสดง GUI
- แสดงกราฟจาก plot_robot
- ผู้ใช้สามารถแก้ไขค่าพารามิเตอร์ (เช่น ความสูง, ความยาวลิงก์, และมุมหัว)
- อัปเดตกราฟแบบเรียลไทม์

ผลลัพธ์: อินเทอร์เฟซที่ผู้ใช้สามารถปรับแต่งหุ่นยนต์และเห็นผลแบบเรียลไทม์


```

1 def main(L1, L2, head_to_waist):
2     # Use these local variables instead of global
3     height = HEIGHT
4     degree_of_head_tilt = DEGREE_OF_HEAD_TILT
5
6     # Dynamically calculate height range
7     min_height, max_height = calculate_valid_height_range(L1, L2, head_to_waist)
8
9     # Pygame setup
10    pygame.init()
11    screen = pygame.display.set_mode((1400, 850)) # Increased height to accommodate larger plot
12    pygame.display.set_caption("Robot Real-Time Plot with Adjustable Parameters")
13    font = pygame.font.SysFont(None, 36)
14    input_font = pygame.font.SysFont(None, 28)
15
16    # Initial parameters
17    input_active = False
18    input_text = ""
19    param_selection = "Height"
20
21    # Main game loop
22    running = True
23    while running:
24        # Recalculate height range in case L1 or L2 changed
25        min_height, max_height = calculate_valid_height_range(L1, L2, head_to_waist)
26
27        # Ensure current height is within new range
28        height = max(min_height, min(height, max_height))
29
30        screen.fill((255, 255, 255))
31
32        # Plot robot and get surface
33        graph_surface, Leg1, Leg2, Leg3, q_Leg1, q_Leg2, q_Leg3 = plot_robot(height, degree_of_head_tilt, L1, L2)
34        screen.blit(graph_surface, (50, 50))
35
36        # Draw parameter input rectangle
37        pygame.draw.rect(screen, (230, 230, 230), (900, 80, 400, 380), border_radius=10)
38        pygame.draw.rect(screen, (200, 200, 200), (910, 100, 380, 40), border_radius=5)
39        input_surface = input_font.render(input_text, True, (0, 0, 0))
40        screen.blit(input_surface, (910, 110))
41
42        # Render parameter texts
43        text_height = font.render(f"Height: {height:.2f} meter", True, (0, 0, 0))
44        text_tilt = font.render(f"Head Tilt: {degree_of_head_tilt} degree", True, (0, 0, 0))
45        text_L1 = font.render(f"Link 1: {L1:.2f} meter", True, (0, 0, 0))
46        text_L2 = font.render(f"Link 2: {L2:.2f} meter", True, (0, 0, 0))
47        param_text = font.render(f"Editing {param_selection} :", True, (0, 0, 0))
48
49        # Blit parameter texts
50        screen.blit(text_height, (910, 180))
51        screen.blit(text_tilt, (910, 220))
52        screen.blit(text_L1, (910, 260))
53        screen.blit(text_L2, (910, 300))
54        screen.blit(param_text, (910, 340))
55
56        # Instructions
57        instruction = input_font.render("Use Tab to switch parameters", True, (0, 0, 0))
58        screen.blit(instruction, (50, 680))
59        instruction2 = input_font.render("Enter value and press Enter to update", True, (0, 0, 0))
60        screen.blit(instruction2, (50, 710))
61        instruction3 = input_font.render(f"Maximum Height: {max_height:.2f} meter", True, (0, 0, 0))
62        screen.blit(instruction3, (910, 390))
63        instruction4 = input_font.render(f"Minimum Height: {min_height:.2f} meter", True, (0, 0, 0))
64        screen.blit(instruction4, (910, 410))

