

# สัมมนาเชิงปฏิบัติการเรื่อง <sup>\*</sup> "ไอโอทีสำหรับระบบควบคุม และการเรียนรู้ของเครื่อง" <sup>\*</sup>

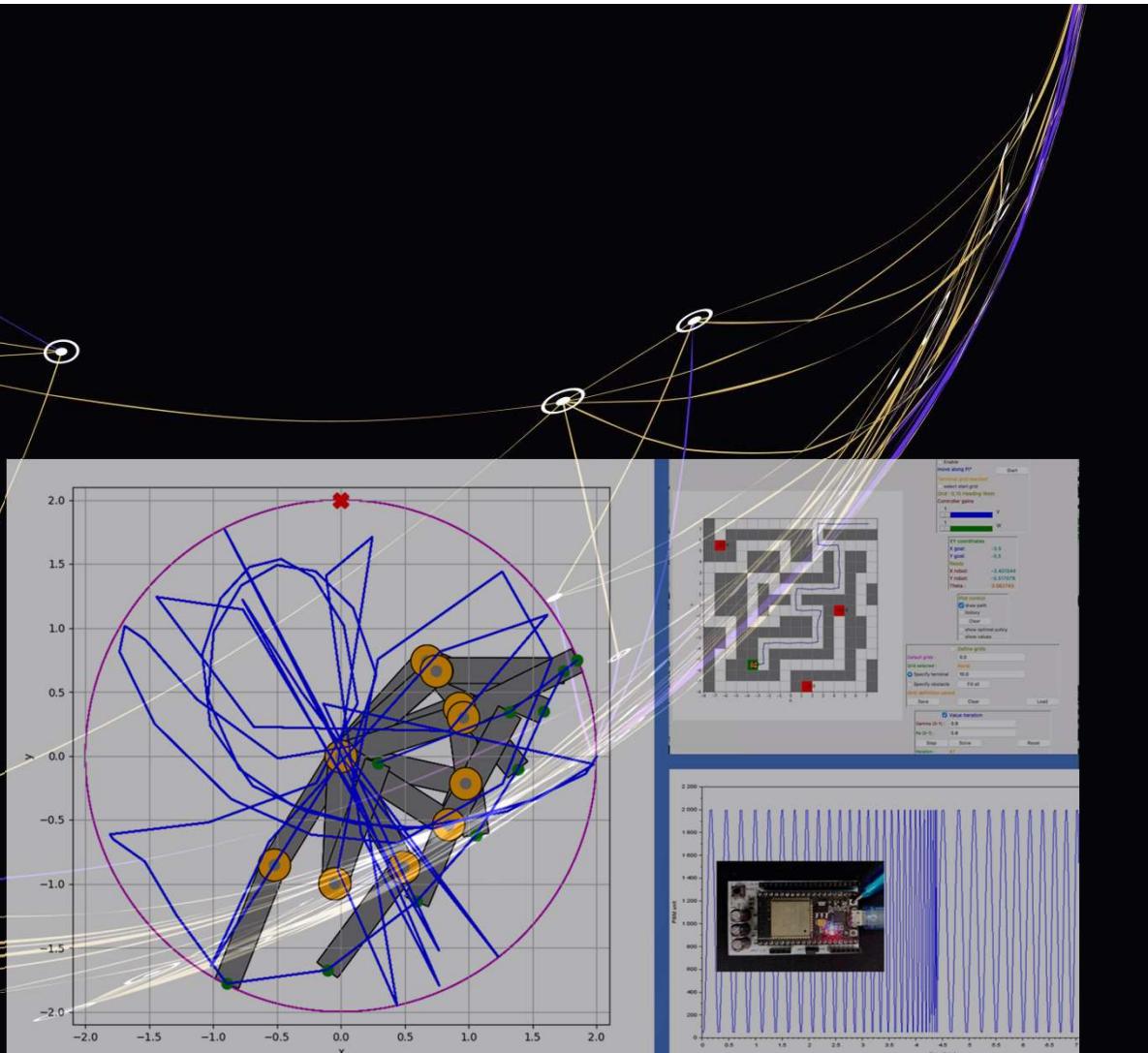
## "IOT FOR CONTROL SYSTEMS AND MACHINE LEARNING" SEMINAR & WORKSHOP

ภาควิชาพิสิกส์ คณะวิทยาศาสตร์  
มหาวิทยาลัยนเรศวร

17 กุมภาพันธ์ 2567 : 9 AM – 4 PM

ดร.วโรดม ตุ้มจินดา

ภาควิชาวิศวกรรมเครื่องกล คณะวิศวกรรมศาสตร์  
มหาวิทยาลัยเกษตรศาสตร์



# กำหนดการ

## ▶ 9 AM - 12 PM

- ติดตั้งซอฟต์แวร์ สาธิตการใช้ Jupyter notebook และ Thonny
- วงจรเชื่อมต่อ กับ MCU และการปรับแต่งสัญญาณ
- พื้นฐานของระบบควบคุม (เน้นตัวควบคุม PID)
- การอิมเพลเม้นต์ระบบควบคุมเชิงเส้น
- การจำลองระบบผังตัวบน Wokwi
- การพัฒนาไอโอทีบัน NETPIE 2020
- การแสดงผลและควบคุมบน NETPIE 2020 dashboard
- เขียนโปรแกรมไมโครไฟทอนเพื่อควบคุมอัตราการไหลของถังน้ำ 3 ระดับ

## ▶ 1 – 4 PM

- การใช้งาน paho-mqtt บน Jupyter notebook เพื่อนำข้อมูลจากระบบผังตัวมาแสดงผลในรูปแบบที่ต้องการ
- การอิมเพลเม้นต์ตัวควบคุมไม่เป็นเชิงเส้นสำหรับแขนกล 2 กำนันต่อ (2-link manipulator)
- การจำลองจลนศาสตร์หุ่นยนต์ (robot kinematics)
- ตัวควบคุมสำหรับหุ่นยนต์เคลื่อนที่ (mobile robot)
- การวางแผนเส้นทางเดินโดยอัลกอริทึมวนซ้ำมูลค่า (value iteration)
- สรุปการสัมมนา

ภาคบ่าย :

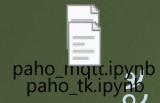
1:00 – 4:00 PM



# การใช้งาน MQTT บน Jupyter notebook (หรือ Python IDE อื่น)

ข้อดี : สามารถออกแบบส่วนติดต่อ  
ผู้ใช้ในรูปแบบตามที่ต้องการได้  
ข้อเสีย : ต้องเขียนโปรแกรมในการ  
พัฒนา





## ขั้นตอนหลัก

ติดตั้งไลบรารี paho-mqtt โดยคำสั่ง

```
!pip install paho-mqtt
```

สร้าง device บน NETPIE 2020

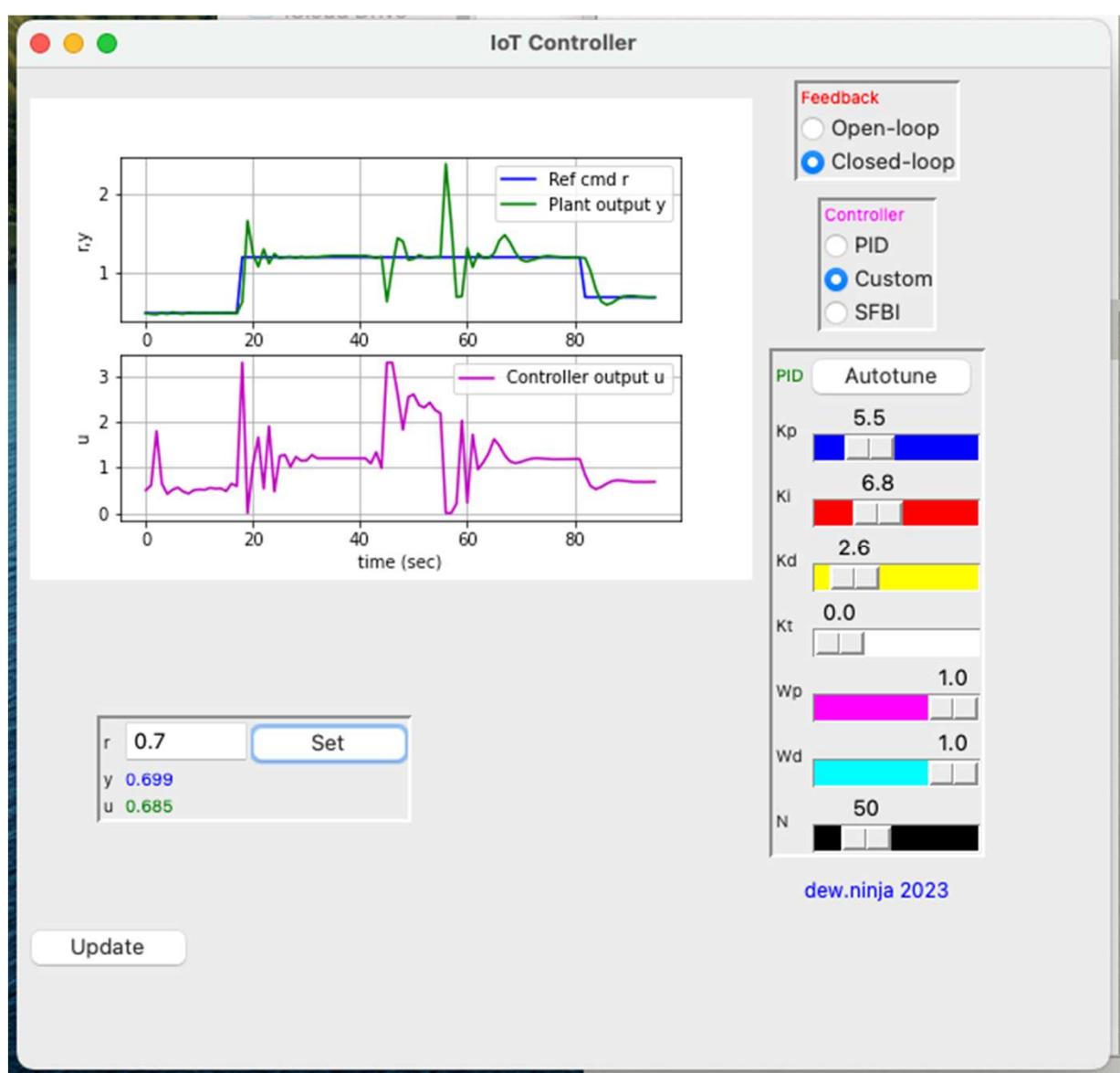
และรวมกลุ่มเข้าด้วยกัน

นำเข้าไลบรารี paho-mqtt โดยคำสั่ง

```
import paho.mqtt.client as mqtt
```

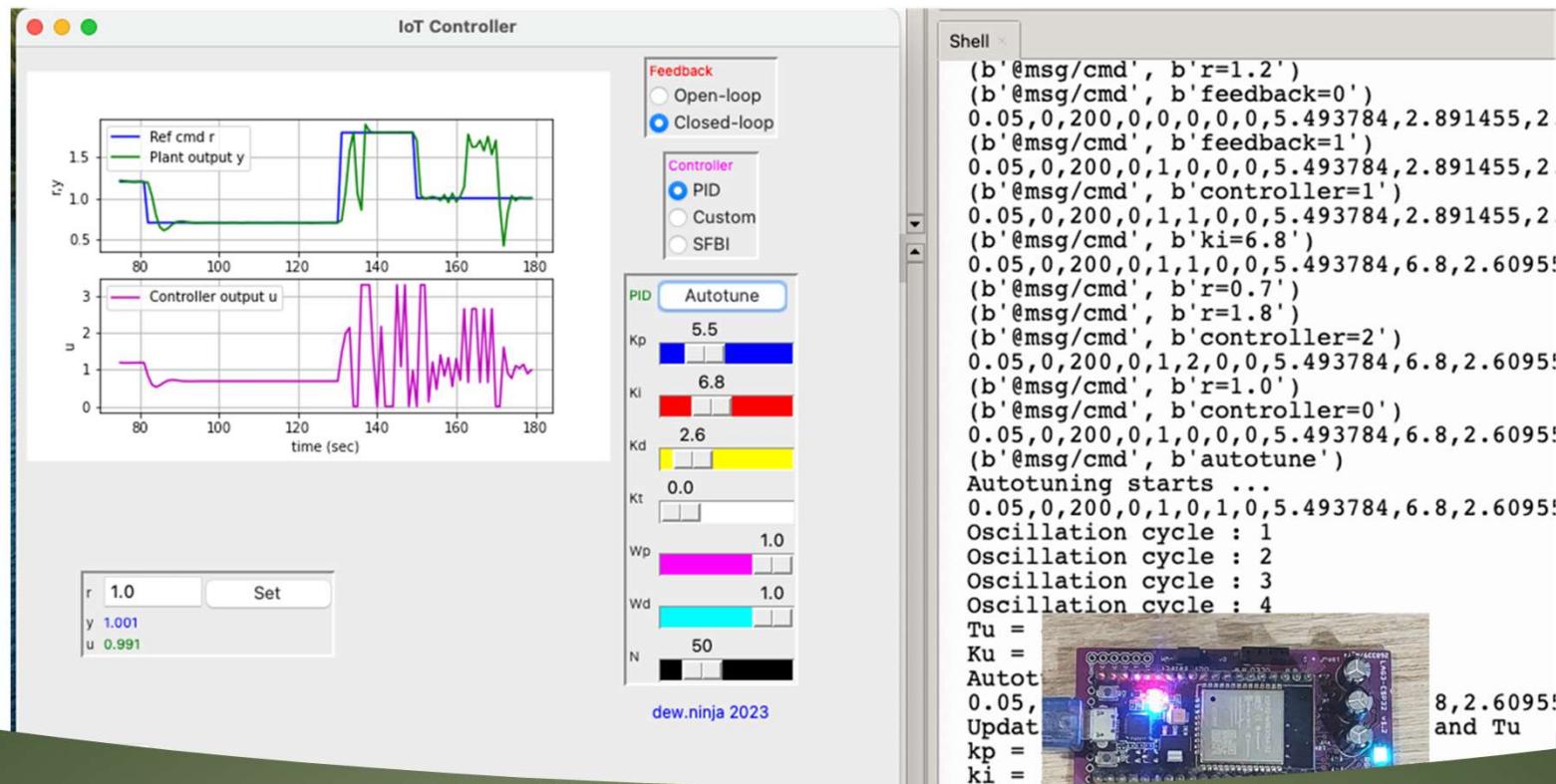
	ID	Name	Group	Status
>	68b718a6-d9b6-4d 66-97ec-b21766b...	paho connect with paho mqtt	group1	Offline
>	7cd53e80-58e4-43 22-856f-e73dae4...	nodered device for node-red	group1	Offline
>	b93a6154-e546-40 e8-87bc-afa7d61a...	aux_device device to receive @msg/update	group1	Offline
>	d206894e-c39d-42 84-9203-a303fde...	esp32 device attach to hardware	group1	Online

ขั้นตอนที่เหลือดูได้จาก Appendix A หรือ notebook **paho\_mqtt.ipynb**



ทดลองสร้าง GUI  
โดยใช้ไลบรารี tkinter

paho\_tk.ipynb



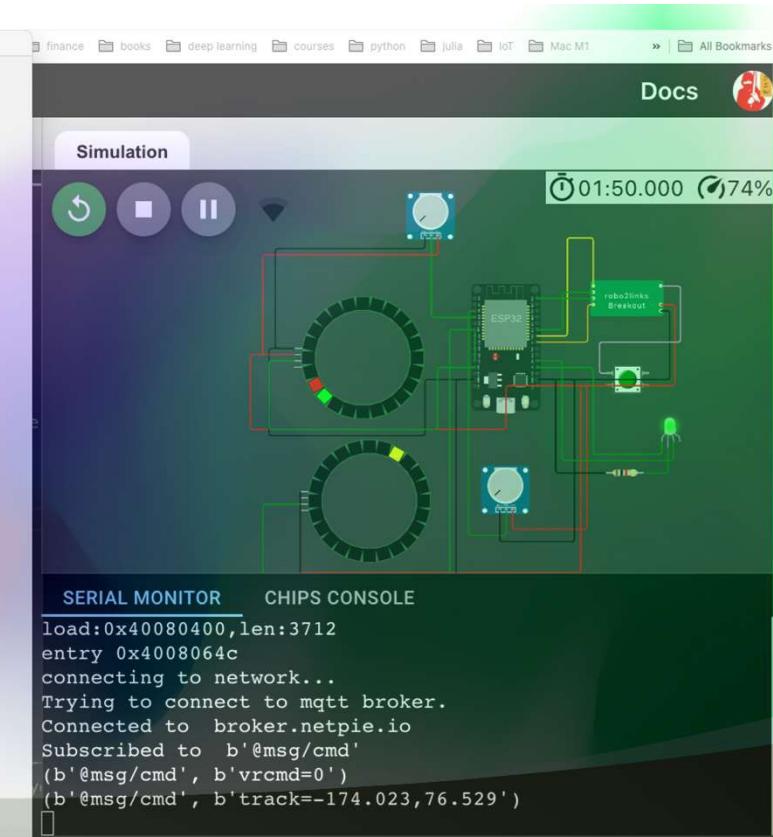
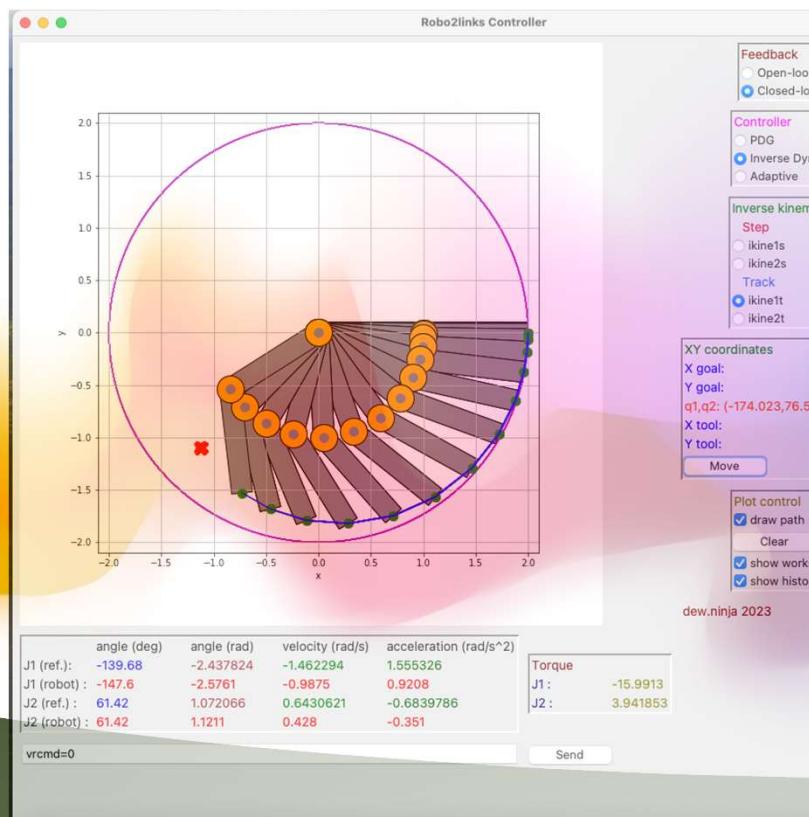
```

Shell
(b'@msg/cmd', b'r=1.2')
(b'@msg/cmd', b'feedback=0')
0.05,0,200,0,0,0,0,5.493784,2.891455,2
(b'@msg/cmd', b'feedback=1')
0.05,0,200,0,1,0,0,0,5.493784,2.891455,2
(b'@msg/cmd', b'controller=1')
0.05,0,200,0,1,1,0,0,5.493784,2.891455,2
(b'@msg/cmd', b'ki=6.8')
0.05,0,200,0,1,1,0,0,5.493784,6.8,2.6095!
(b'@msg/cmd', b'r=0.7')
(b'@msg/cmd', b'r=1.8')
(b'@msg/cmd', b'controller=2')
0.05,0,200,0,1,2,0,0,5.493784,6.8,2.6095!
(b'@msg/cmd', b'r=1.0')
(b'@msg/cmd', b'controller=0')
0.05,0,200,0,1,0,0,0,5.493784,6.8,2.6095!
(b'@msg/cmd', b'autotune')
Autotuning starts ...
0.05,0,200,0,1,0,0,5.493784,6.8,2.6095!
Oscillation cycle : 1
Oscillation cycle : 2
Oscillation cycle : 3
Oscillation cycle : 4
Tu =
Ku =
Autot
0.05,
Updat
kp =
ki =

```

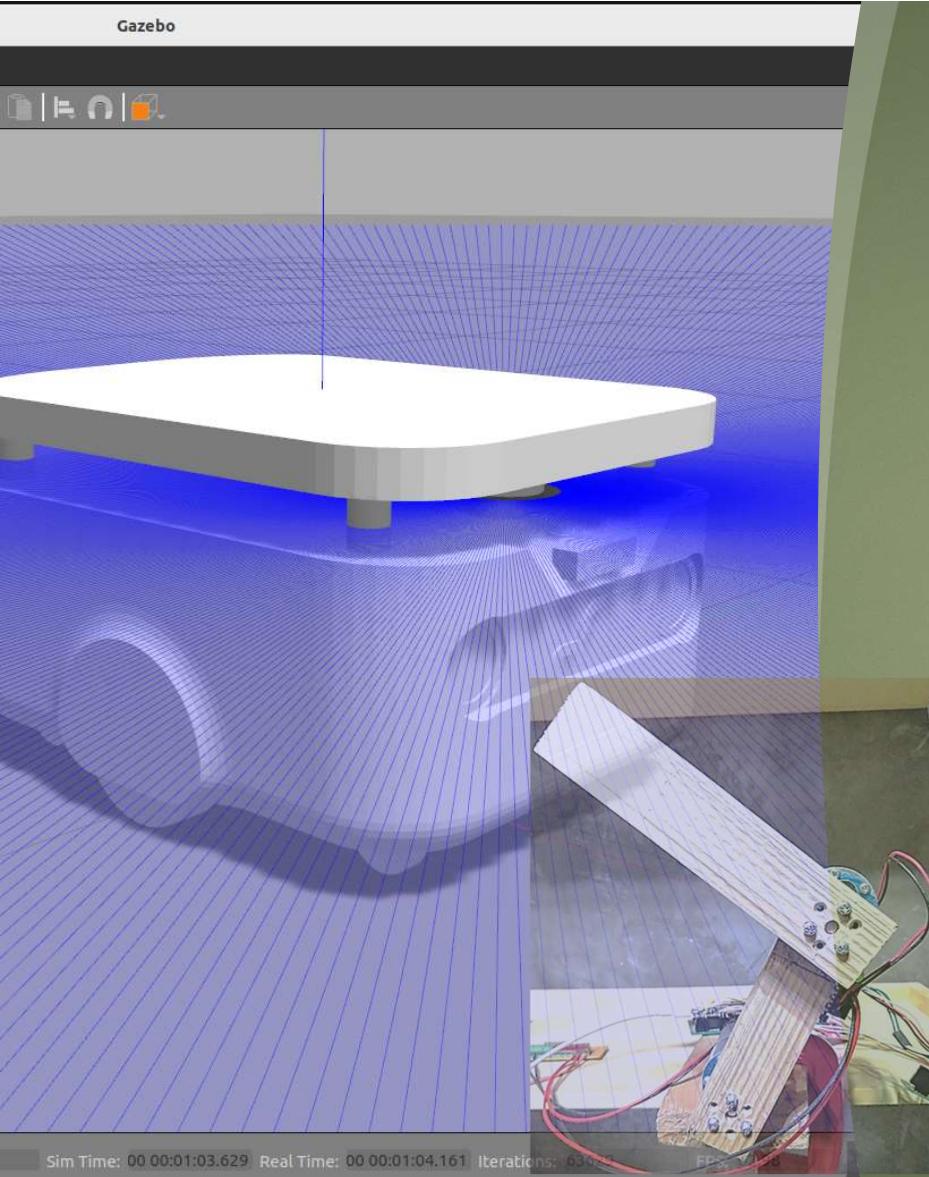
8,2.6095!  
and Tu

ทดสอบการสื่อสารข้อมูลกับอุปกรณ์ IoT โวที



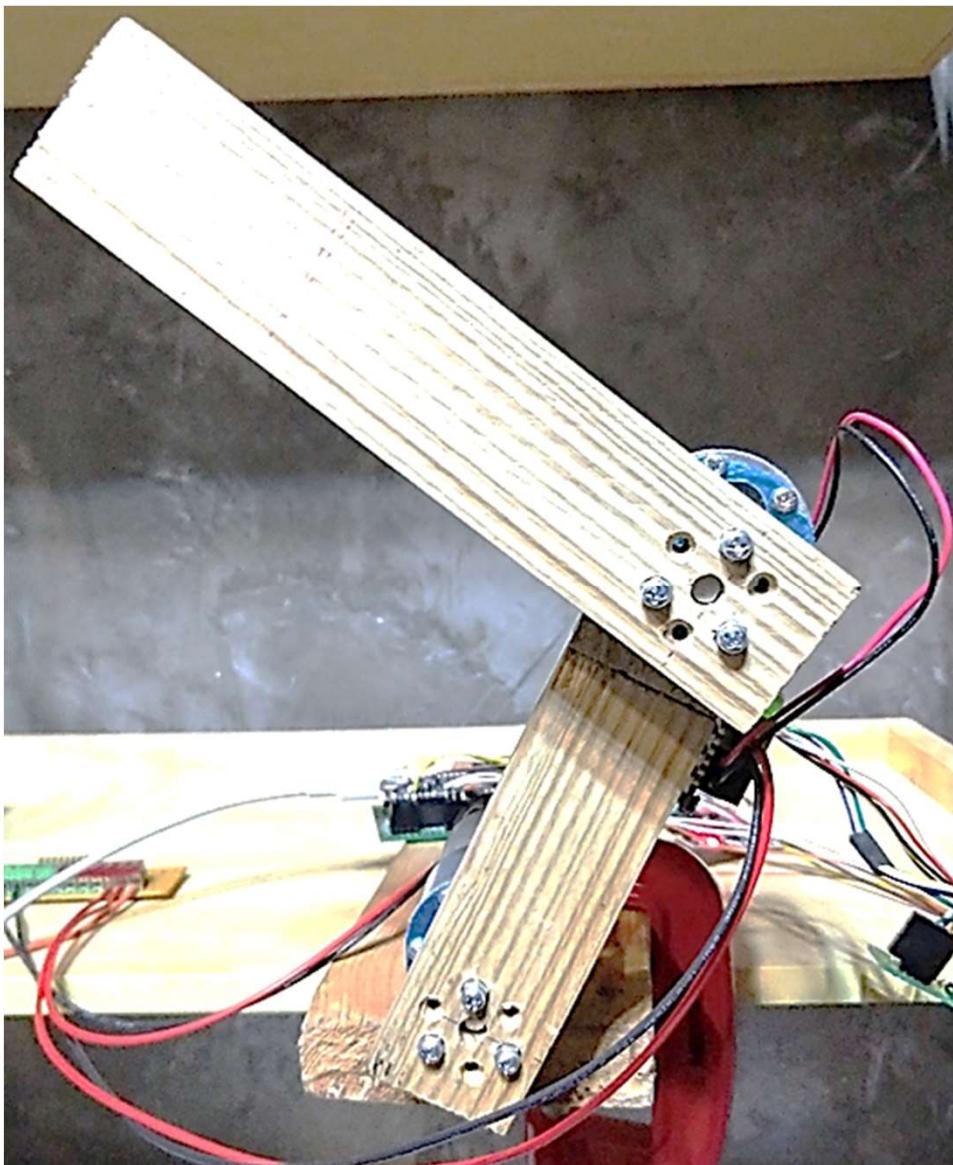
# ພ້ມນາຕົວຄວບຄຸມຫຸ້ນຍິນຕີ

## ROBOT CONTROLLER DEVELOPMENT



## หัวข้อการสัมมนาเรื่องตัว ควบคุมหุ่นยนต์

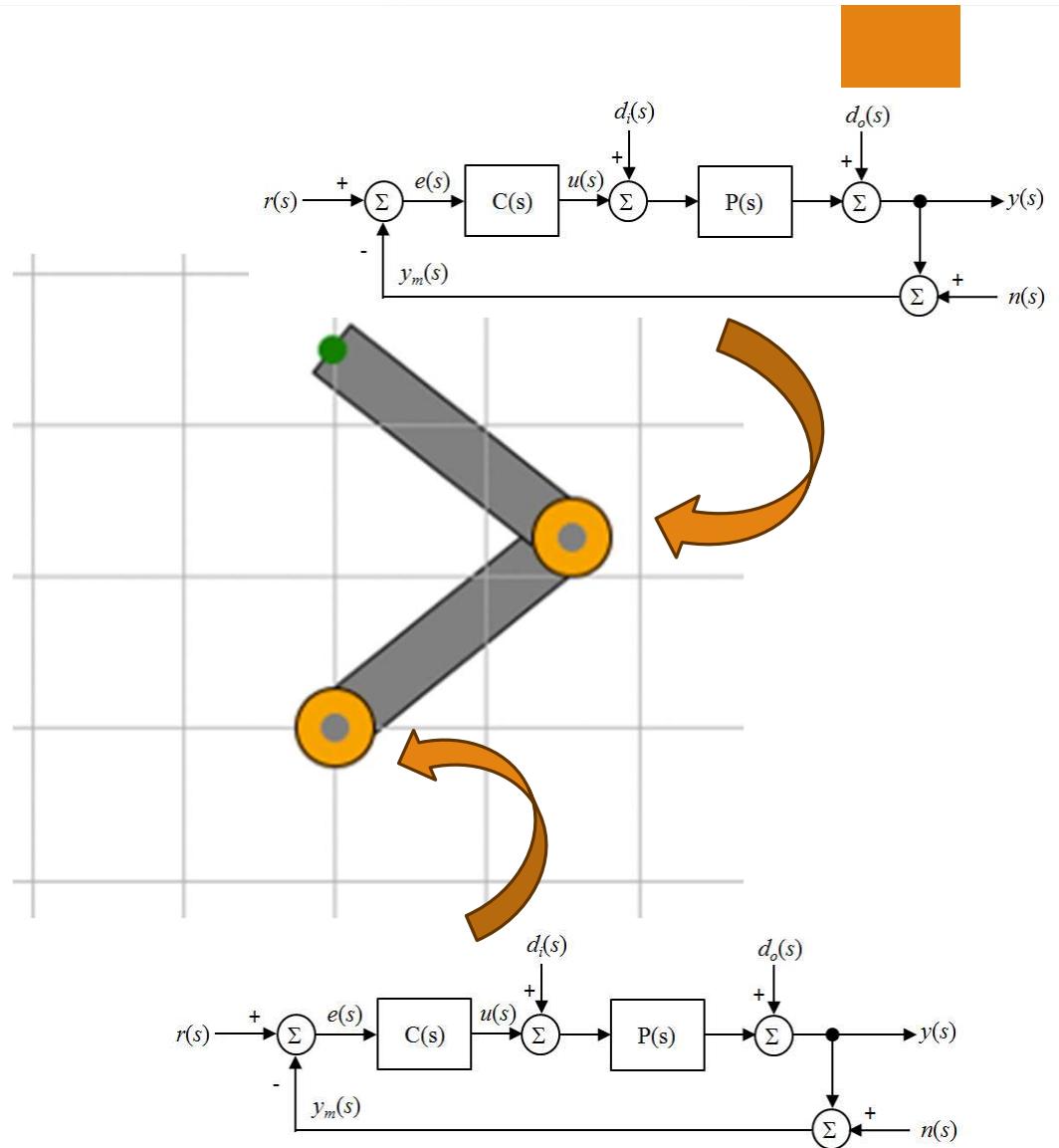
- ▶ ตัวควบคุมแขนกล 2 ก้านต่อ (2-link manipulator)
  - ▶ พลวัตของแขนกล 2 ก้านต่อ การควบคุมข้อต่ออิสระ (independent joint control)
  - ▶ ตัวควบคุมไม่เป็นเชิงเส้น (nonlinear controllers)
  - ▶ จลนศาสตร์ข้างหน้าและผกผัน (forward & inverse kinematics)
- ▶ ตัวควบคุมหุ่นยนต์เคลื่อนที่ (mobile robots)
  - ▶ หุ่นยนต์สองล้อขับเคลื่อนโดยค่าส่วนต่าง (differential drive)
  - ▶ จลนศาสตร์ของหุ่นยนต์สองล้อ
  - ▶ การควบคุมและแสดงตำแหน่งผ่านไอโอที
  - ▶ การวางแผนเส้นทางการเคลื่อนที่โดยหาคำตอบสมการของเบล曼

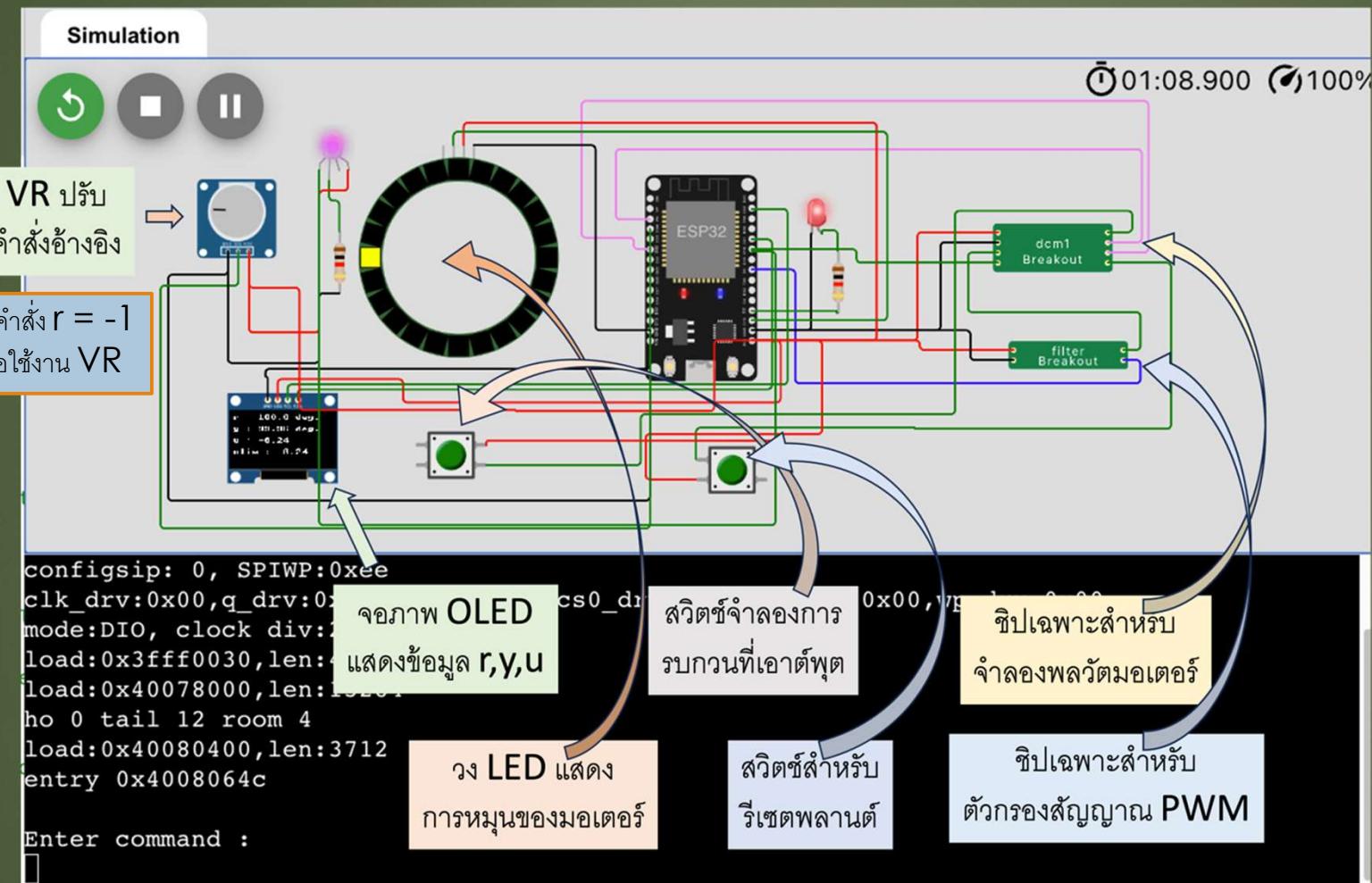


แขนกล 2 ก้านต่อ  
(2-link manipulator)

## การควบคุมข้อต่ออิสระ (independent joint control)

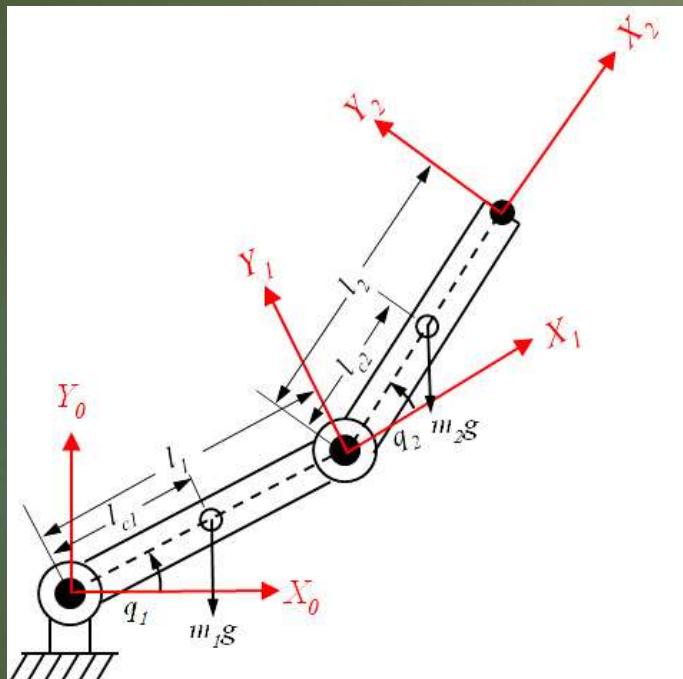
- ▶ พิจารณาแต่ละข้อต่อเป็นอิสระจากกัน
- ▶ ใช้การควบคุมป้อนกลับเชิงเดี่ยน SISO สำหรับแต่ละข้อต่อ
- ▶ ไม่เดลแรงที่ส่งผ่านระหว่างข้อต่อเป็นการรบกวน (disturbance)





# การควบคุมข้อต่ออิสระ (independent joint control)

## ພລຄາສຕຣ໌ແບນກດ 2 ກໍານຕ່ອ



$$g_1 = \frac{\partial P}{\partial q_1} = (m_1 l_{c1} + m_2 l_1) g \cos q_1 + m_2 l_{c2} g \cos(q_1 + q_2) \quad \text{gravity}$$

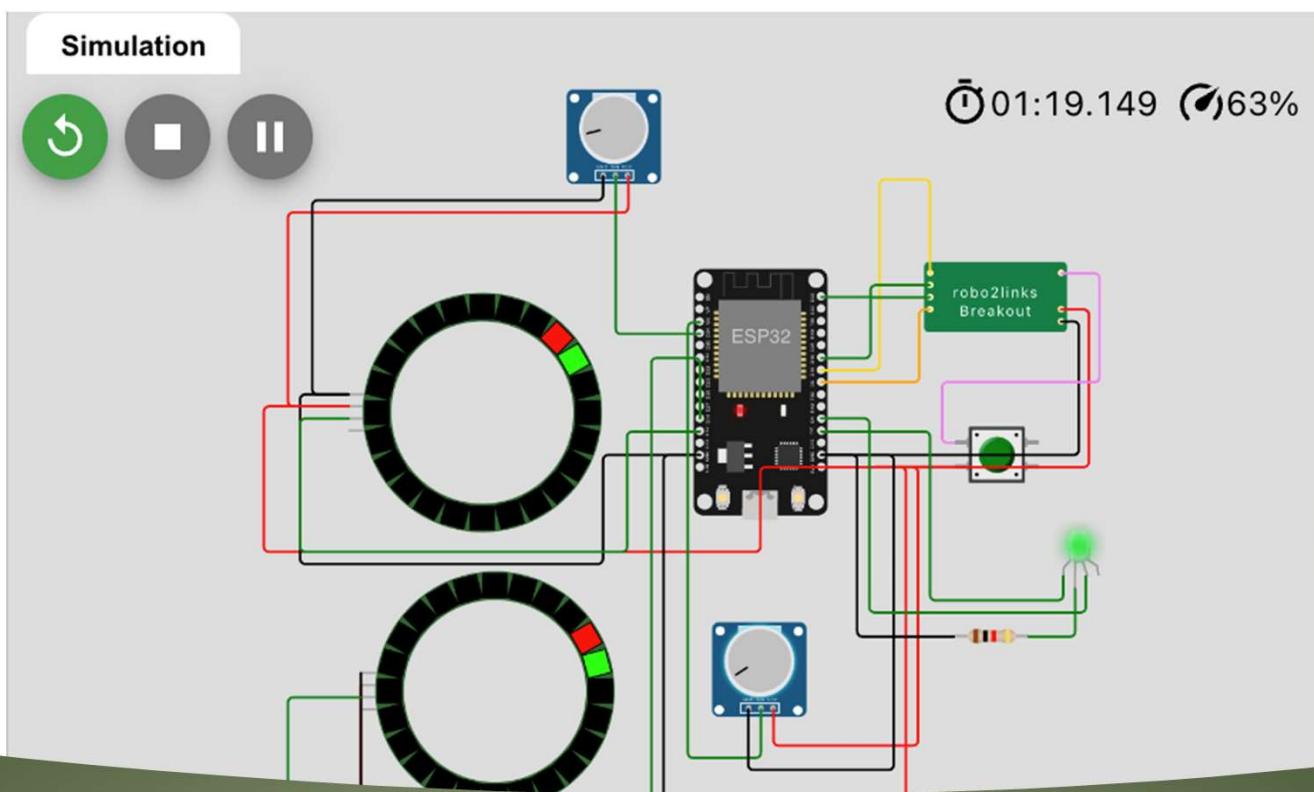
$$d_{11}\ddot{q}_1 + d_{12}\ddot{q}_2 + c_{121}\dot{q}_1\dot{q}_2 + c_{211}\dot{q}_2\dot{q}_1 + c_{221}\dot{q}_2^2 + g_1 = \tau_1 \\ d_{21}\ddot{q}_1 + d_{22}\ddot{q}_2 + c_{112}\dot{q}_1^2 + g_2 = \tau_2$$

$$d_{11} = m_1 l_{c1}^2 + m_2 (l_1^2 + l_{c2}^2 + 2l_1 l_{c2} \cos q_2) + I_1 + I_2 \\ d_{12} = d_{21} = m_2 (l_{c2}^2 + l_1 l_{c2} \cos q_2) + I_2 \\ d_{22} = m_2 l_{c2}^2 + I_2 \quad \text{inertia}$$

$$c_{121} = c_{211} = \frac{1}{2} \frac{\partial d_{11}}{\partial q_2} = -m_2 l_1 l_{c2} \sin q_2 = \xi \\ c_{221} = \frac{\partial d_{12}}{\partial q_2} - \frac{1}{2} \frac{\partial d_{22}}{\partial q_1} = \xi \\ c_{112} = \frac{\partial d_{21}}{\partial q_1} - \frac{1}{2} \frac{\partial d_{11}}{\partial q_2} = -\xi \quad \text{Coriolis}$$

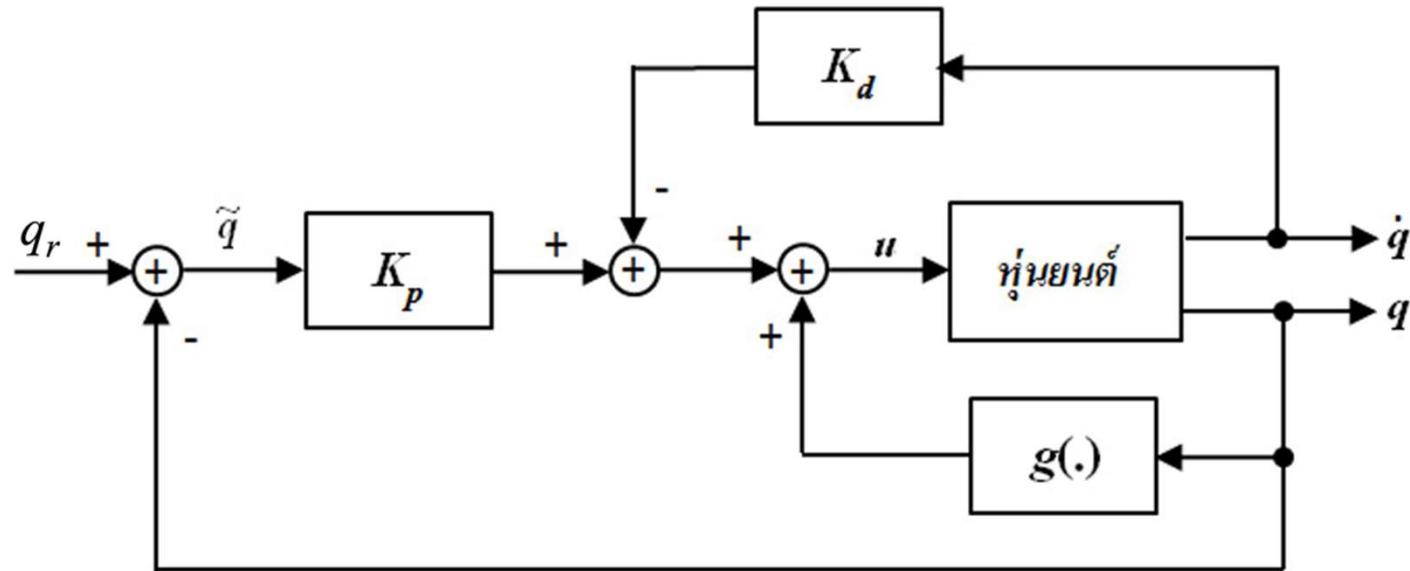
$$C = \begin{bmatrix} \xi \dot{q}_2 & \xi(\dot{q}_1 + \dot{q}_2) \\ -\xi \dot{q}_1 & 0 \end{bmatrix}$$

$$g_2 = \frac{\partial P}{\partial q_2} = m_2 l_{c2} g \cos(q_1 + q_2)$$



ผังวงจรการจำลองบน Wokwi

<https://wokwi.com/projects/389313368946834433>



$$u = g(q) + K_p \tilde{q} - K_d \dot{q}$$

ตัวควบคุม PD ที่มีการชดเชยแรงโน้มถ่วง (PDG)