```

รูปที่ 23 ฟังก์ชัน main

```

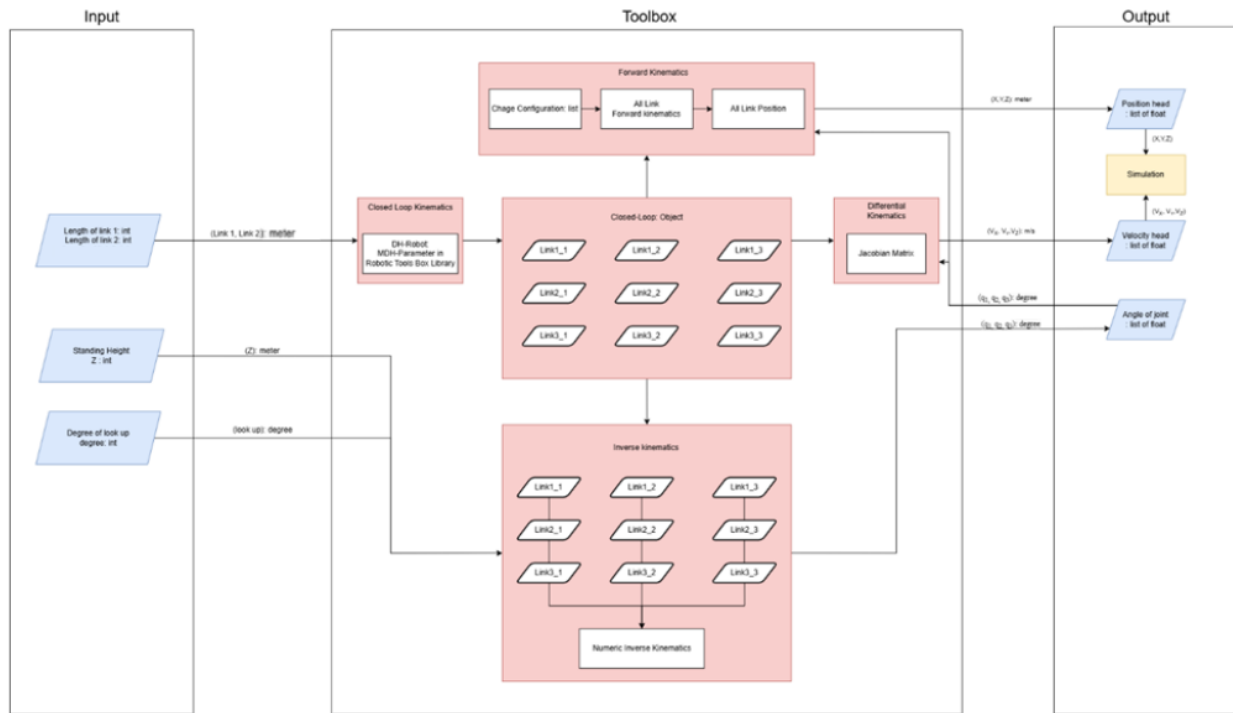
1
2     # Event handling
3     for event in pygame.event.get():
4         if event.type == pygame.QUIT:
5             running = False
6         if event.type == pygame.KEYDOWN:
7             if input_active:
8                 if event.key == pygame.K_RETURN:
9                     try:
10                        new_value = float(input_text)
11
12                        if param_selection == "Height":
13                            height = max(min_height, min(new_value, max_height))
14                        elif param_selection == "Degree_of_head_tilt":
15                            degree_of_head_tilt = new_value
16                        elif param_selection == "L1":
17                            L1 = new_value
18                        elif param_selection == "L2":
19                            L2 = new_value
20                        input_text = ""
21                    except ValueError:
22                        input_text = ""
23                elif event.key == pygame.K_BACKSPACE:
24                    input_text = input_text[:-1]
25                else:
26                    input_text += event.unicode
27            elif event.key == pygame.K_TAB:
28                if param_selection == "Height":
29                    param_selection = "Degree_of_head_tilt"
30                elif param_selection == "Degree_of_head_tilt":
31                    param_selection = "L1"
32                elif param_selection == "L1":
33                    param_selection = "L2"
34                else:
35                    param_selection = "Height"
36        if event.type == pygame.MOUSEBUTTONDOWN:
37            if 910 <= event.pos[0] <= 1140 and 100 <= event.pos[1] <= 140:
38                input_active = True
39            else:
40                input_active = False
41
42        pygame.display.flip()
43
44    pygame.quit()
45
46    # Call main with initial parameter values
47    if __name__ == "__main__":
48        main(L1, L2, HEAD_TO_WAIST)