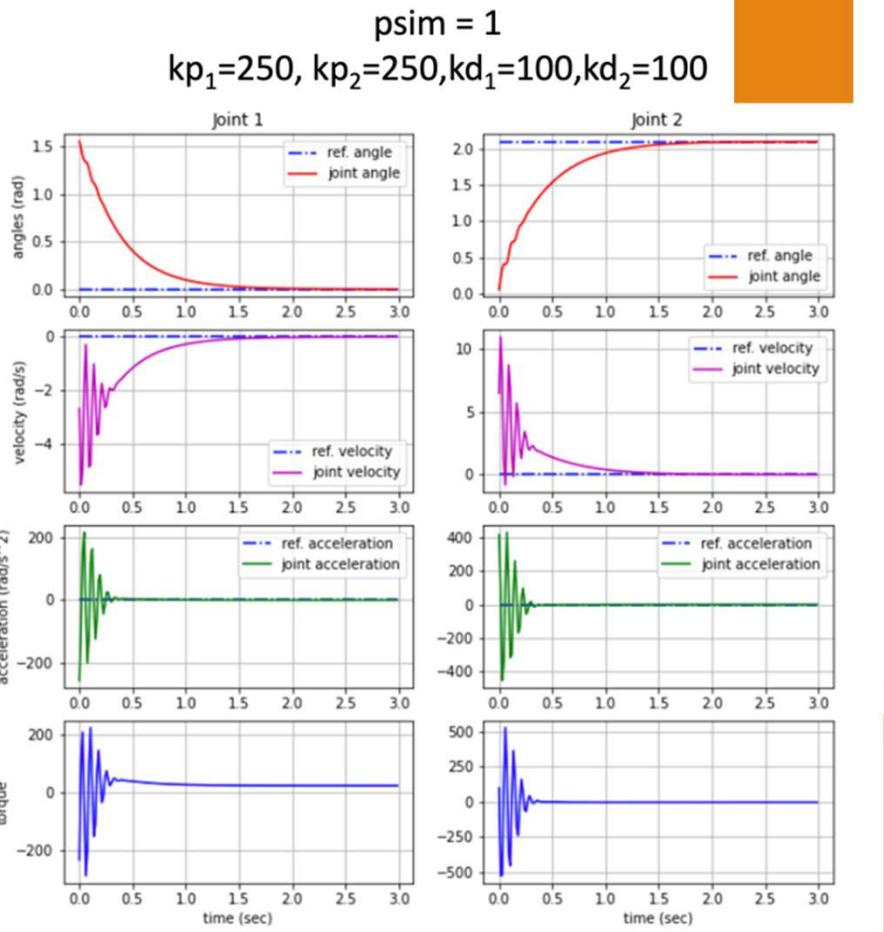
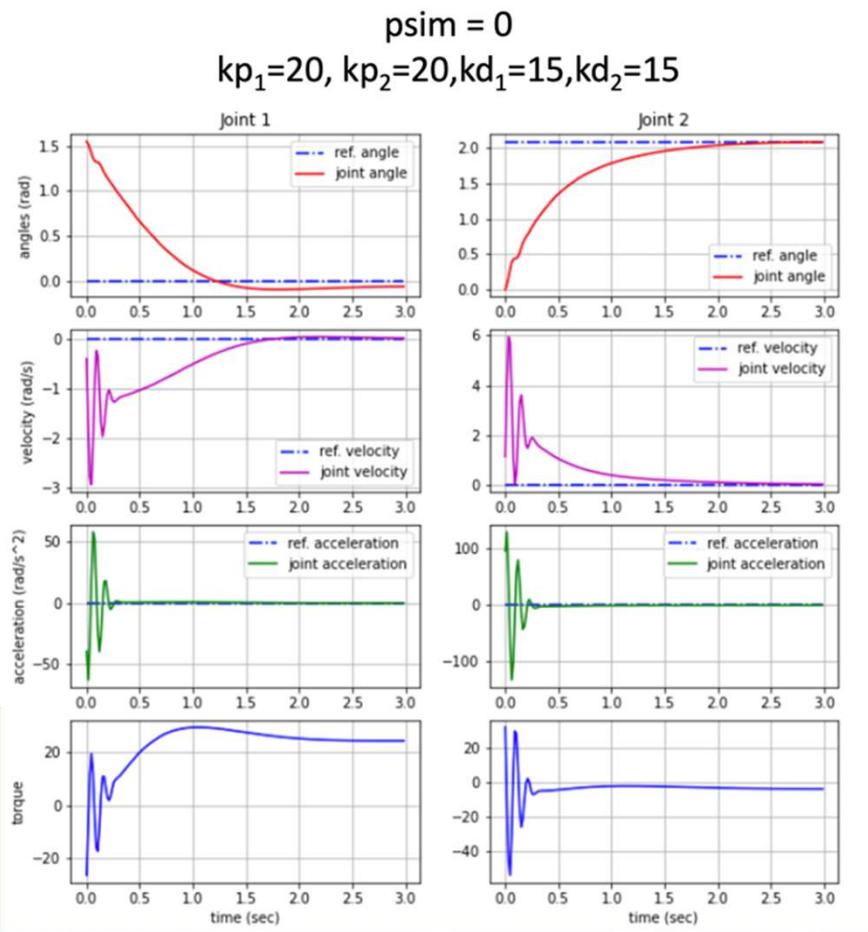
เพิ่มค่าผิดพลาดจากการ โมเดลเพื่อสอดคล้องกับความเป็นจริง

```
# add modeling error  
m1_e = m1*(1+et*(random.random() - 0.5))  
l1_e = l1*(1+et*(random.random() - 0.5))  
lc1_e = 0.5*l1_e # link c.g'  
l1_e = l1*(1+et*(random.random() - 0.5))
```

```
m2_e = m2*(1+et*(random.random() - 0.5))  
l2_e = l2*(1+et*(random.random() - 0.5))  
lc2_e = 0.5*l2_e # link c.g'  
l2_e = l2*(1+et*(random.random() - 0.5))
```

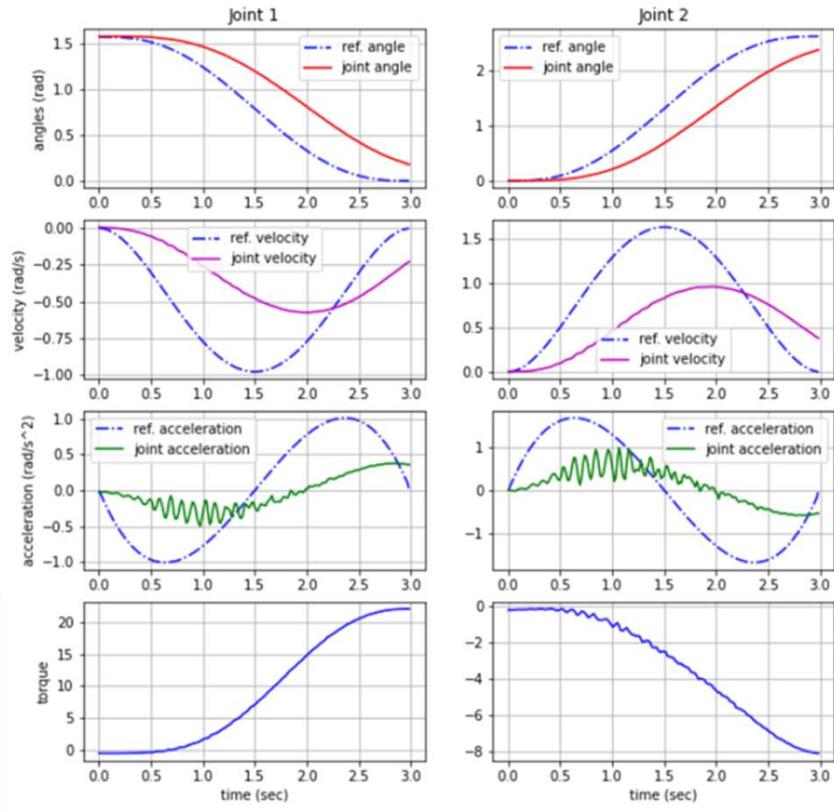
```
def PD_grav():
    global tau1, tau2, q1e, q2e
    cos12 = math.cos(q1+q2)
    q1e = q1r - q1
    q2e = q2r - q2
    g1 = ((m1_e*lc1_e + m2_e*l1_e)*g*math.cos(q1) +
           m2_e*lc2_e*g*cos12)
    g2 = m2_e*lc2_e*g*cos12
    tau1 = kp1_pdg*q1e - kd1_pdg*qd1 + g1
    tau2 = kp2_pdg*q2e - kd2_pdg*qd2 + g2
```

ตัวควบคุม PD ที่มีการชดเชยแรงโน้มถ่วง (PDG)

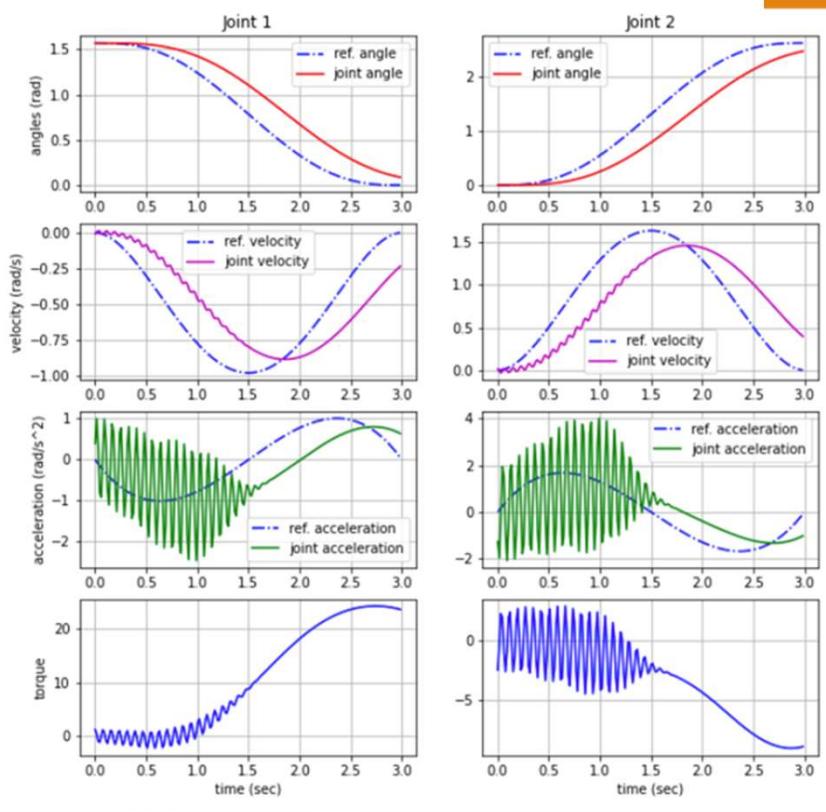


ผลตอบสนองขั้นบัน្ត ได้จากตัวควบคุม PDG

$p_{sim} = 0$   
 $k_{p1}=20, k_{p2}=20, k_{d1}=15, k_{d2}=15$



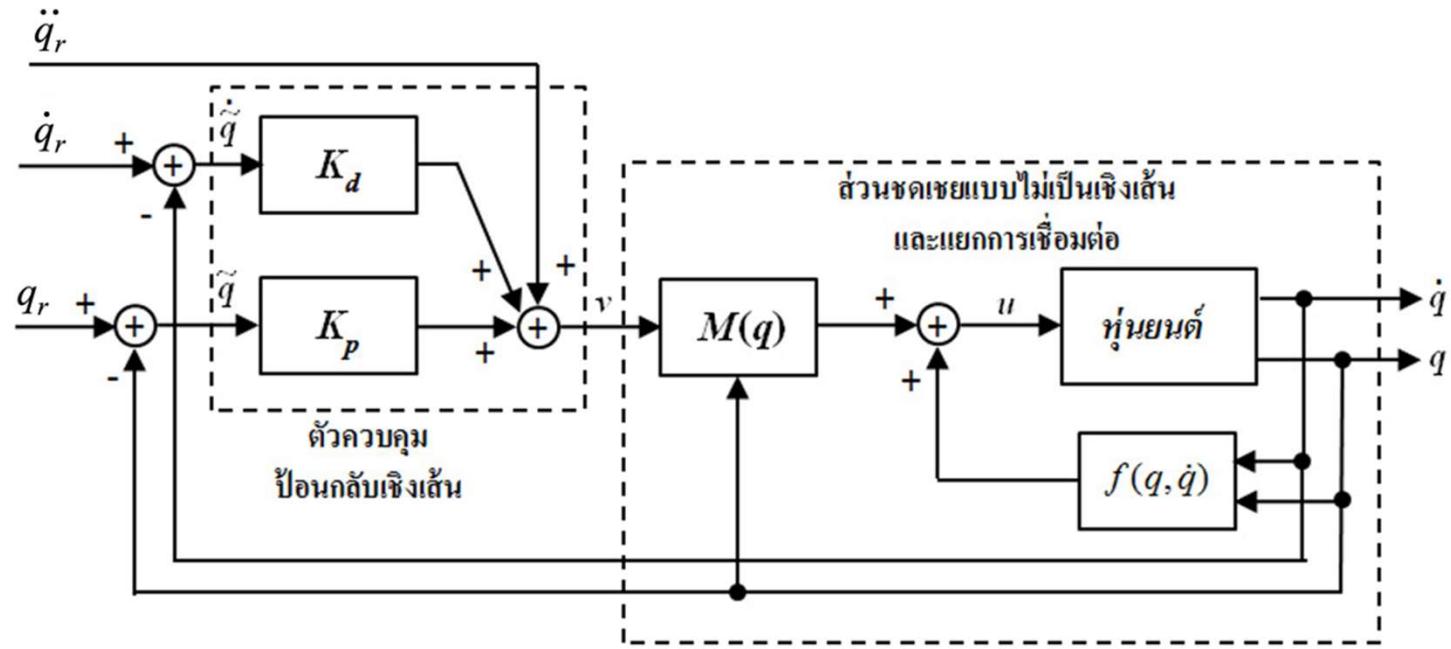
$p_{sim} = 1$   
 $k_{p1}=250, k_{p2}=250, k_{d1}=100, k_{d2}=100$



ผลการตามรอยแนววิถีแบบ qpoly จากตัวควบคุม PDG

ข้อสังเกตจาก  
ผลการตามรอยของ  
ตัวควบคุม PDG

- ผลการตามรอยค่าตัวแหน่งทำได้ในระดับหนึ่ง ยังไม่เรียกว่าดีมาก
- ผลการเปรียบเทียบความเร็วและความเร่งยังมีความแตกต่างพอสมควร ทั้งนี้ตัวควบคุม PDG ไม่มีการใช้ข้อมูลคำสั่งความเร็วและความเร่งทั้งในวงป้อนกลับหรือป้อนข้างหน้า
- สามารถใช้อัตราขยายสูงขึ้นในการจำลองบนโปรแกรมไมโครไฟฟอนทางด้านขวา ทำให้ค่าแตกต่างการตามรอยน้อยลง แต่จะเห็นการแกว่งของความเร่งและแรงบิดมากขึ้น ซึ่งเป็นสิ่งที่ต้องพิจารณาเมื่อสร้างระบบควบคุมหุ่นยนต์จริง เพราะการสั่นอาจมีผลเสียกับโครงสร้างเชิงกล อย่างไรก็ตามการแกว่งจากการจำลองนี้อาจมีน้อยกว่าเมื่อนำไปใช้กับแขนกลจริง เพราะระบบเชิงกลจะมีลักษณะเป็นตัวกรองความถี่ต่ำผ่านทางธรรมชาติ คือไม่สามารถตอบสนองต่อการเปลี่ยนแปลงความถี่สูงมากได้



## ตัวควบคุมพลวัตผกผัน (inverse dynamics)

ดูรายละเอียดการอนุพัทธ์จากหนังสือในหัวข้อ 8.1.8

```

def inverse_dynamics():
    global tau1, tau2, q1e, q2e, b
    cos12 = math.cos(q1+q2)
    cos2 = math.cos(q2)

    q1e = q1r - q1 # joint angle error
    q2e = q2r - q2
    qde1 = qd1r - qd1 # joint velocity error
    qde2 = qd2r - qd2

    # inertia matrix element
    d11e = (m1_e*lc1_e**2 +
             m2_e*(l1_e**2 + lc2_e**2 + 2*l1_e*lc2_e*cos2)
             + l1_e + l2_e)
    d12e = m2_e*(lc2_e**2+l1_e*lc2_e*cos2)+l2_e
    d21e = d12e

    # corioris matrix element
    zeta = -m2_e*l1_e*lc2_e*math.sin(q2)
    c11 = zeta*qd2
    c12 = zeta*(qd1+qd2)
    c21 = -zeta*qd1
    # c22 = 0

    # gravity term
    g1 = ((m1_e*lc1_e + m2_e*l1_e)*g*math.cos(q1) +
           m2_e*lc2_e*g*cos12)
    g2 = m2_e*lc2_e*g*cos12

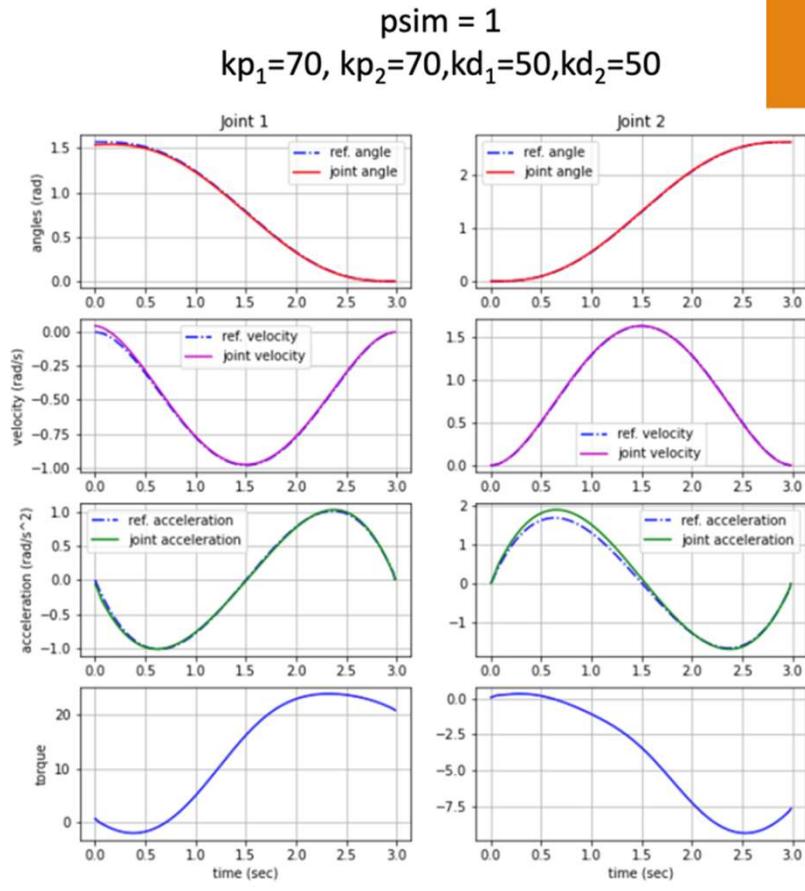
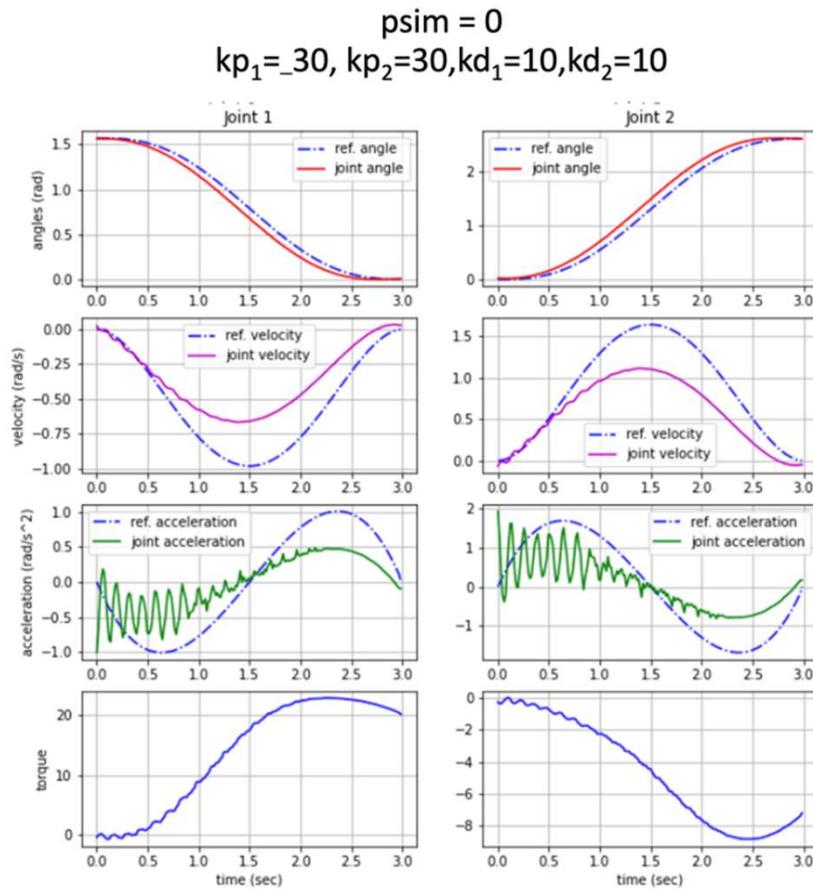
    # controller output computation
    v1 = kp1_invd*q1e + kd1_invd*qde1 + qdd1r
    v2 = kp2_invd*q2e + kd2_invd*qde2 + qdd2r

    tau1 = d11e*v1 + d12e*v2 + (c11+b)*qd1 + c12*qd2 + g1
    tau2 = d21e*v1 + d22e*v2 + c21*qd1 + b*qd2 + g2

```

$$d22e = m2_e * lc2_e^{**2} + l2_e \# constant$$

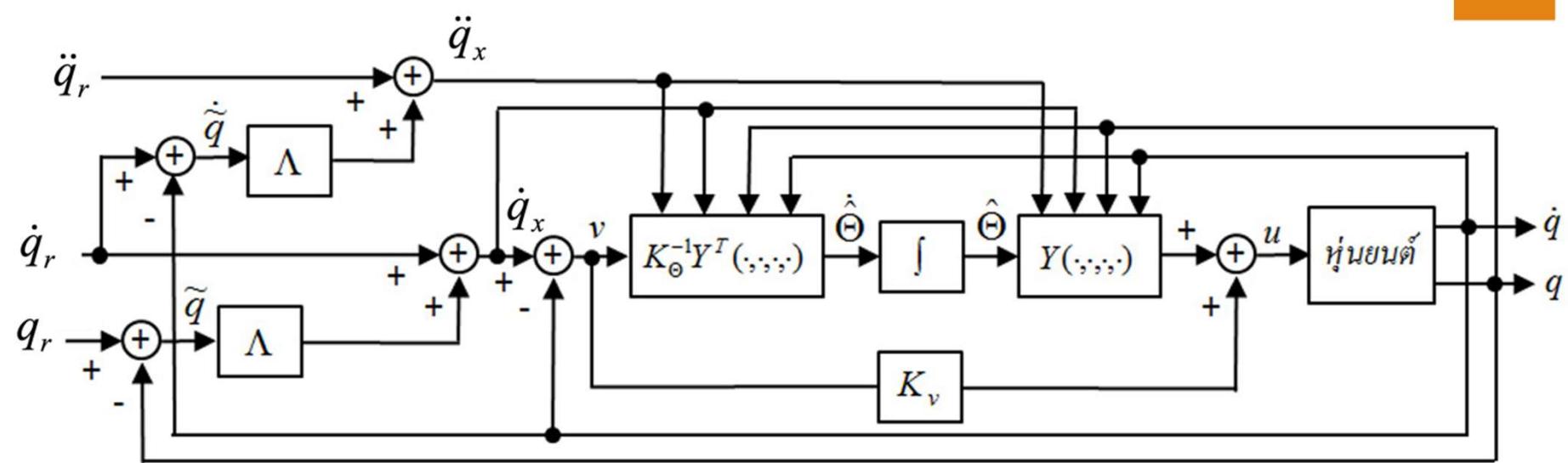
ฟังก์ชันตัวควบคุม  
ผลวัตผลพัฒนา



ผลการตามรอยแนววิถีแบบ qpoly จากตัวควบคุมพลวัตผกผัน

ข้อสังเกตจาก  
ผลการตามรอยของ  
ตัวควบคุมพลวัตพกผน

- จากแผนภาพการควบคุมในรูปที่ 8.8 ตัวควบคุมพลวัตพกผนใช้ข้อมูลคำสั่งความเร็วจากตัวกำหนดแนวโน้มเปรียบเทียบกับความเร็วจริงของแขนกล ในขณะที่ใช้คำสั่งความเร่งในรูปแบบการป้อนข้างหน้า มีส่วนช่วยทำให้สมรรถนะการตามรอยดีกว่าตัวควบคุม PDG
- ในแผนภาพการควบคุมไม่มีการใช้ความเร่งจริงจากเอกสารพุตของแขนกล ในทางปฏิบัติ เช่นเซอร์ที่สามารถวัดความเร่งอย่างแม่นยำอาจมีราคาสูง
- จากการเปรียบเทียบในรูปที่ 8.9 พบร่วงการจำลองบนโปรแกรมไมโครไฟฟอนให้ผลการตามรอยที่ดีกว่าการจำลองโดยใช้ชิปอย่างเด่นชัด แสดงถึงผลจากการหน่วงเวลาเมื่อตัวควบคุมบน ESP32 และพลวัตแขนกลบนชิปจำลองโดยใช้ฐานเวลาแยกจากกัน



## ត៊វគបគុមប្រើបាយ (adaptive controller)

ទូរសព្ទនៃការងារនេះផ្តល់ព័ត៌មានលម្អិតនៃការងារនេះដោយសៀវភៅ 8.1.9 – 8.1.10

```

def adaptive_control():
    global tau1, tau2, q1e, q2e, qd1e, qd2e, q1, q2, qd1,\ 
        qd2, qdd1, qdd2

    cos12 = math.cos(q1+q2)
    cos1 = math.cos(q1)
    cos2 = math.cos(q2)
    sin2 = math.sin(q2)

    q1e = q1r - q1 # joint angle error
    q2e = q2r - q2
    qd1e = qd1r - qd1 # joint velocity error
    qd2e = qd2r - qd2

    qd1x = lamda1*q1e + qd1r
    qd2x = lamda2*q2e + qd2r
    qdd1x = lamda1*qd1e + qdd1r
    qdd2x = lamda2*qd2e + qdd2r

    v1 = qd1x - qd1
    v2 = qd2x - qd2

    # create elements of Y matrix
    y11 = qdd1x
    y12 = (cos2*(2*qdd1x + qdd2x) - sin2*(qd1*qd2x +
                                                qd2*qd1x + qd2*qd2x))
    y13 = qdd2x
    y14 = g*cos1
    y15 = g*cos12
    y21 = 0.0
    y22 = cos2*qdd1x + sin2*qd1*qd1x
    y23 = qdd1x + qdd2x
    y24 = 0.0
    y25 = g*cos12

```

```

# compute theta_dot vector in (8.31)
thd1_hat = (y11*v1 + y21*v2)/kth
thd2_hat = (y12*v1 + y22*v2)/kth
thd3_hat = (y13*v1 + y23*v2)/kth
thd4_hat = (y14*v1 + y24*v2)/kth
thd5_hat = (y15*v1 + y25*v2)/kth

# get theta vector from integrator objects
th1_hat = Int1.out(thd1_hat)
th2_hat = Int2.out(thd2_hat)
th3_hat = Int3.out(thd3_hat)
th4_hat = Int4.out(thd4_hat)
th5_hat = Int5.out(thd5_hat)

y1 = (y11*th1_hat + y12*th2_hat + y13*th3_hat +
      y14*th4_hat + y15*th5_hat)
y2 = (y21*th1_hat + y22*th2_hat + y23*th3_hat +
      y24*th4_hat + y25*th5_hat)

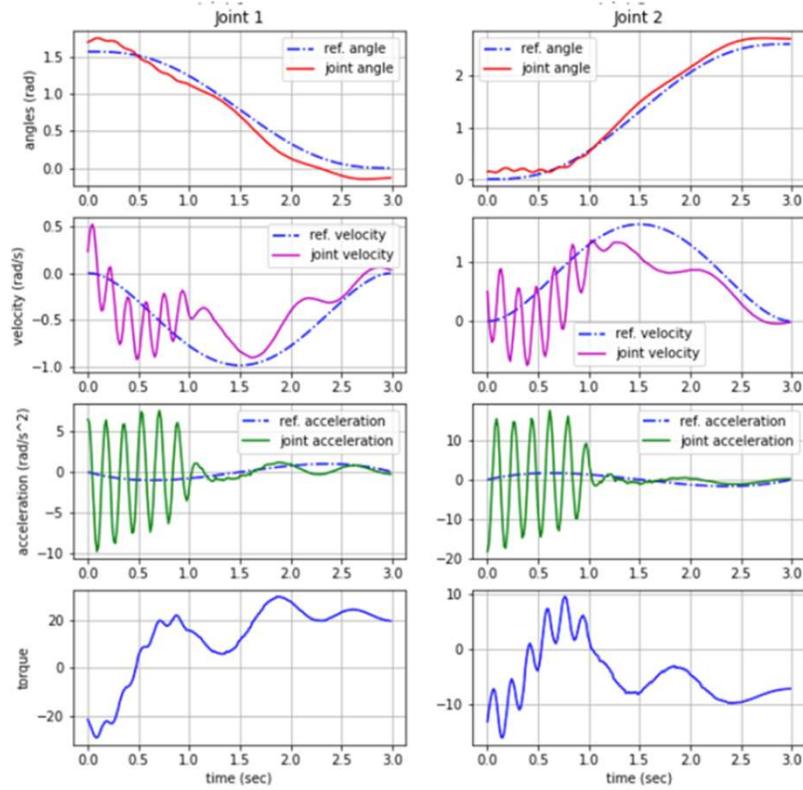
tau1 = y1 + kv1*v1
tau2 = y2 + kv2*v2

```

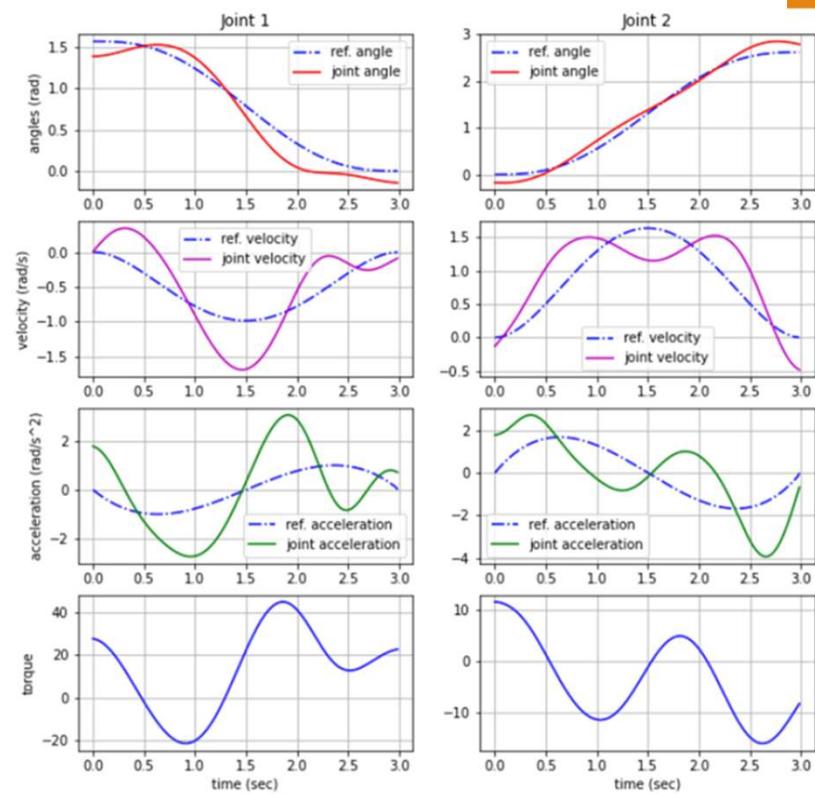
ພຶ້ມກໍ່ສັນຕິວຄວບຄຸມປັບຕົວ

$\lambda_1 = 20.0, \lambda_2 = 12.0, k_{th} = 5000.0, kv_1 = 7.5, kv_2 = 5.0$

$psim = 0$



$psim = 1$



ผลการตามรอยแนววิถีแบบ qpoly จากตัวควบคุมปรับตัว

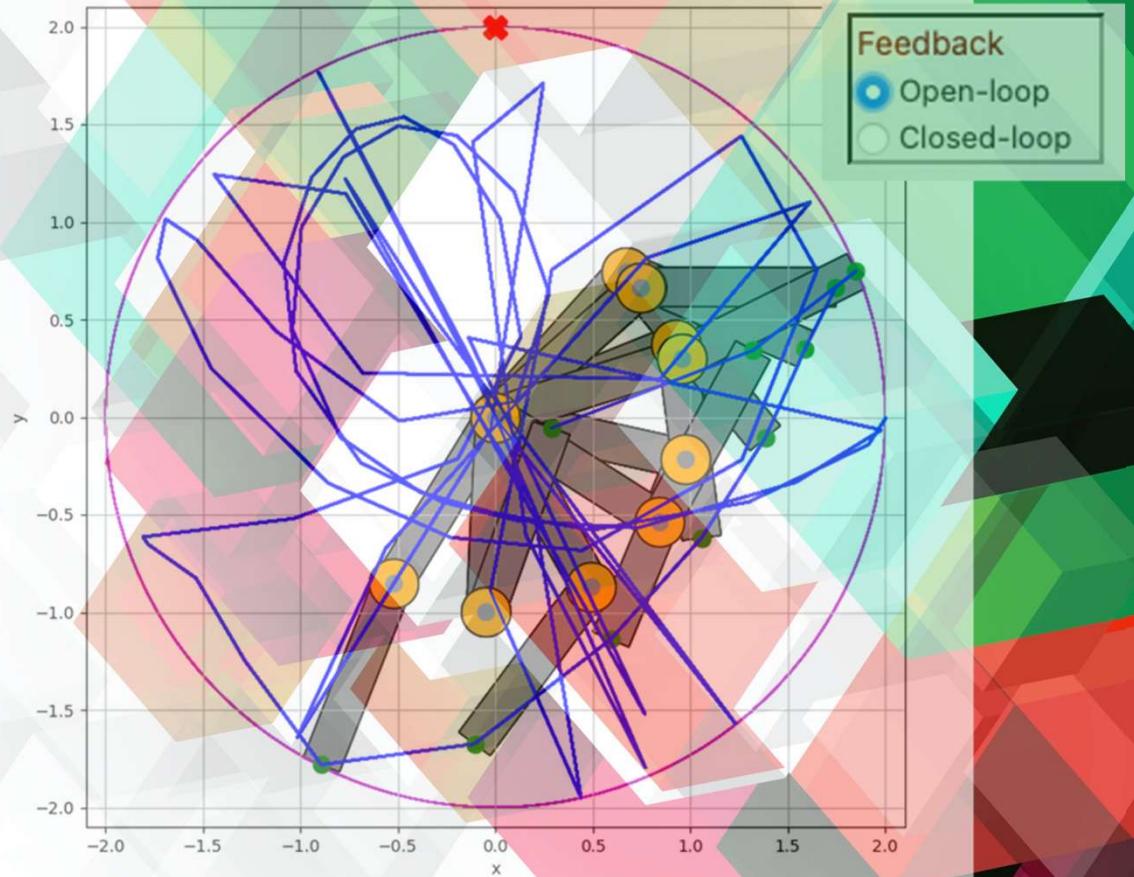
ข้อสังเกตจาก  
ผลการตามรอยของ  
ตัวควบคุมปรับตัว

- ระบบควบคุมปรับตัวมีความไวต่อพารามิเตอร์ lamda1, lamda2, kv1, kv2 ค่อนข้างมาก ไม่สามารถปรับให้มีค่าสูงโดยไม่เสียเสถียรภาพ แต่การปรับค่า kth ไม่มีผลอย่างเด่นชัด โดยการจำลองทั้ง 2 รูปแบบมีพฤติกรรมใกล้เคียงกัน
- จากผลการตามรอยค่าตำแหน่งในรูปที่ 8.11 ไม่สามารถสรุปได้ว่าการจำลองบนไมโครไฟฟอนให้ผลดีกว่าการจำลองบนชิป
- โดยรวมแล้วสมรรถนะการตามรอยด้อยกว่าตัวควบคุมพลวัตผกผันในกรณีที่ไม่เดลของแขนกลมีความแม่นยำสูง แต่ข้อได้เปรียบคือใช้การประมาณค่าพารามิเตอร์โดยไม่ต้องทราบค่าจริง เพียงกำหนดรูปแบบของสมการพลวัตเท่านั้น

# ເສຣີມພຶກໜັນໄວໂອທີ ໃຫ້ຕ້ວຄວບຄຸນຫຼຸ່ມຍິນຕີ

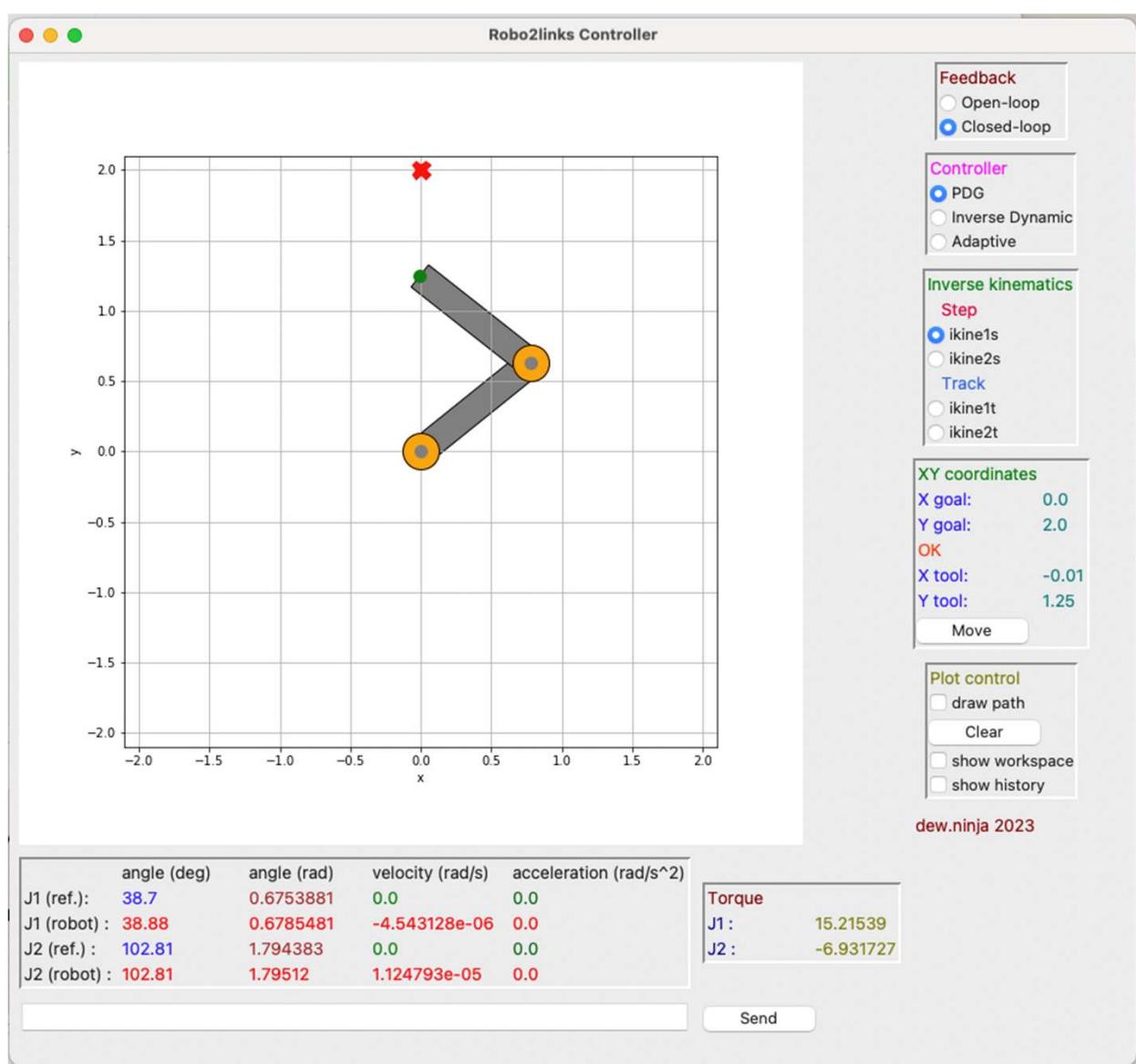
# จลนศาสตร์ของ แขนกล 2 ก้านต่อ

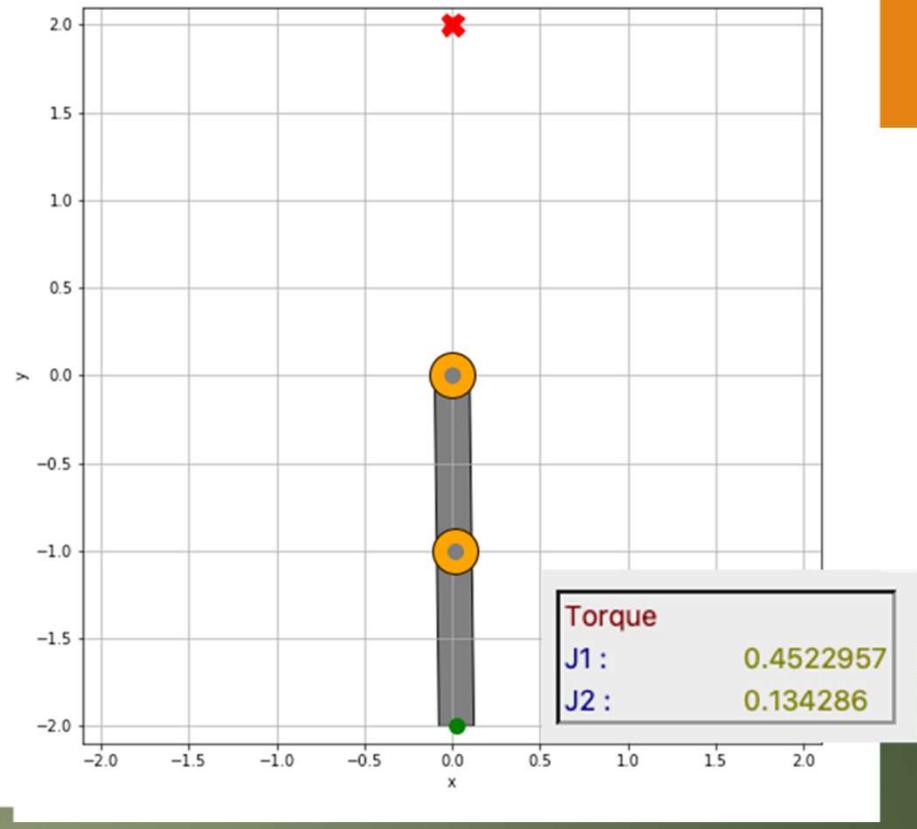
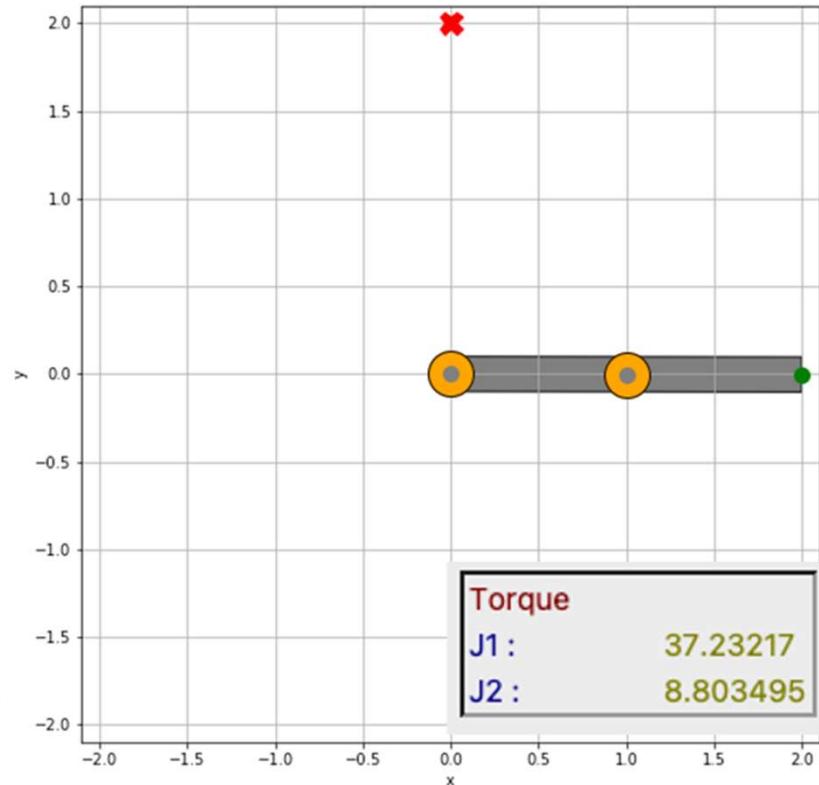
KINEMATICS OF 2-LINK  
MANIPULATOR



# r2 GUI สำหรับ ศึกษาจลนศาสตร์ ของแขนกล 2 ข้าง ด้วย

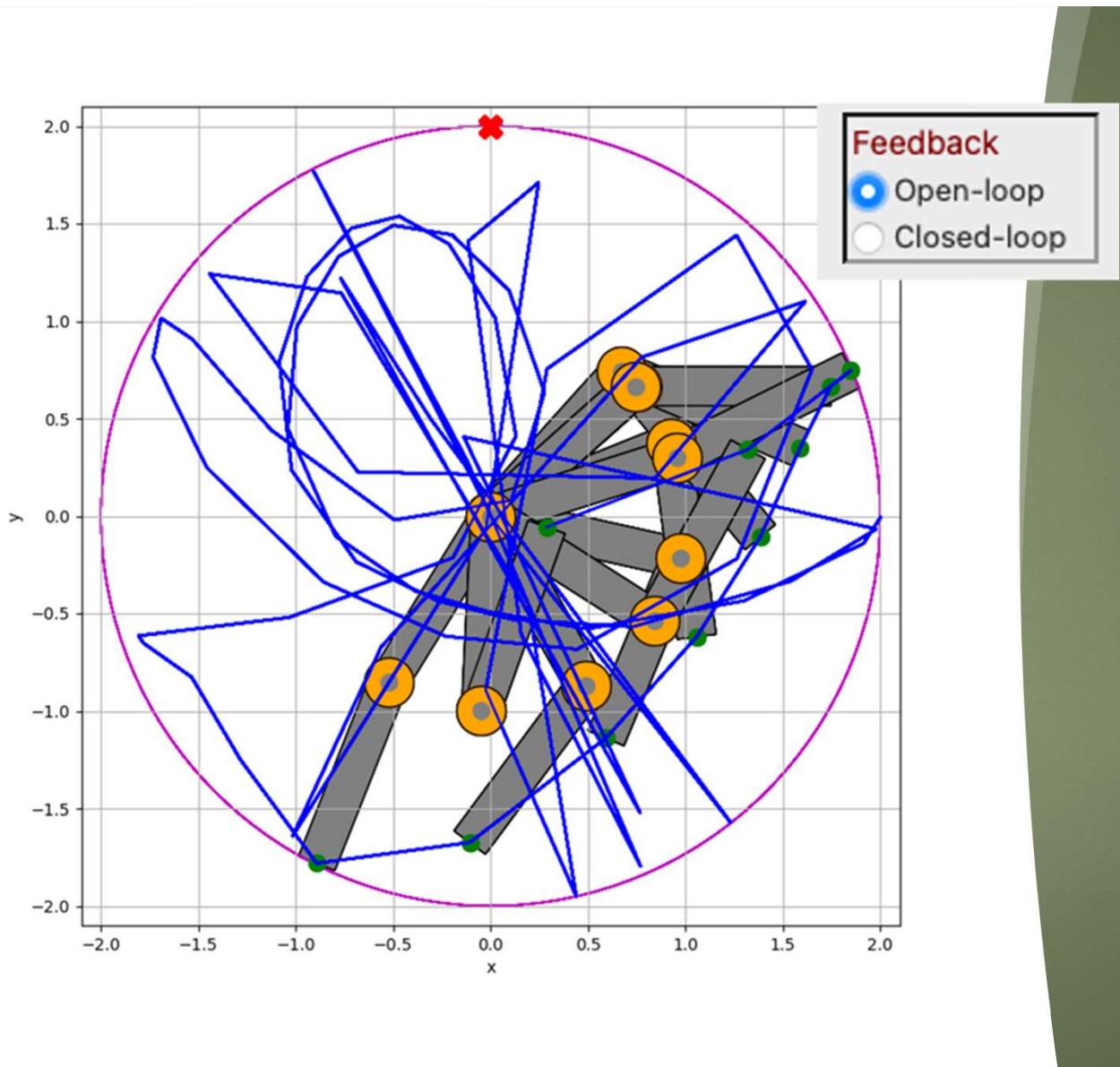
r2.ipynb





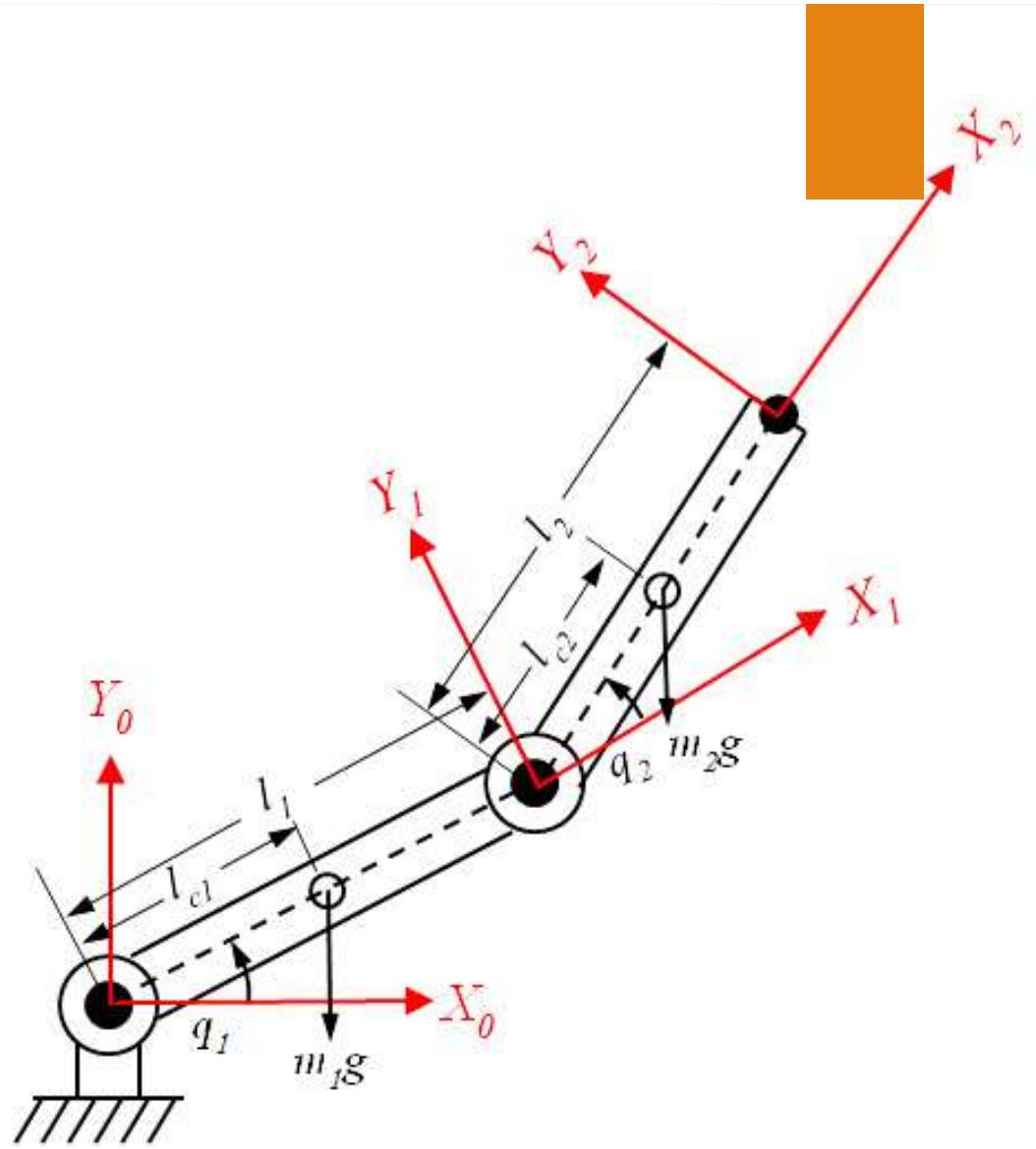
ค่าของแรงบิดข้อต่อที่ขึ้นกับโครงร่างแบบหุ่นยนต์