```

รูปที่ 24 ฟังก์ชัน main ส่วนที่ 2

บทที่ 4 ผลการดำเนินงาน

4.1 System Diagram / System Overview (Function and Argument)



รูปที่ 25 System Diagram

4.1.1 Input

- ฟังก์ชันสำหรับกำหนดความยาวขาระหว่างข้อต่อหุ่นยนต์:
 - เป็นฟังก์ชันที่ใช้ในการกำหนดความยาวขาของหุ่นยนต์โดยรับค่าเป็นจำนวนเต็มในหน่วยเมตร
- ฟังก์ชันสำหรับกำหนดความสูงในการยืนหุ่นยนต์:
 - เป็นฟังก์ชันที่ใช้ในการกำหนดความสูงในการยืนของหุ่นยนต์โดยรับค่าเป็นจำนวนเต็มในหน่วยเมตร
- ฟังก์ชันสำหรับกำหนดองศาการเงยหน้าหุ่นยนต์:
 - เป็นฟังก์ชันที่ใช้ในการกำหนดองศาการเงยหน้าของหุ่นยนต์โดยรับค่าเป็นจำนวนเต็มในหน่วยองศา

4.1.2 Toolbox

1) ฟังก์ชันสำหรับการหา Forward Kinematics:

- ฟังก์ชันนี้จะคำนวณตำแหน่งของ Head โดยอ้างอิงจากมุมข้อต่อต่างๆ ของหุ่นยนต์ ที่ถูกกำหนดไว้
- Forward Kinematics คำนวณหาตำแหน่งของปลายหุ่นยนต์ Head ในระบบพิกัด (X, Y, Z) โดยใช้มุมข้อต่อต่างๆ ในปัจจุบัน

2) ฟังก์ชันสำหรับการหา Inverse Kinematics:

- ฟังก์ชันนี้ใช้พารามิเตอร์ที่กำหนดไว้ใน ฟังก์ชันกำหนดความสูงในการยืนหุ่นยนต์ และฟังก์ชันกำหนดองศาการเงยหน้าหุ่นยนต์ เพื่อหาค่ามุมข้อต่อที่ทำให้ Head ของหุ่นยนต์สามารถไปถึงตำแหน่งเป้าหมายใน Task Space
- การคำนวณนี้ใช้วิธี Numerical Method โดยจะปรับแต่งมุมข้อต่ออย่างต่อเนื่อง จนกระทั่งได้ตำแหน่งที่ต้องการ เพื่อให้สามารถคำนวณหา Configuration ของข้อต่อที่เหมาะสมได้

3) ฟังก์ชันสำหรับการหา Differential Kinematics:

- ฟังก์ชันนี้ใช้ค่ามุมข้อต่อของหุ่นยนต์ในปัจจุบัน เพื่อหาความเร็ว Head ของหุ่นยนต์
- การคำนวณนี้ใช้ Jacobian Matrix โดยจะรับมุมข้อต่ออย่างต่อเนื่อง เพื่อให้เห็น ลักษณะการเคลื่อนที่ของหุ่นยนต์