# พฤติกรรมของแขน กลเมื่อเปิดวงควบคุม



## จลนศาสตร์ของแขนกล 2 ก้านต่อ

- ▶ จลนศาสตร์ข้างหน้า (forward kinematics)
- ▶ จลนศาสตร์ผกผัน (inverse kinematics)



## จลนศาสตร์ข้างหน้า

- ▶ แปลงมุมของข้อต่อ  $q_1, q_2$  เป็นค่าพิกัด  $x, y$  (และทิศทาง) ของตัวทำงานส่วนปลาย
- ▶ ใช้ในเมธอด **animate()** ของ r2GUI เพื่อพล็อตโครงร่างแบบของแขนกล

$$x = l_1 c_1 + l_2 c_{12} \quad (8.34)$$

$$y = l_1 s_1 + l_2 s_{12} \quad (8.35)$$

$$c_i = \cos(q_i), s_i = \sin(q_i), c_{ij} = \cos(q_i + q_j), s_{ij} = \sin(q_i + q_j)$$

## ปัญหาจลนศาสตร์พกพัน

- ▶ ไม่มีคำตอบ (no solution)
- ▶ คำตอบเป็นได้ออย่างเดียว (unique solution)
- ▶ มีหลายคำตอบ (multiple solution)

```

def ikine(self, *args):
    # first check if solution exist
    c2 = (self.xg**2 + self.yg**2 - self.l1**2
          - self.l2**2)/(2*self.l1*self.l2)
    if abs(c2)>1.0: # no solution
        self.issolution = False
        self.ikmsg_txt.set("No solution")
    else:
        self.issolution = True
        self.iktype = self.iktype_txt.get() # ikine types
        # ikine1s or ikine1t
        if self.iktype == '0' or self.iktype == '2':
            s2 = np.sqrt(1 - c2**2)
        else: # ikine2s or ikine2t
            s2 = -np.sqrt(1 - c2**2)
        q2g = np.arctan2(s2,c2)
        k1 = self.l1 + self.l2*c2
        k2 = self.l2*s2
        q1g = np.arctan2(self.yg,self.xg)-np.arctan2(k2,k1)
        self.q1g = self.rad2deg*q1g
        self.q2g = self.rad2deg*q2g
        self.ikmsg_txt.set("q1,q2: ("+str(round(self.q1g,3)) +
                           "," + str(round(self.q2g,3)) + ")")

```

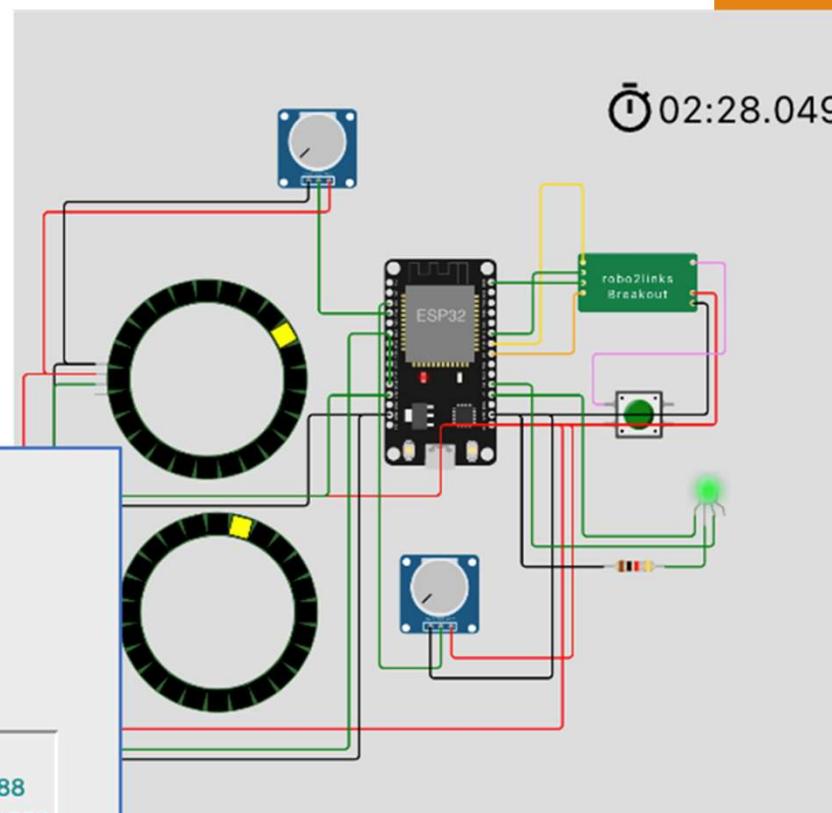
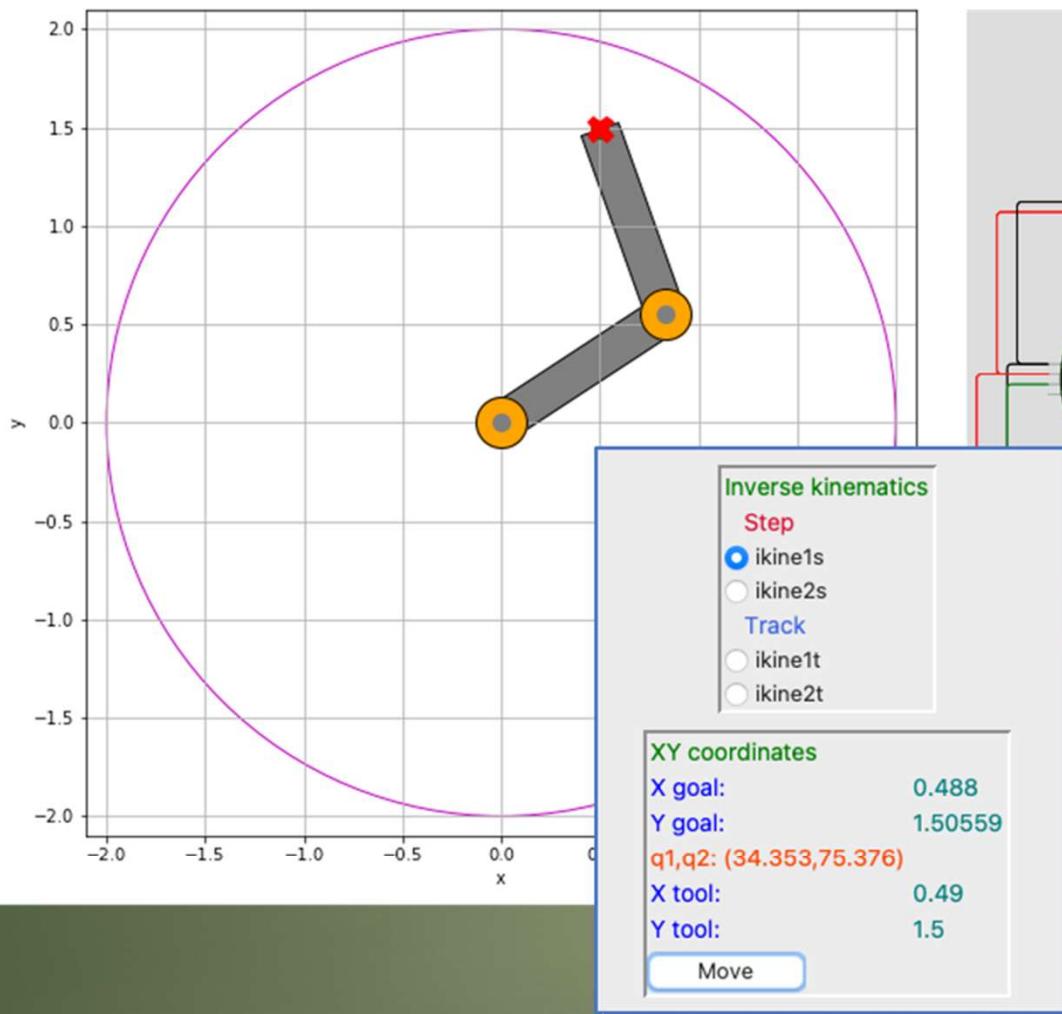
## เมธอด **ikine()**

คำนวณจลนศาสตร์

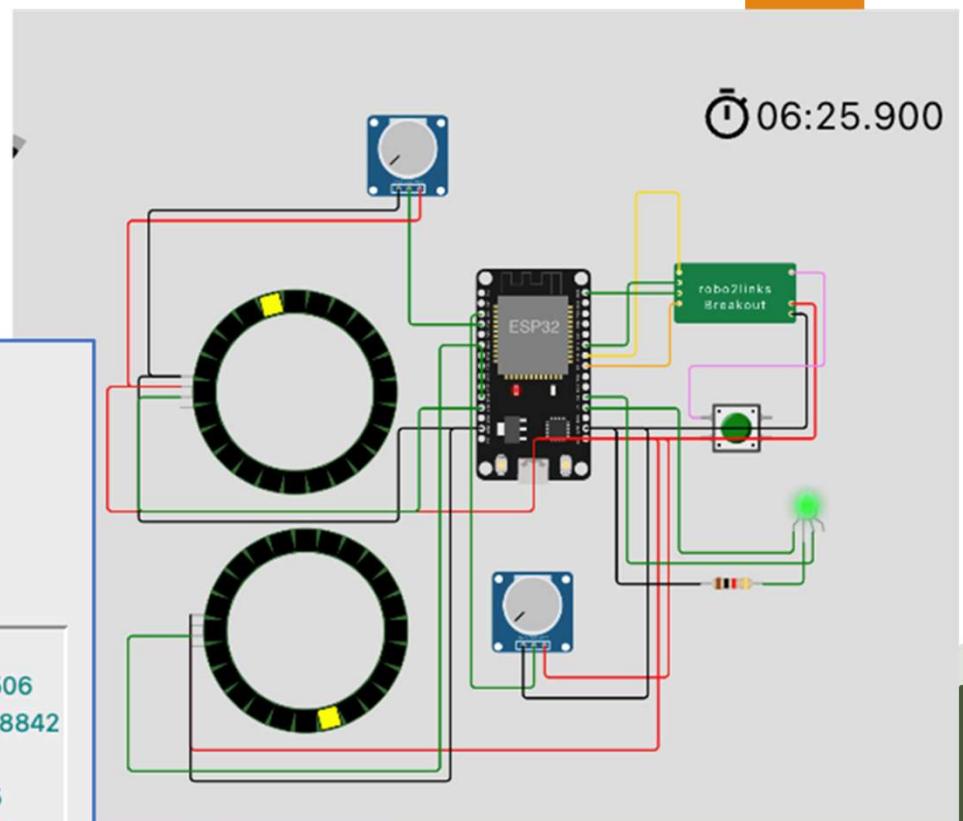
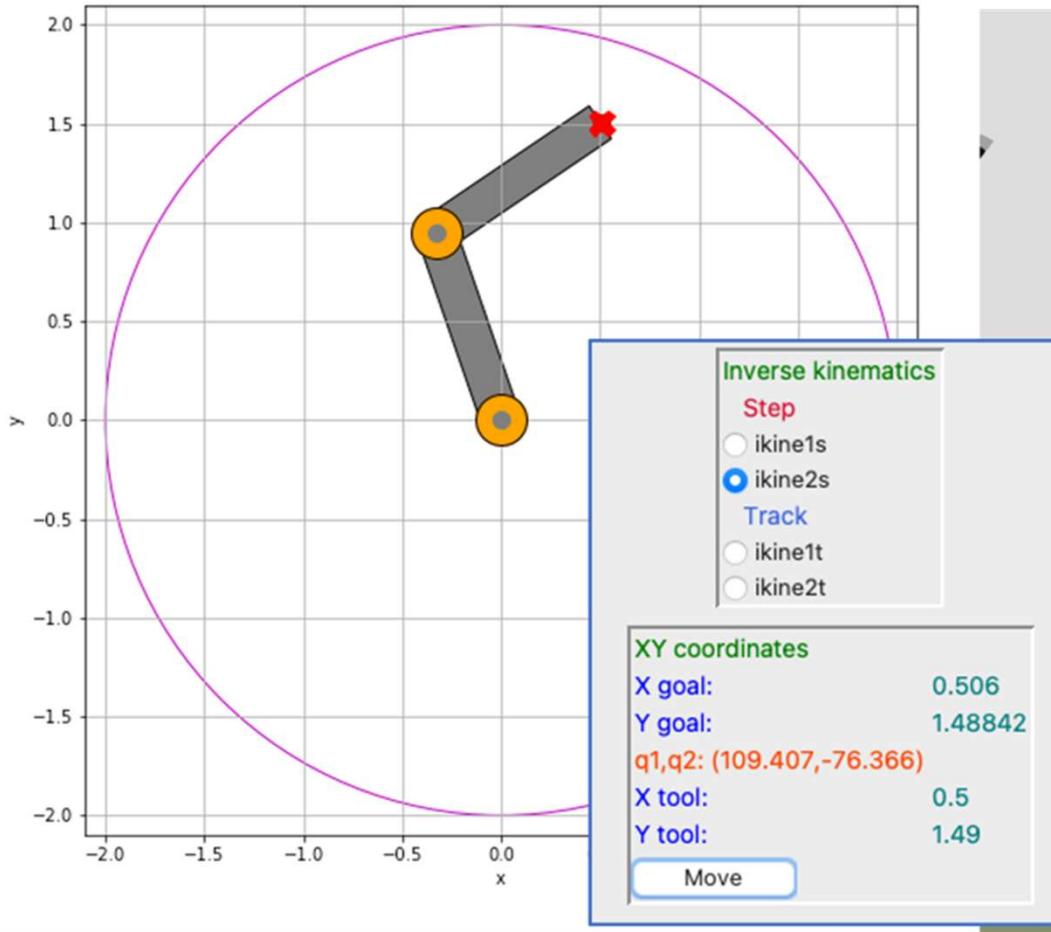
พกผันของแขนกล 2

ทำงานต่อ

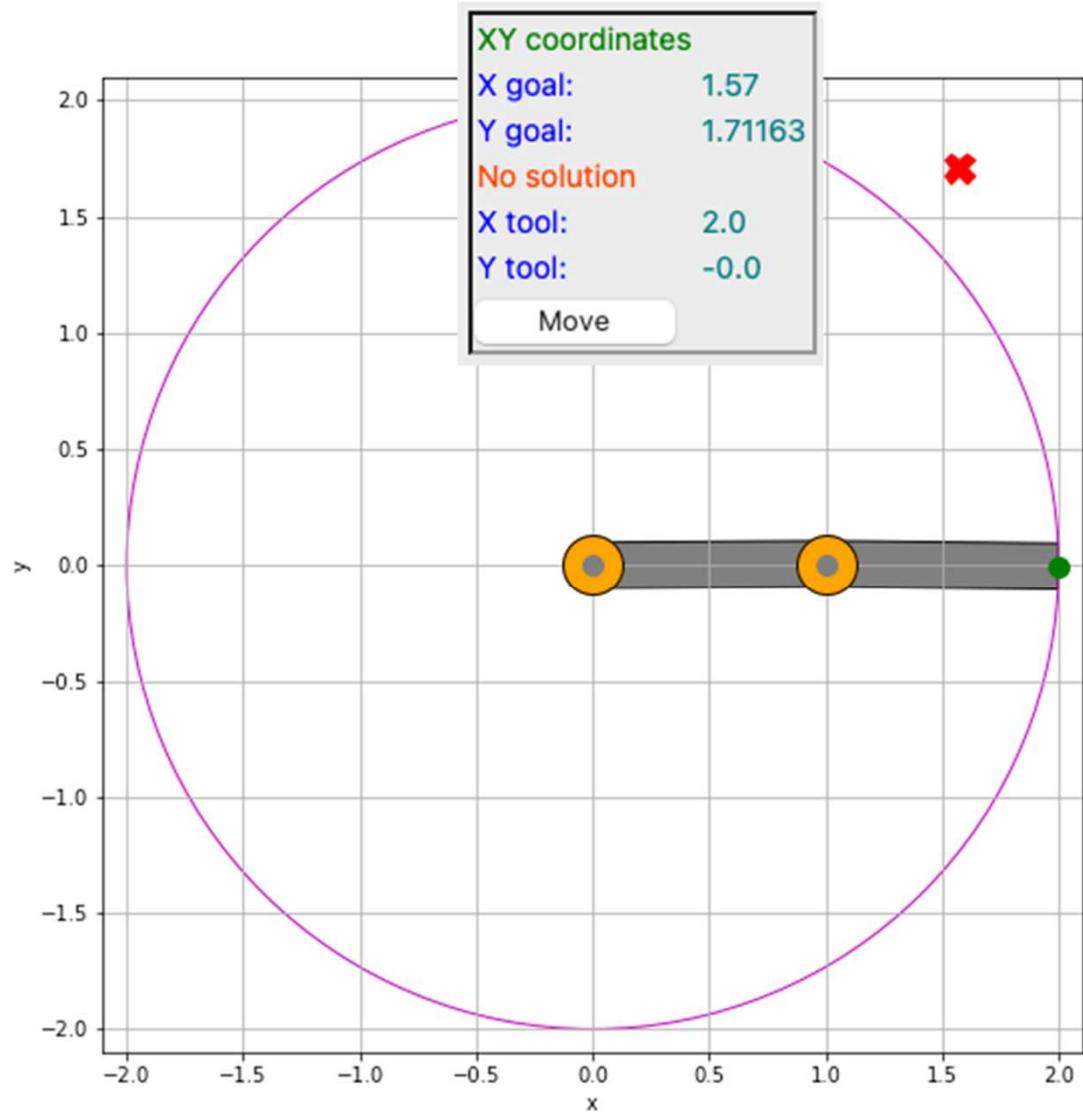
รายละเอียดในตัวอย่าง 8.8



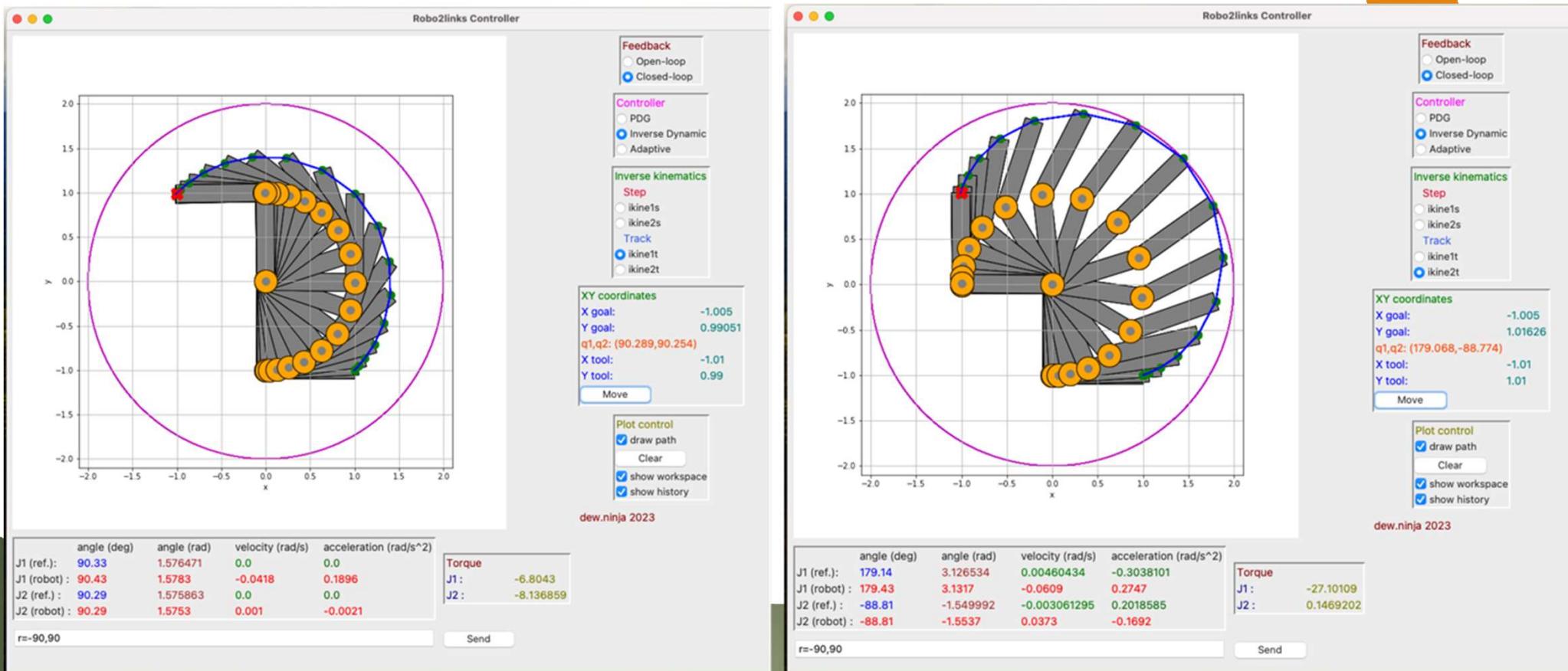
คำตอบจนศาสตร์พกผนสำหรับตัวเลือก ikine1s



คำตอบจนศาสตร์/pkg ผนึกสำหรับตัวเลือก ikine2s



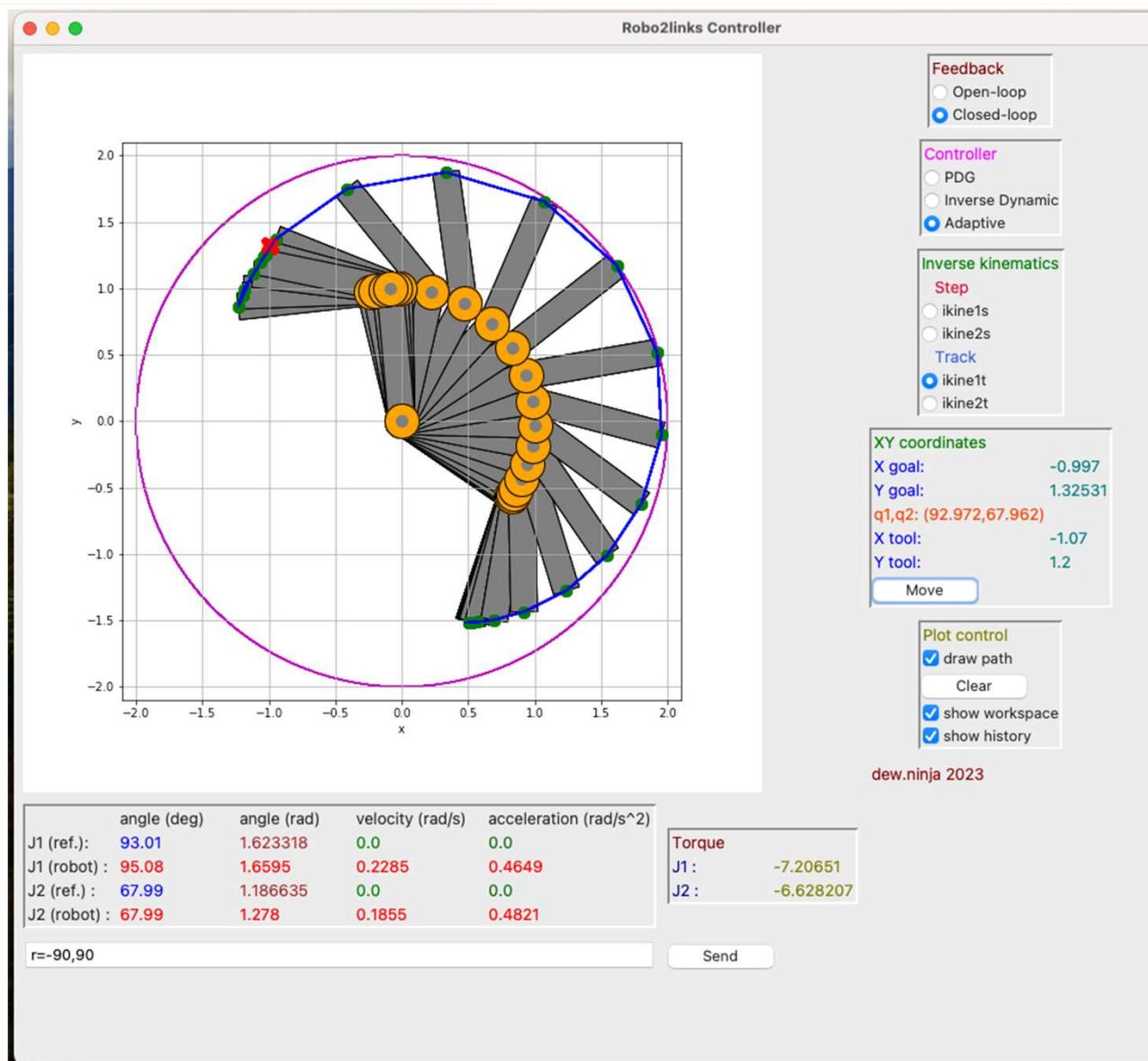
พิกัดนอกพื้นที่การ  
ทำงาน ไม่สามารถหา  
คำตอบบนศาสตร์  
พกผันได้

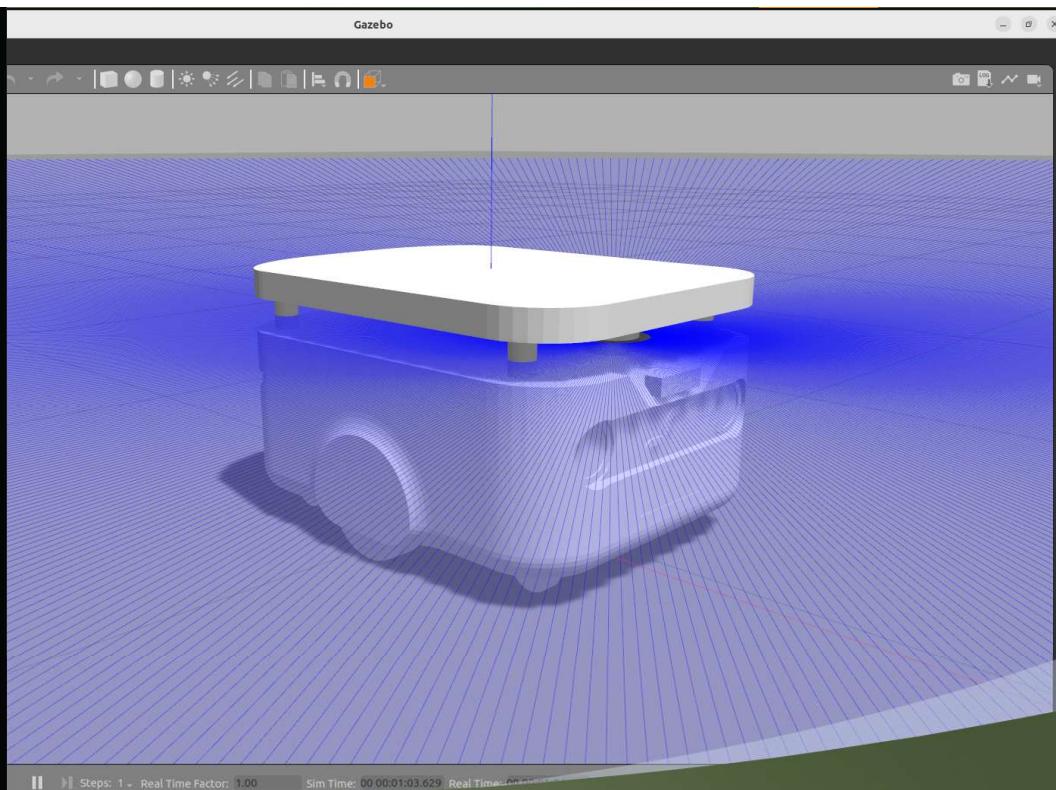
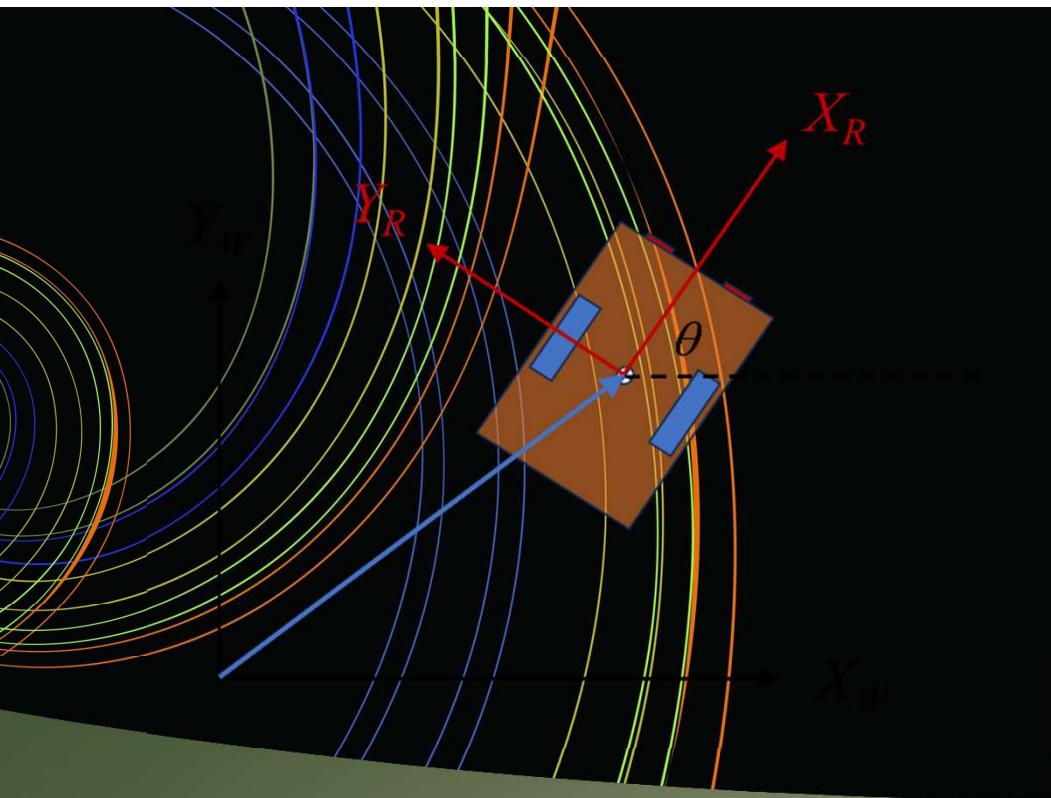


ผลการตามรอยโดยคำสั่ง ikine1t และ ikine2t

ผลการตามรอย

จากตัวควบคุม  
ปรับตัว





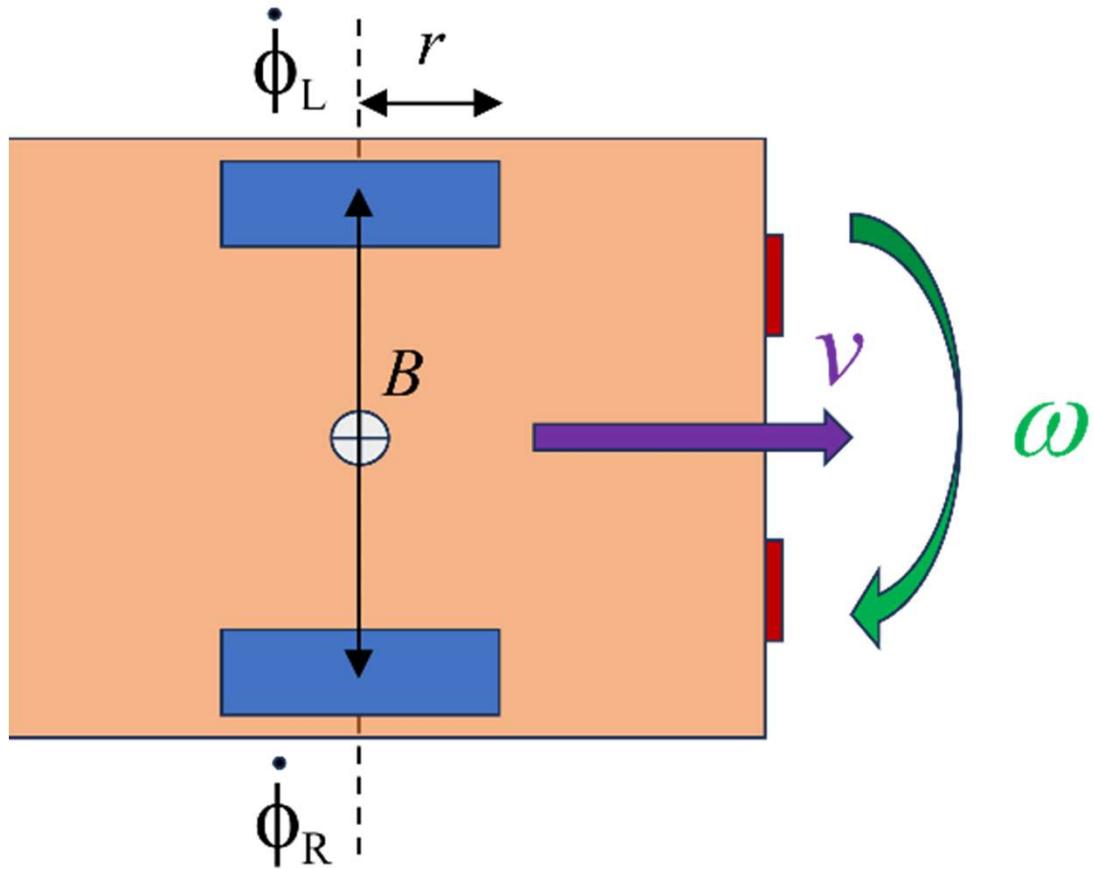
หุ่นยนต์เคลื่อนที่

MOBILE ROBOTS



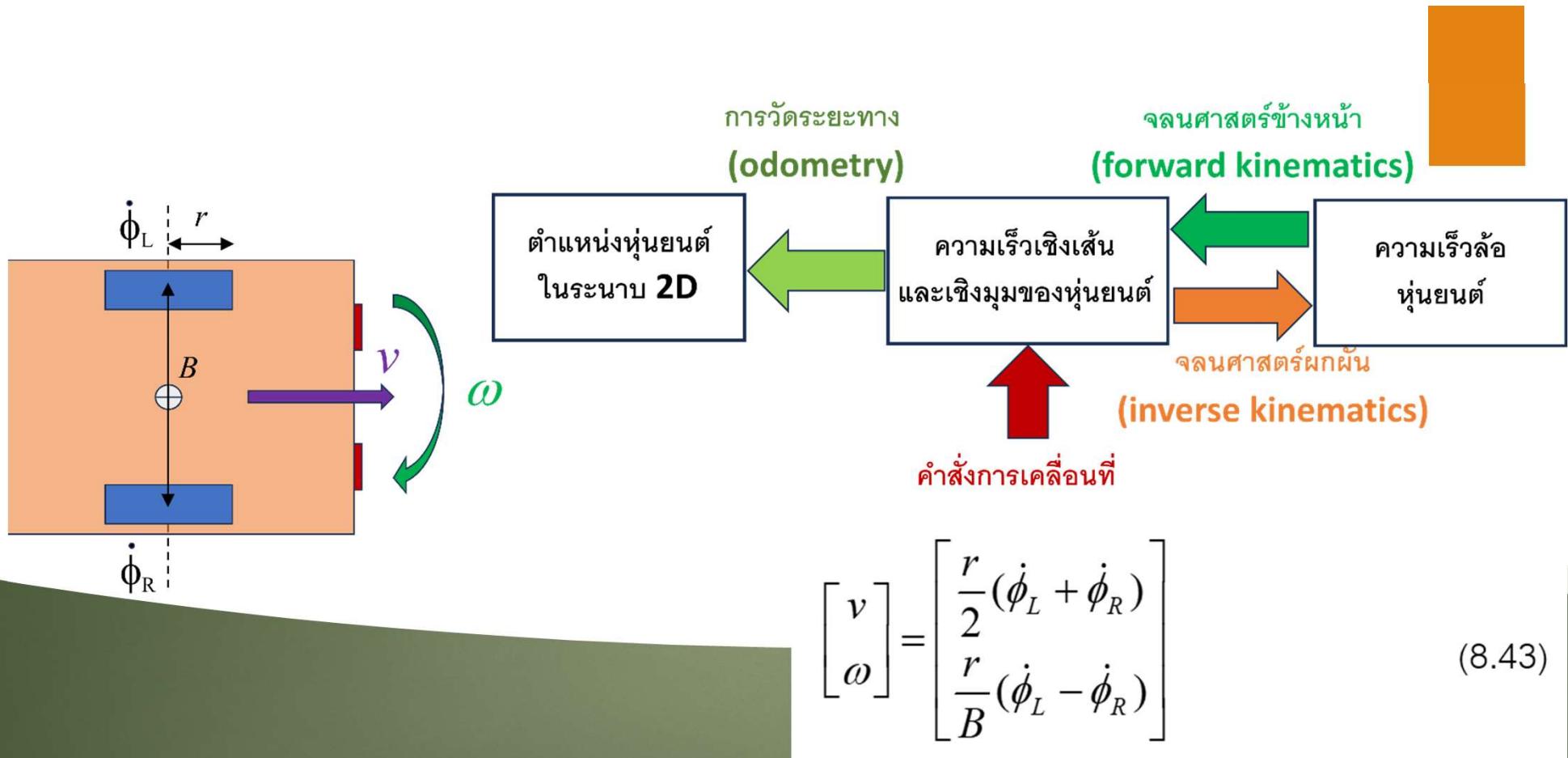
ตัวอย่างหุ่นยนต์สองล้อ<sup>ขับเคลื่อนโดยค่าส่วนต่าง</sup>  
(differential drive)

“coconut robot” by CoXsys Robotics  
<https://youtu.be/PkcSTzwtf5M?si=83zyKQb9XiYENqe7>

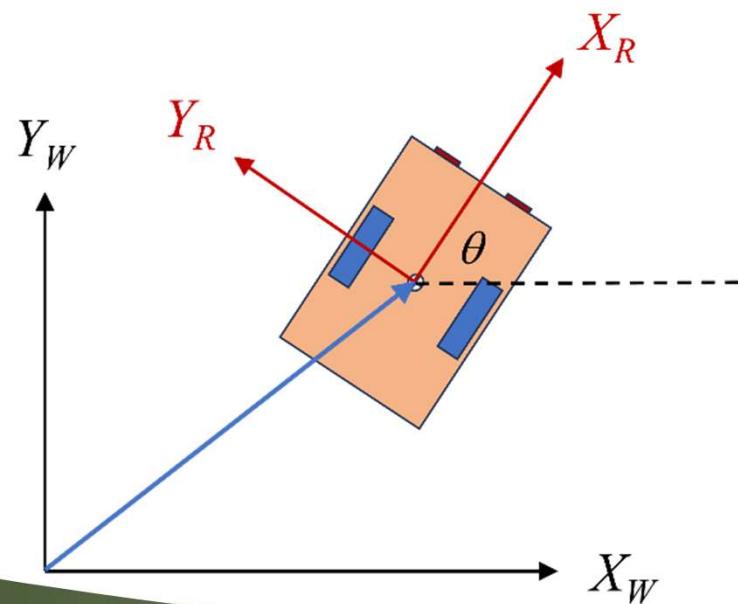


## หุ่นยนต์สัม

หุ่นยนต์สองล้อที่ใช้เป็นตัวอย่างใน  
หนังสือและสไลด์นี้มีขนาดสอดคล้องกับ  
“coconut robot” เพียงแต่ตัด  
ส่วนประกอบปลิกย่ออย่างล้อ  
casters ออกเพื่อความเข้าใจง่าย ตั้ง  
ชื่อว่าหุ่นยนต์สัมตามสีของกราฟิกที่ใช้



จlnศาสตร์ (ความเร็ว) ของหุ่นยนต์สองล้อ



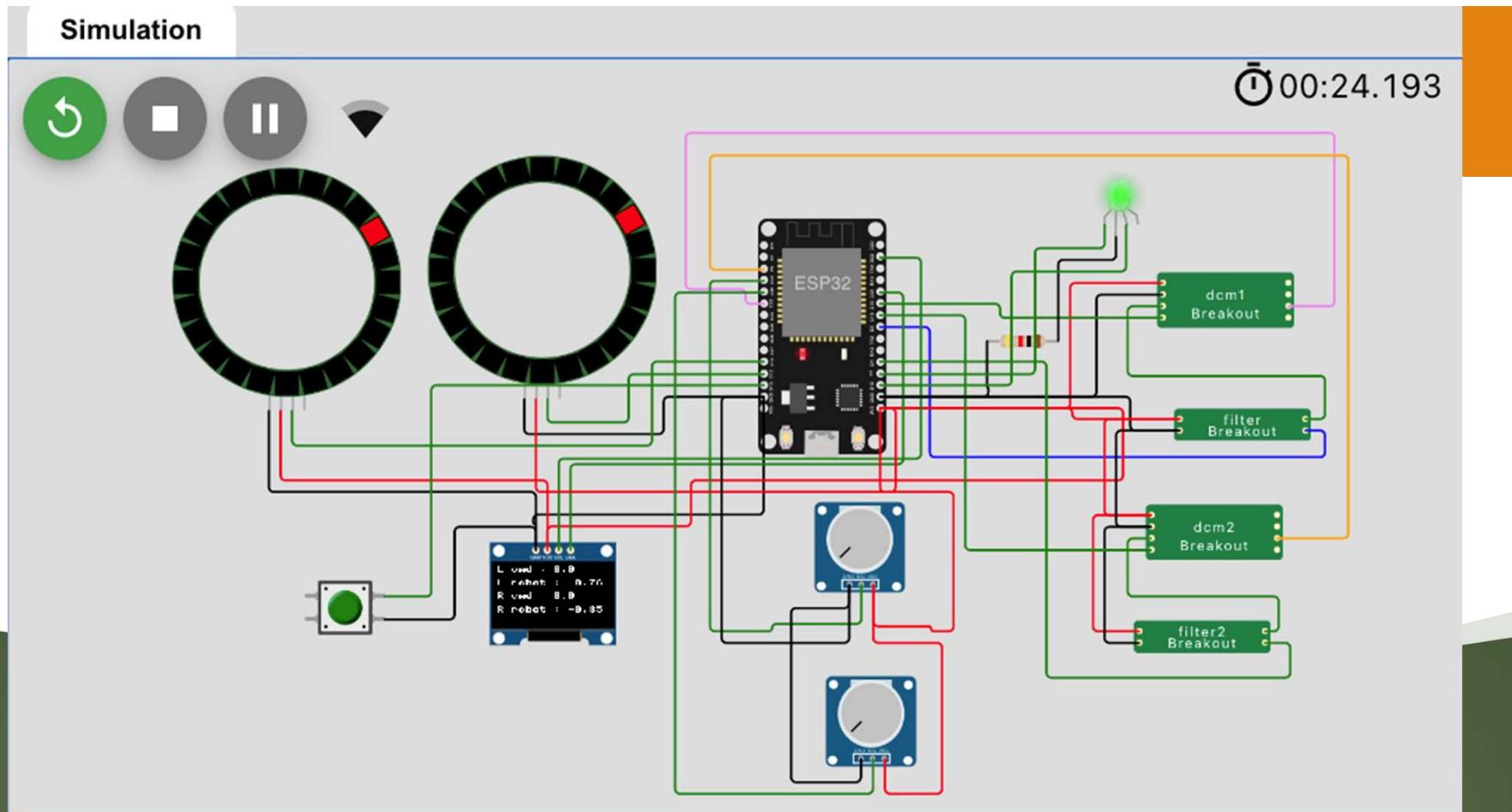
$$\begin{bmatrix} {}^W v_X \\ {}^W v_Y \\ {}^W \omega_Z \end{bmatrix} = \begin{bmatrix} v \cos(\theta) \\ v \sin(\theta) \\ \omega \end{bmatrix} \quad (8.45)$$

$$x(t) = x_0 + \int_0^t v(\tau) \cos(\theta(\tau)) d\tau$$

$$y(t) = y_0 + \int_0^t v(\tau) \sin(\theta(\tau)) d\tau \quad (8.46)$$

$$\theta(t) = \theta_0 + \int_0^t \omega(\tau) d\tau$$

การวัดระยะทาง (odometry) เพื่อประมาณค่าตำแหน่งหุ่นยนต์



ผังวงจรสำหรับตัวควบคุมหุ่นยนต์สองล้อบน Wokwi

<https://wokwi.com/projects/389517381277100033>

# ขาของ ESP32 ที่ใช้สำหรับ หุ่นยนต์สีม

ขา	หน้าที่
2	LED สีแดงแสดงสถานะหยุดทำงาน
4	PWM สำหรับล้อด้านขวา
5	PWM สำหรับล้อด้านซ้าย
12	วง NeoPixel LED สำหรับล้อด้านขวา
13	สลับสถานะ disable/enable
14	วง NeoPixel LED สำหรับล้อด้านซ้าย
15	LED สีเขียวแสดงสถานะพร้อมทำงาน
18	ทิศทางหมุน (DIR) สำหรับล้อด้านขวา
19	ทิศทางหมุน (DIR) สำหรับล้อด้านซ้าย
22	I2C SCL (สำหรับจอ SSD1306 OLED)
21	I2C SDA (สำหรับจอ SSD1306 OLED)
32	ความเร็วของล้อด้านซ้าย
34	คำสั่งความเร็วเชิงเส้นจาก VR
35	คำสั่งความเร็วเชิงมุมจาก VR
39	ความเร็วของล้อด้านขวา