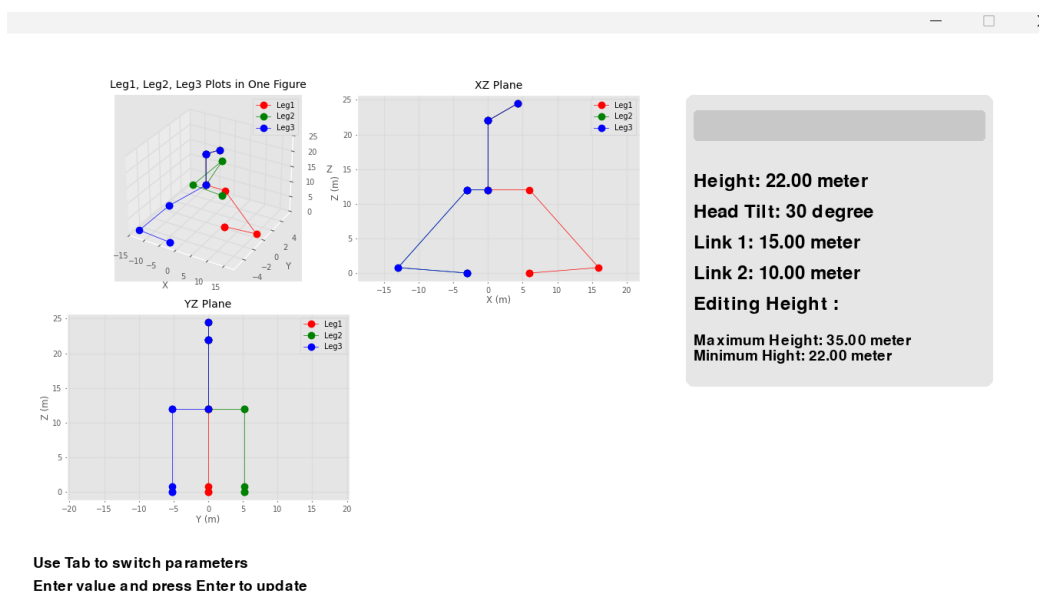
4.1.3 Output

1) การแสดงผลลักษณะการเคลื่อนที่ของหุ่นยนต์ (Simulation):

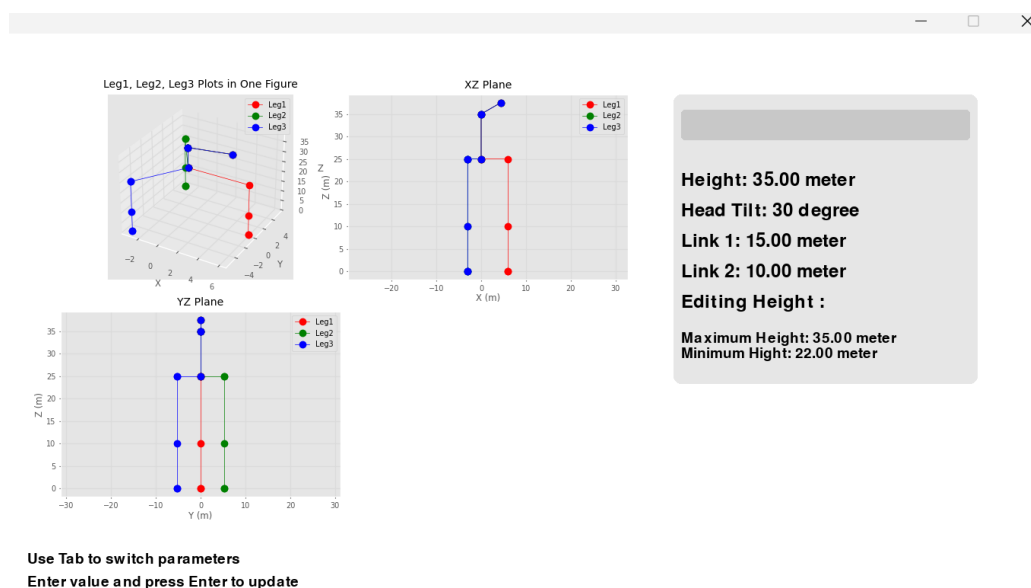
- ค่าที่ได้จากการคำนวณ Forward Kinematics จะถูกนำมาแสดงการเคลื่อนที่ของ หุ่นยนต์ ซึ่งช่วยให้เห็นลักษณะการเคลื่อนที่ของลิงก์และข้อต่อต่างๆ ของหุ่นยนต์ใน รูปแบบภาพ
- การ Simulation นี้ทำให้สามารถสังเกตท่าทางของหุ่นยนต์ได้อย่างชัดเจน โดยเฉพาะการหมุนและการจัดทำทางของข้อต่อต่างๆ ตามที่กำหนด

4.2 Simulation

4.2.1 หน้าต่างการแสดงผลและกรอกค่าของความสูง, มุมเงย, ความยาวของ link ที่ 1 และความยาวของ link ที่ 2 ของหุ่นยนต์ มีกราฟแสดงท่าทางของหุ่นยนต์ในรูปแบบ 3มิติ และ 2มิติ หุ่นยนต์สามารถขยับท่าทางขึ้นและนั่งลงได้ตาม forward kinematics, inverse kinematics และ differential kinematics ที่คำนวณไว้ได้ ดังภาพที่ 26 และ 27