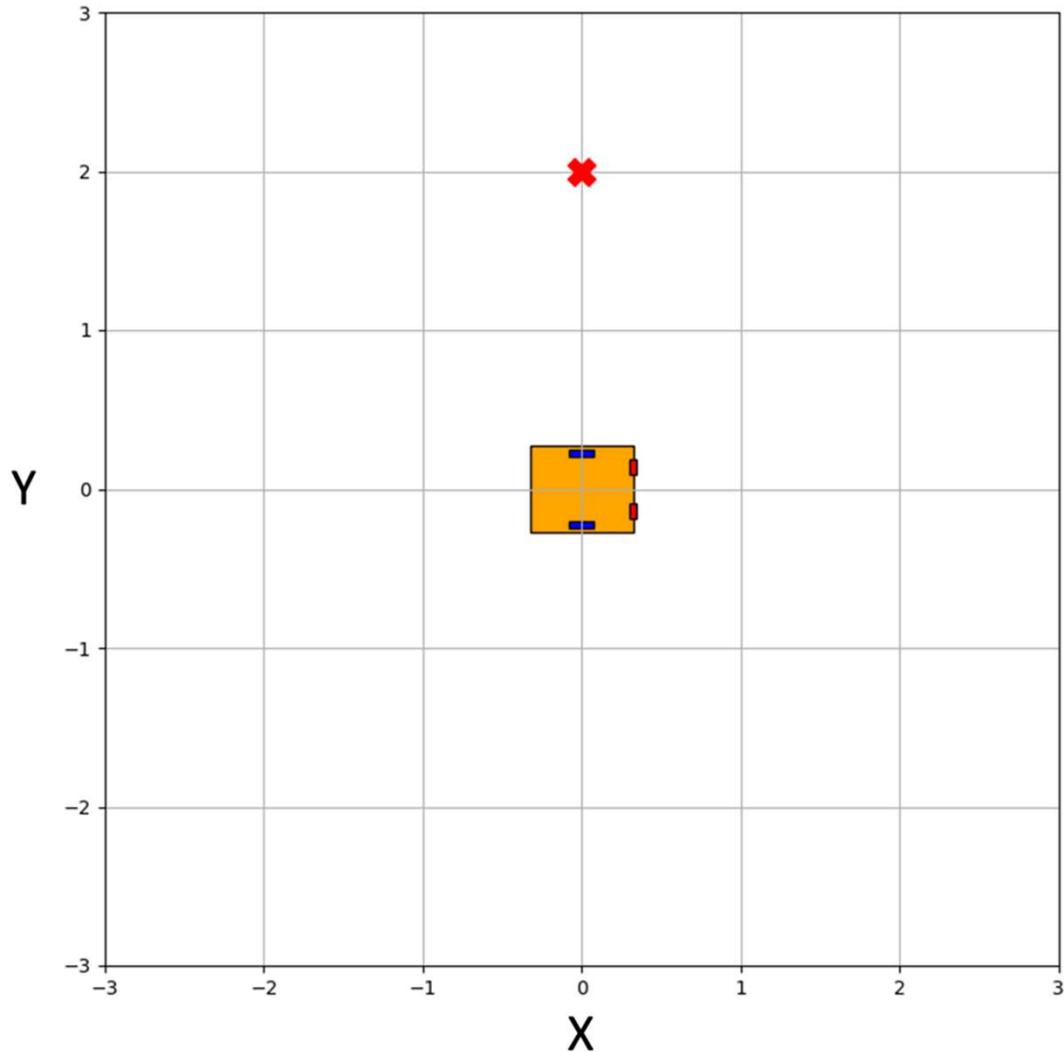


```
def update_tk():
    global updatestr
    updatestr = "{} ,{} ,{} ,{} ,{} ".format(ddrobot_pose[X],
                                              ddrobot_pose[Y],ddrobot_pose[THETA],enable,trackmode)
    if online:
        client.publish('@msg/update', updatestr)
```

## การควบคุมและแสดงตำแหน่งผ่าน Wokwi

Wokwi link : <https://wokwi.com/projects/389518777719375873>

Jupyter notebook : ddtk\_track.ipynb

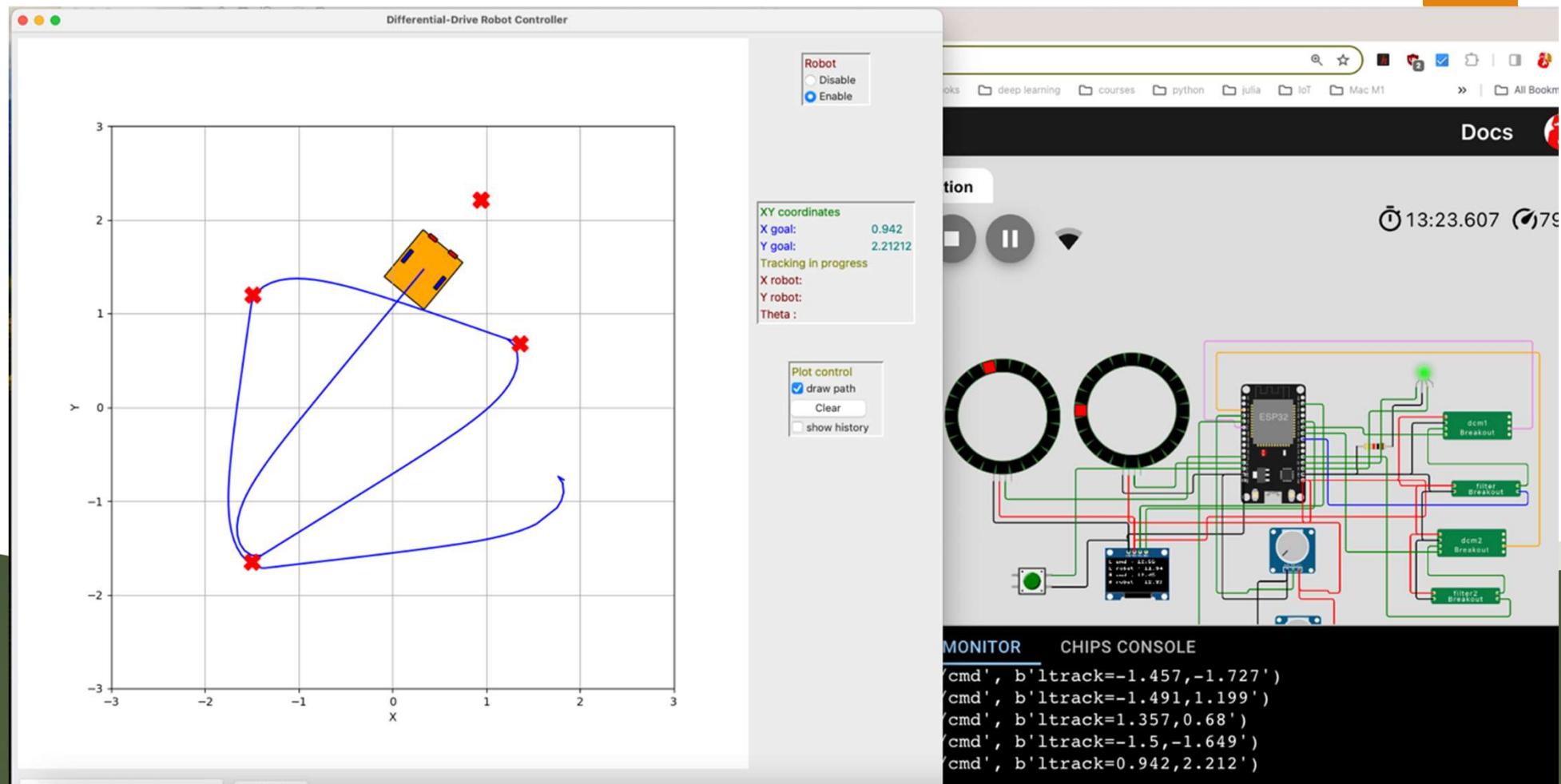


หุ่นยนต์สัมภាន  
ดำเนินการเริ่มต้น



```
if trackmode: # move to drobot_goal
    dp[X] = drobot_goal[X] - drobot_pose[X]
    dp[Y] = drobot_goal[Y] - drobot_pose[Y]
    norm_dp = math.sqrt(dp[X]**2 + dp[Y]**2)
    e = math.atan2(dp[Y],dp[X]) - drobot_pose[THETA]
    K = 3.0 # adjust this for sensitivity
    w_cmd = K*math.atan2(math.sin(e),math.cos(e))
    if norm_dp > 0.1: # distanct to target is still too large
        v_cmd = 1.0
    else: # stop the robot
        v_cmd = 0.0
        w_cmd = 0.0
    trackmode = 0
    enable = 0
```

ໂຄດກາຣເຄລືອນທີ່  
ເຂົ້າສູ່ເປົ້າໝາຍ



# การเคลื่อนที่เข้าสู่เป้าหมายของหุ่นยนต์สัมภาระ

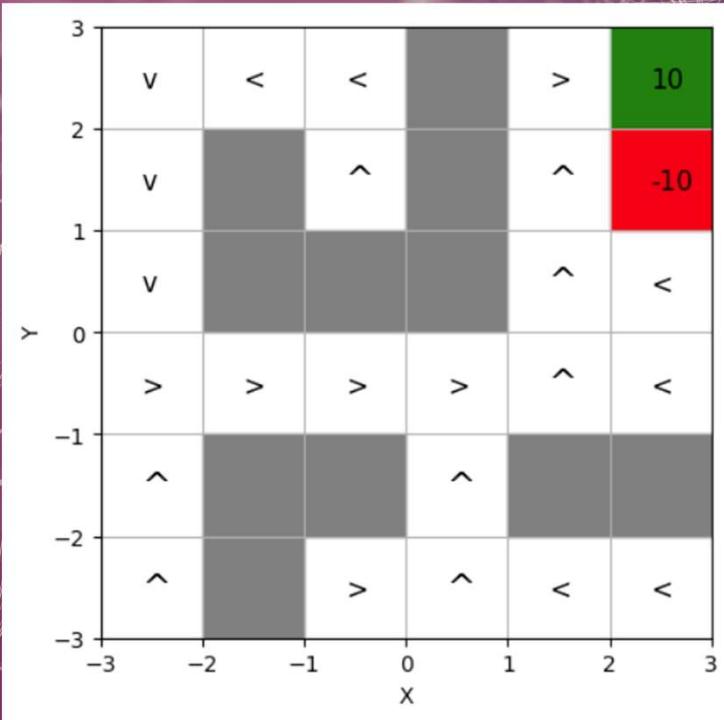


การวางแผนเส้นทาง  
การเคลื่อนที่ของ  
หุ่นยนต์

ROBOT PATH PLANNING

## ขั้นตอนวิธีการวนซ้ำมูลค่า (value iteration)

- ▶ เป็นพื้นฐานเบื้องต้นของการศึกษาการเรียนรู้เสริม กำลัง (reinforcement learning)
- ▶ คือวิธีการหนึ่งของปัญหาการโปรแกรมเชิงพอลวัต (dynamic programming) โดย นำคำตอบสมการของเบลแมน (Bellman equation)
- ▶ ยกตัวอย่างกริดเวิลด์ (gridworld) ขนาด  $6 \times 6$  ที่สามารถขยายไปยังขนาด  $n \times n$
- ▶ สถานะทั้งหมดอยู่ในรูปของ MDP (Markov decision process)



ในหนังสือนี้จะไม่กล่าวถึงรายละเอียดการอนุพัทธ์ โดยสามารถศึกษาได้จาก [14] หัวใจสำคัญของขั้นตอนวิธีการวนซ้ำมูลค่าคือสมการของเบล曼

$$v(s) = r(s) + \gamma \max_a \sum_{s'} p(s'|s, a)v(s') \quad (\text{A.1})$$

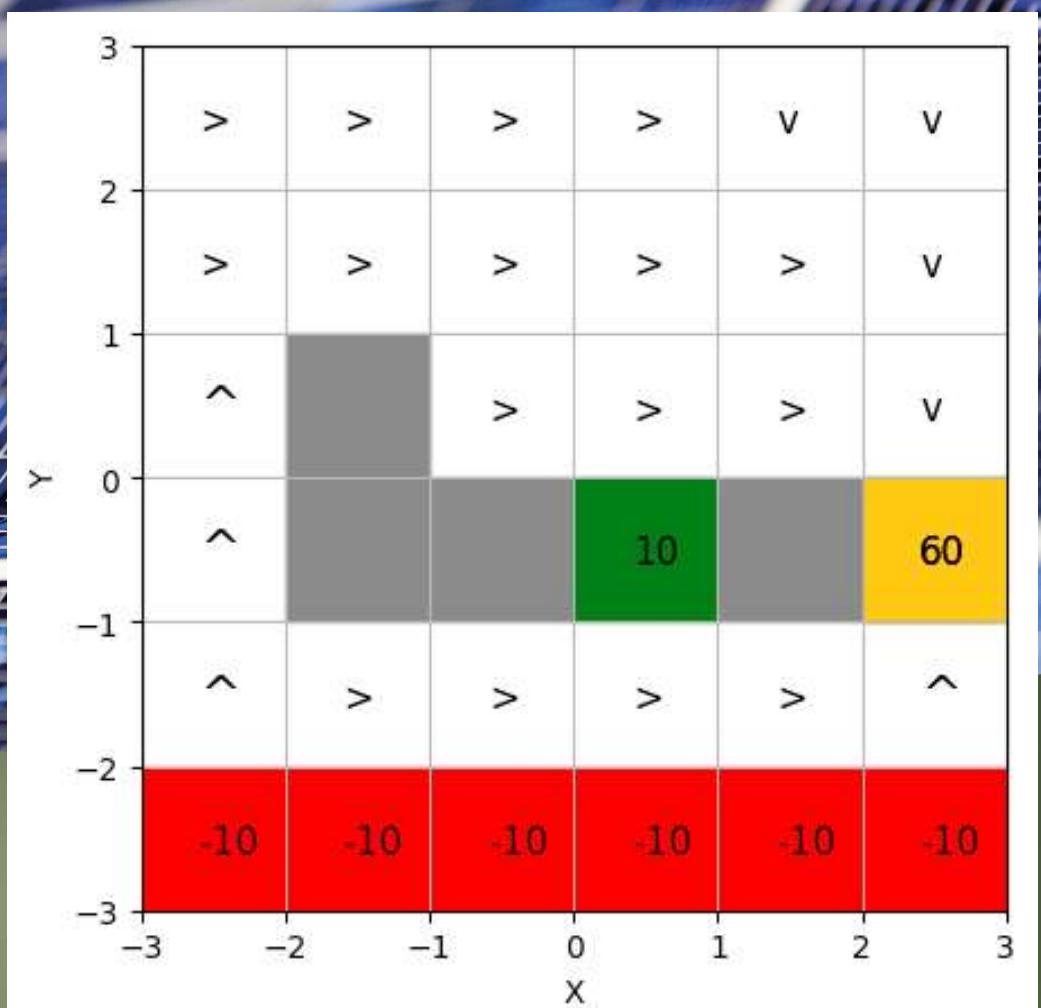
โดย  $r(s)$  แทนรางวัลของสถานะ  $s$  ค่า  $\gamma$  คือตัวแปรกอบส่วนลดในช่วง 0 – 1 และ  $p(s'|s, a)$  คือความน่าจะเป็นในการเลือกการทำ  $a$  เพื่อเปลี่ยนสถานะจาก  $s$  เป็น  $s'$  เริ่มต้นให้มูลค่าของทุกสถานะเท่ากับ 0 และคำนวณมูลค่า  $v(s)$  ของทุกสถานะ เลือกใช้วิธีการแทนค่า มูลค่าที่คำนวนได้โดยทันที (in place) เราสามารถพิสูจน์ทางคณิตศาสตร์ได้ว่าขั้นตอนวิธีวนซ้ำมูลค่าจะลู่เข้าสู่ค่าคงตัวในที่สุด โดยใช้วิธีการตรวจสอบว่ามูลค่าของแต่ละสถานะจะแตกต่างจากเดิมน้อยกว่าค่าที่กำหนด เช่น 0.001

## สมการของเบลמן



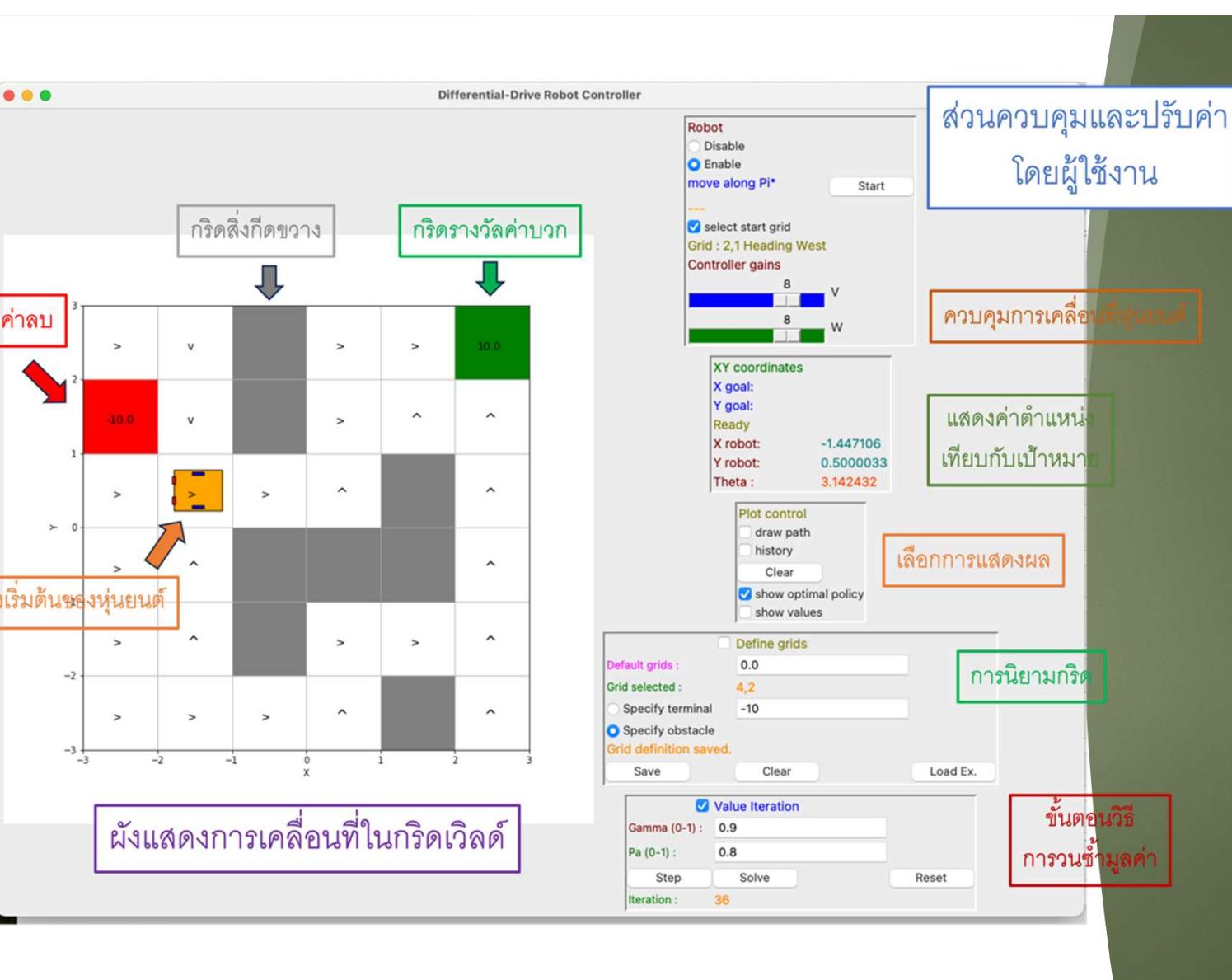
ทดสอบกริดเวิลด์ขนาด  $6 \times 6$

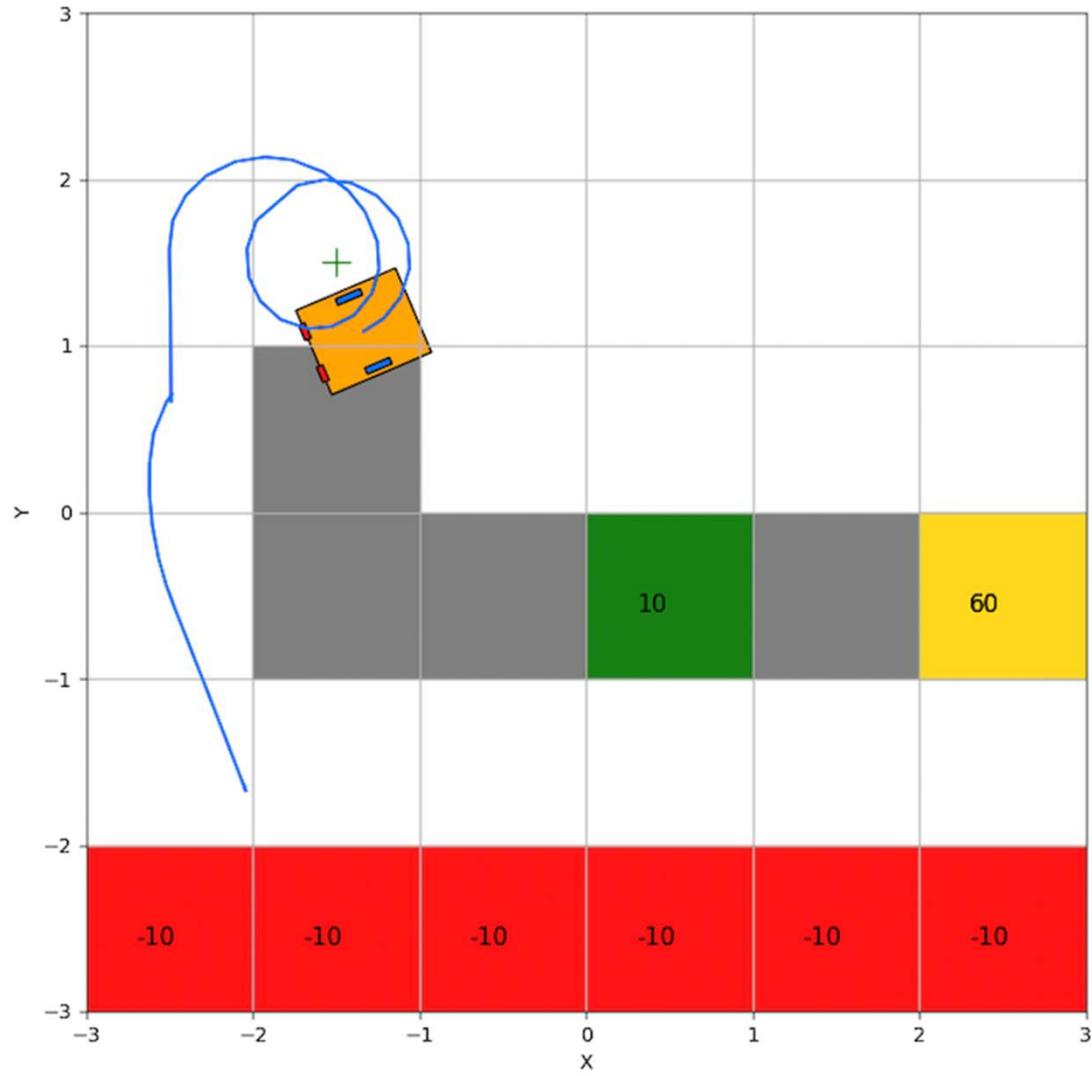
gridworld.ipynb



# GUI ทดสอบ การวางแผน เส้นทางหุ่นยนต์

ddtk\_gw6x6.ipynb





รูปแบบการ  
เคลื่อนที่เดิมไม่  
เหมาะสมกับการ  
ตามรอยกริด

## Exercise B.1

<https://wokwi.com/projects/389594525633230849>

► ใน Wokwi project

ที่ชื่อ `nuws24_ex_B_1_ddrobot_iot_gw`

► เขียนโค้ดเงื่อนไข `gtrackmode`: เพื่อให้หุ่นยนต์สามารถรอยกริดได้โดยไม่ชนสิ่งกีดขวาง