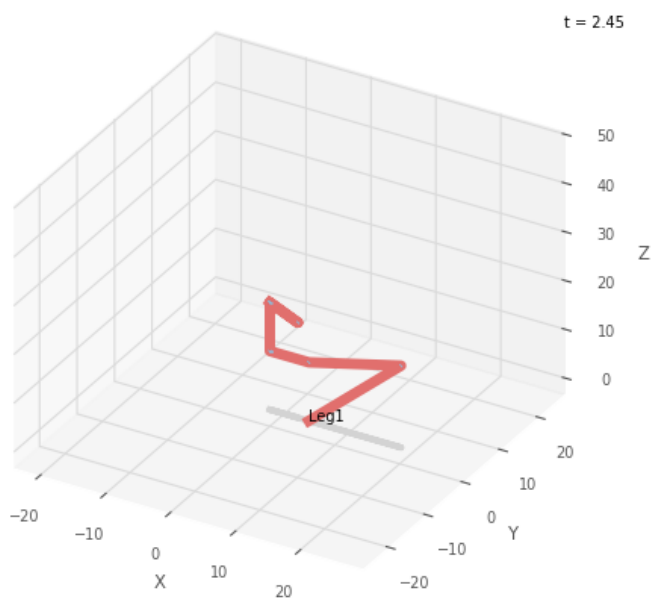


รูปที่ 26 หน้าจอแสดงผลของหุ่นยนต์ tripod ท่าทางนั่ง

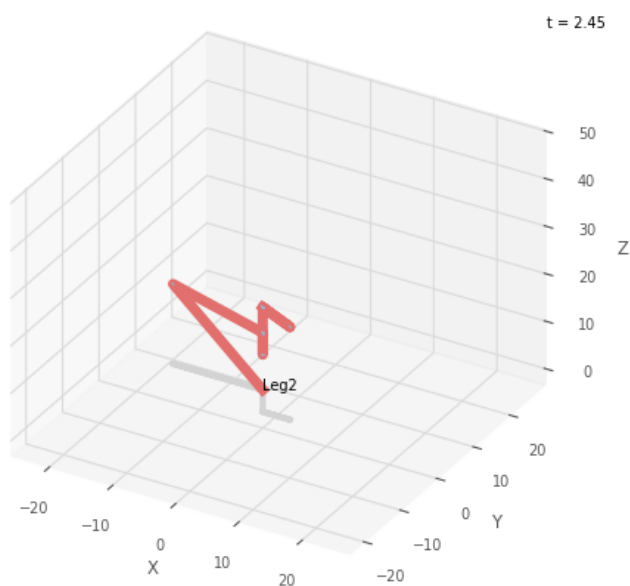


รูปที่ 27 หน้าจอแสดงผลของหุ่นยนต์ tripod ท่าทางยืน

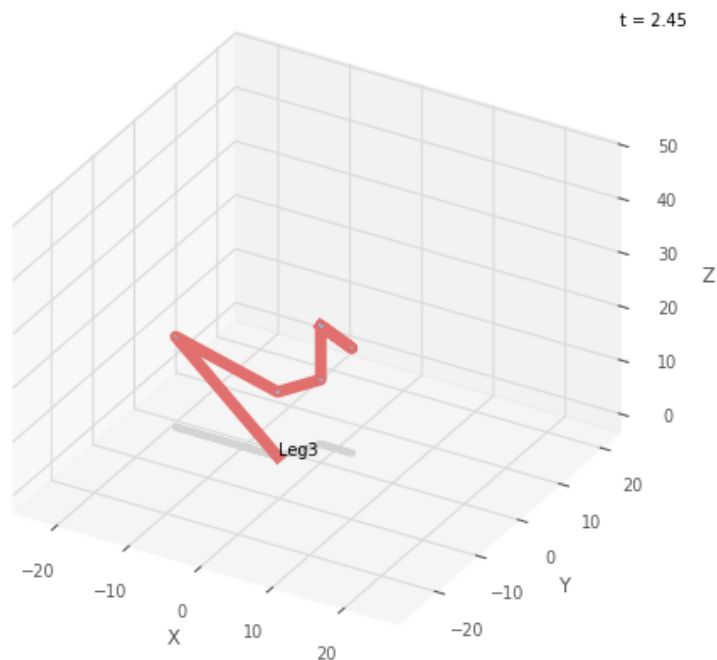
4.2.2 รูปภาพแสดงท่าทางการเคลื่อนที่ของขาของหุ่นยนต์ด้วย trajectory ด้วยภาพที่ 28, 29 และ 30 จะเป็นการแสดงเส้นทางการเคลื่อนที่ของขาที่ 1, 2 และ 3 ของหุ่นยนต์ตามลำดับ และหน้าจอแสดงผลภาพรวมของการเคลื่อนที่ของทุกขารวมกันเป็นรูปที่ 31



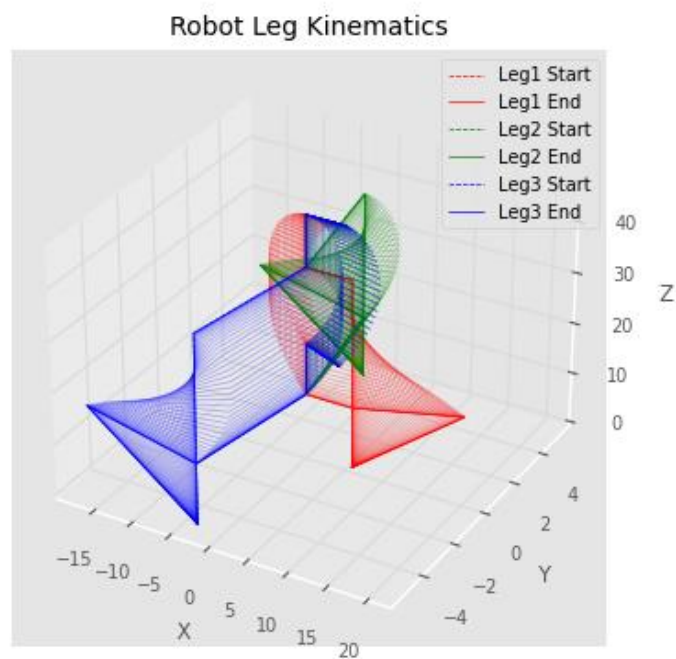
รูปที่ 28 ภาพแสดงการเคลื่อนที่ของขาที่ 1 ของหุ่นยนต์



รูปที่ 29 ภาพแสดงการเคลื่อนที่ของขาที่ 2 ของหุ่นยนต์



รูปที่ 30 ภาพแสดงการเคลื่อนที่ของขาที่ 3 ของหุ่นยนต์



รูปที่ 31 ภาพแสดง path การเคลื่อนที่ของขาหุ่นยนต์ทั้ง 3 ขา

บทที่ 5 สรุป อภิปรายผล และข้อเสนอแนะ

5.1 สรุป

จากการทำ Simulation หุ่นยนต์ Tripod จากภาพยนตร์ War of the worlds ให้สามารถยืนขึ้นในแนวตั้งและเงยหน้าได้ โดยทราบความยาวของขา (Fixed Link) เพื่อคำนวณหาตำแหน่งและองศาของหัว (end effector) จาก Forward Kinematic และใช้ Inverse Kinematic เพื่อคำนวณหา Configuration Space ที่ทำให้หุ่นยนต์ Tripod ยืนขึ้นในแนวตั้งและเงยหน้าตามที่ต้องการ โดยใช้ DH-parameter และมีการแสดงผลของท่าทางที่สามารถทรงตัวได้ ท่าคณะผู้จัดทำได้ดำเนินการตามขั้นตอนการดำเนินงาน ดังนี้

1. ศึกษาหาข้อมูลเกี่ยวกับ 3-RRR Kinematics จากงานวิจัย A NOVEL BIOLOGICALLY INSPIRED TRIPOD WALKING ROBOT (Marco Ceccarelli, Conghui Liang, Giuseppe Carbone., 2009) เพื่อศึกษาสมการ kinematics ของหุ่นยนต์ 3-RRR ที่มีลักษณะคล้ายหุ่นยนต์ tripod
2. คำนวณ Forward Kinematic และ Inverse Kinematic ของหุ่นยนต์ Tripod เพื่อนำไปใช้ในการคำนวณค่าของ End-effector และมุมของข้อต่อแต่ละจุดของหุ่นยนต์ เมื่อหุ่นยนต์ต้องการให้หุ่นยนต์ทำท่าทางนั่งหรือยืน และคำนวณมุมเงยของหัวของหุ่นยนต์ตามที่ต้องการ
3. ทำ Simulation ของหุ่นยนต์ Tripod นำสมการ Forward Kinematic และ Inverse Kinematic มาเขียนโปรแกรมเพื่อทำการจำลองท่าทางของหุ่นยนต์ Tripod ใช้ ภาษา Python ในการเขียนโปรแกรม และ library ดังนี้ 1) robotictoolbox ในการคำนวณ kinematics ของหุ่นยนต์ 2) matplotlib.pyplot ในการใส่ค่าของความยาวขา, ความสูงของหุ่นยนต์, ความยาวของแต่ละ link ของขา, มุมเงยของหุ่นยนต์, องศาของแต่ละข้อต่อ และ kinematics ที่ได้ในการ plot กราฟ 3) pygame ใช้ทำหน้าต่างแสดงผลของท่าทางของหุ่นยนต์ Tripod ในรูปแบบปริภูมิ 2 มิติและ 3 มิติและหน้าต่างสำหรับการกรอกค่าที่ต้องการกำหนด

5.2 อภิปรายผล

- วัตถุประสงค์ศึกษาของหุ่นยนต์ tripod แบบ revolute จากทำยืนในปริภูมิ 2 มิติและศึกษาการคำนวณ Forward Kinematic และ Inverse Kinematic ตรวจสอบตำแหน่งและองศาของหัว หุ่นยนต์ใน Task Space ทางคณะผู้จัดทำเข้าใจและสามารถคำนวณหา Kinematics ของหุ่นยนต์ Tripod เพื่อทำให้หุ่นยนต์ Tripod ทำท่าทางนั่งและยืนได้ตามวัตถุประสงค์

- วัตถุประสงค์พัฒนาระบบแสดงผลการเคลื่อนไหวของหุ่นยนต์ในเชิงภาพเพื่อให้เข้าใจการทำงานของแต่ละระบบได้ง่ายขึ้นทางคณะผู้จัดทำสามารถทำหน้าตาสำหรับแสดงผลของท่าทางของหุ่นยนต์ Tripod โดยแสดงผลเป็นกราฟในปริภูมิ 2 มิติ และ 3 มิติ และสามารถกรอกค่าเพื่อเปลี่ยนแปลงความสูงของหุ่นยนต์, ความยาวของ link1 และ link2 ของขาหุ่นยนต์, มุมเงยของหุ่นยนต์ และแสดงผลตามค่าที่กรอกไปแบบเรียลไทม์ได้

5.3 ข้อเสนอแนะ

- เพิ่มเติมท่าทางในการเคลื่อนที่ของหุ่นยนต์
- ควรเพิ่มการใช้ Simulation แบบอื่นในการจำลองประกอบ เช่น PySimbody, Drake, PyBullet หรือ Mujoco เพื่อให้สามารถแสดงการทำงานของหุ่นยนต์ได้สำเร็จ

เอกสารอ้างอิง

- [1] Peter Balch. “War of the Worlds - Tripod Gait”. 2005: [War of the Worlds - Tripod Gait](#)
- [2] John J. Craig. “Introduction to Robotics: Mechanics and Control (3rd Edition)”. 2014
- [3] Marco Ceccarelli, Conghui Liang, Giuseppe Carbone. “A Novel Biologically Inspired Tripod Walking Robot” July 2009: [A-novel-biologically-inspired-tripod-walking-robot.pdf](#)
- [4] Erika Ottaviano, Marco Ceccarelli, Salvatore Grande. “A Biped Walking Mechanism for

ภาคผนวก

Code Github: https://github.com/Constantine404/FRA333_Project_Kinematics_Tripod_Robot.git