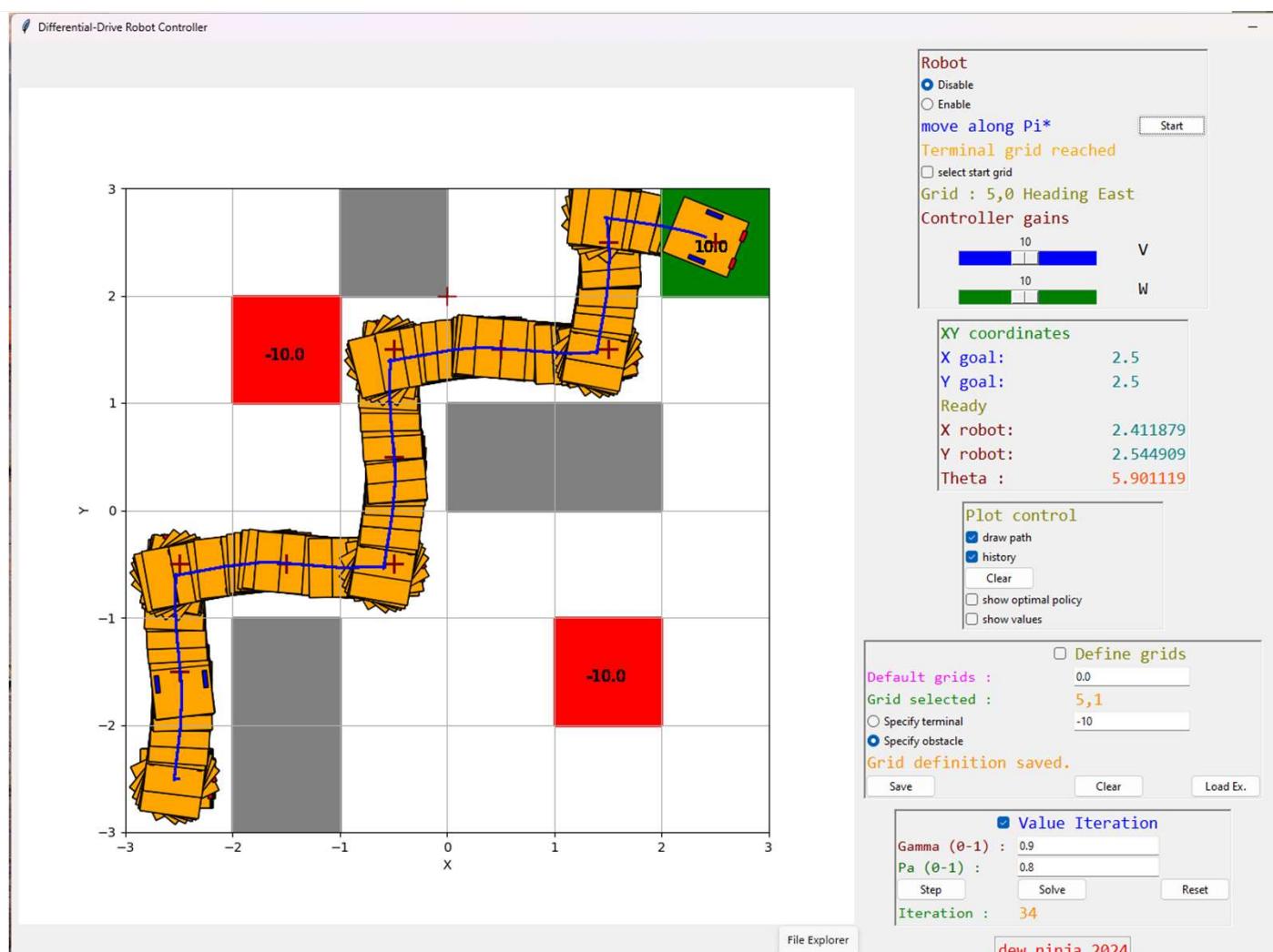
► ในโค้ดคำสั่ง `gtrack` ใน `cmdInt()` แก้ไข `trackmode=1` เป็น `gtrackmode=1`

► หากทำแบบฝึกหัดนี้ไม่สำเร็จ ใช้ลิงก์ Wokwi ในスタイルดัดแปลง

# ນາຮີ້ວິ່ງ

# history mode

ddtk\_gw6x6.ipynb



<https://wokwi.com/projects/389594465354786817>



- กำหนดเป้าหมาย Xg,Yg คือจุดศูนย์กลางของกริดต่อไปตามนโยบายหมายที่สุด
- ส่งคำสั่ง gtrack = xg,yg ไปยังหุ่นยนต์
- รับค่าตำแหน่งและทิศทางปัจจุบันของหุ่นยนต์ Xt,Yt,theta เพื่อแสดงภาพหุ่นยนต์
- รับค่าสถานะ enable เมื่อเท่ากับศูนย์หมายความว่าหุ่นยนต์ถึงเป้าหมายแล้ว คำนวน  
เป้าหมาย Xg,Yg ของกริดใหม่ หรือหากถึงปลายทางแล้วคือจบตอน

การควบคุมการเคลื่อนที่ตามนโยบายหมายที่สุด

## ฟังก์ชัน `follow_pistar()` อพဲในส่วนของ `animate()`

```
def follow_pistar(self): # steer robot along optimal policy
    if self.pistart:
        # find which state it is in
        self.robot_row, self.robot_col = self.detect_grid(self.xt,
                                                          self.yt)
        self.pistarmsg_txt.set("Current grid : (" +
                              str(self.robot_row)+","+str(self.robot_col)+")")
    if not self.enable: # initial, or target reached
        # check if a terminal is reached
        if self.terminal[self.robot_row, self.robot_col]:
            self.pistarmsg_txt.set("Terminal grid reached")
            self.pistart = False
            self.pibutton_txt.set("Start")
            self.showtarget = False
        else:
            # find center of next grid in pi* direction
            self.locate_next_grid_center()
            self.send_cmd_enable()
            time.sleep(0.5)
            self.track_next_grid()
    else: # check if robot hits an obstacle or falls into a pit
        self.check_clearance()
```

ฟังก์ชัน `locate_next_grid_center()` เพื่อคำนวณค่าศูนย์กลางของ  
กริดต่อไปจากเมทริกซ์ `self.pimat` ที่เก็บค่าทิศทางการเคลื่อนที่ตามนโยบายเมะ  
ที่สุด

```
def locate_next_grid_center(self):
    if self.pimat[self.robot_row, self.robot_col] == 0: # north
        next_row = self.robot_row - 1
        next_col = self.robot_col
    elif self.pimat[self.robot_row, self.robot_col] == 1: # east
        next_row = self.robot_row
        next_col = self.robot_col + 1
    elif self.pimat[self.robot_row, self.robot_col] == 2: # south
        next_row = self.robot_row + 1
        next_col = self.robot_col
    elif self.pimat[self.robot_row, self.robot_col] == 3: # west
        next_row = self.robot_row
        next_col = self.robot_col - 1
    self.xg = self.grid_center_x[next_col]
    self.yg = self.grid_center_y[next_row]
    self.xg_txt.set(str(self.xg))
    self.yg_txt.set(str(self.yg))
```

ฟังก์ชันเสริม **detect\_grid()** สำหรับคำนวณค่าหมายเลขอ각และคอลัมน์ของกริดเดิล์ดจากอาร์กิวเมนต์ที่ค่าพิกัด **X, y**

```
self.grid_bounds_x = np.array([-3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0])
self.grid_bounds_y = np.array([3.0, 2.0, 1.0, 0.0, -1.0, -2.0, -3.0])

def detect_grid(self,x,y): # find grid row and column from x,y
    row = col = 0
    for i in range(len(self.grid_bounds_x)-1):
        if x>self.grid_bounds_x[i] and x<= self.grid_bounds_x[i+1]:
            col = i
    for i in range(len(self.grid_bounds_y)-1):
        if y<=self.grid_bounds_y[i] and y> self.grid_bounds_y[i+1]:
            row = i
    return row, col
```

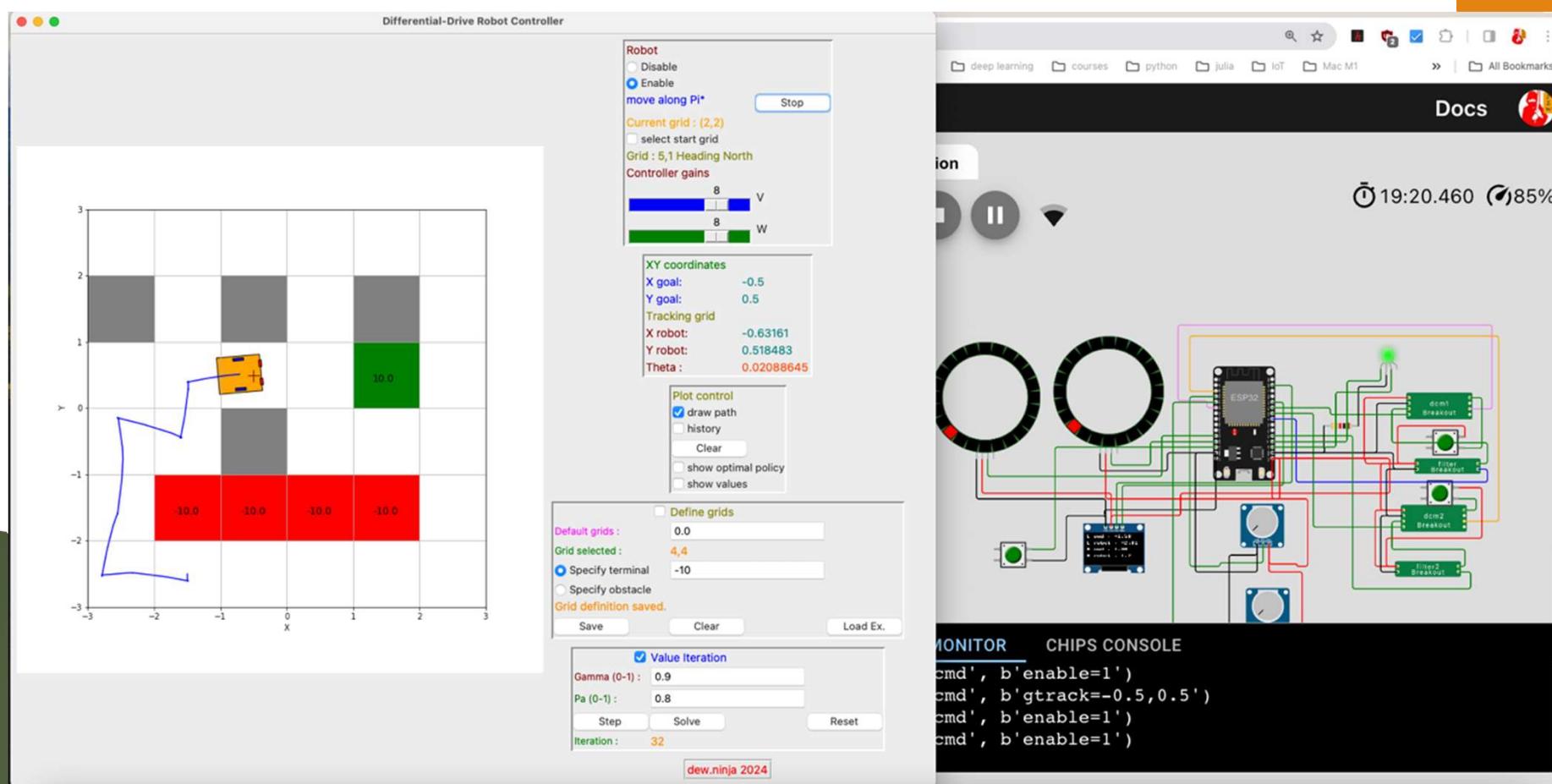
เมื่อคำนวนและบันทึกค่าตัวดำเนินการหน่วงศูนย์กลางของกริดใหม่ที่ต้องการเคลื่อนที่ในตัวแปร self.xg, self.yg และเรียกฟังก์ชัน **track\_next\_grid()** เพื่อส่งคำสั่งตามรอยกริด

```
def track_next_grid(self): # send command to track next grid
    if self.online:
        if (self.xg_prev != self.xg) or (self.yg_prev != self.yg):
            cmd_txt = "gtrack="+str(round(self.xg,3)) + "," + \
                      str(round(self.yg,3))
            self.client.publish("@msg/cmd",cmd_txt)
            self.xg_prev = self.xg
            self.yg_prev = self.yg
```

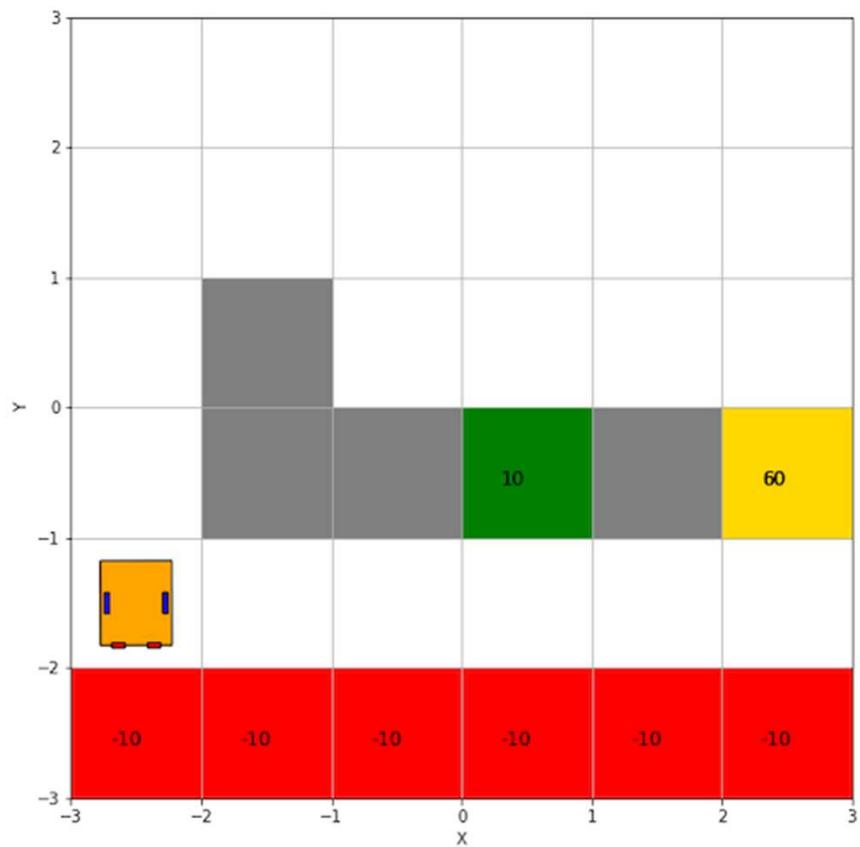
ໂດຍພັງກໍຕັ້ນ

```
def on_message(self,client, userdata, message):
    self.rcvd_msg = str(message.payload.decode("utf-8"))
    self.rcvd_topic = message.topic

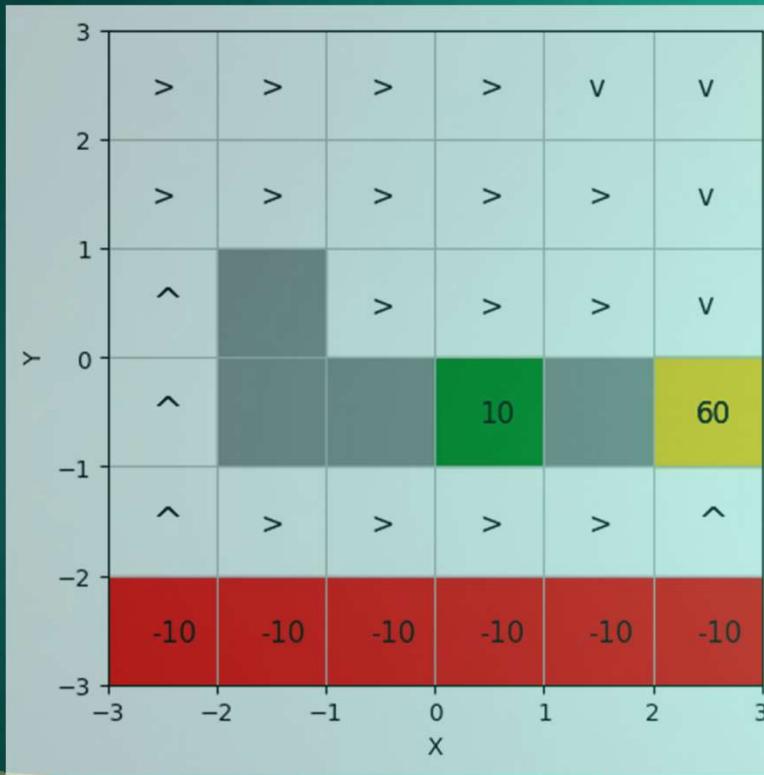
    if self.rcvd_topic == "@msg/update":
        parm_data = self.rcvd_msg.split(',')
        # strings for GUI update
        self.enable_txt.set(parm_data[3])
        self.xt_txt.set(parm_data[0])
        self.yt_txt.set(parm_data[1])
        self.theta_txt.set(parm_data[2])
        if self.gtrackmode:
            self.gtrackmsg_txt.set("Tracking grid")
        else:
            self.gtrackmsg_txt.set("Ready")
        # update parameters
        self.enable = int(parm_data[3])
        self.gtrackmode = int(parm_data[4])
        self.xt = float(parm_data[0])
        self.yt = float(parm_data[1])
        self.theta = float(parm_data[2])
```



การทำงานของ GUI ร่วมกับการจำลองหุ่นยนต์บน Wokwi



ปัญหากริดเวิลเด้ใน  
ตัวอย่าง 8.12  
ในหนังสือ



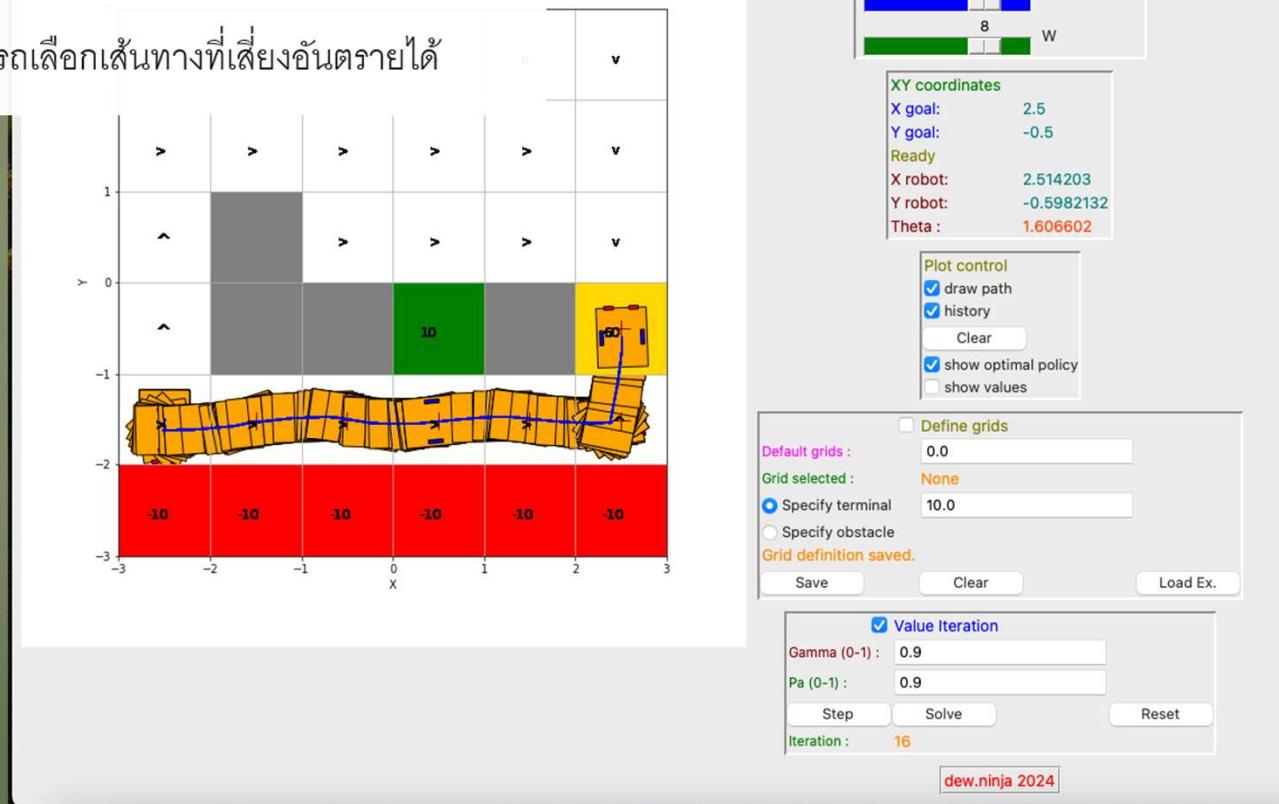
# การปรับนโยบายเหมาะสมที่สุด โดยโปรแกรมเตอร์การวนซ้ำมุ่ลค่า

ตัวอย่างนี้ดัดแปลงจาก Exercise 1 ในวีดีโอ L1 MDPs, Exact Solution Methods, Max-ent RL โดย Prof. Pieter Abbeel, University of California, Berkeley [https://youtu.be/2GwBez0D20A?si=XL-\\_Y6YStqxFA3oL](https://youtu.be/2GwBez0D20A?si=XL-_Y6YStqxFA3oL)

กรณี 1:  $\gamma = 0.9$ ,  $P_a = 0.9$

ค่าพารามิเตอร์นี้แสดงนัยว่า

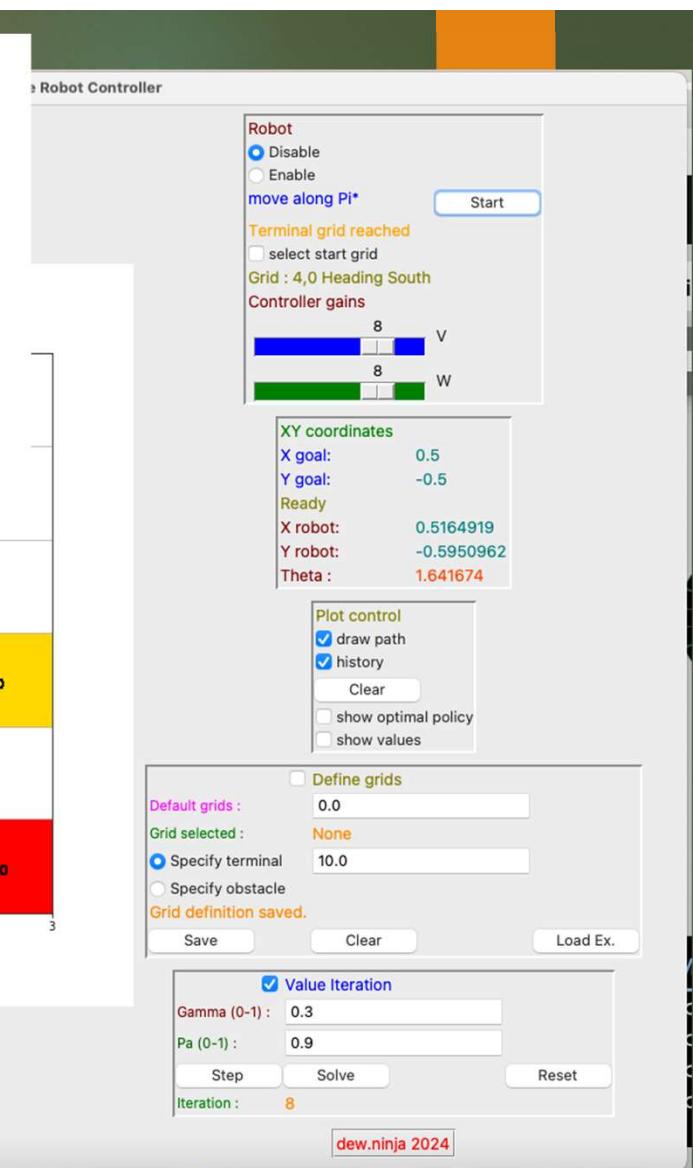
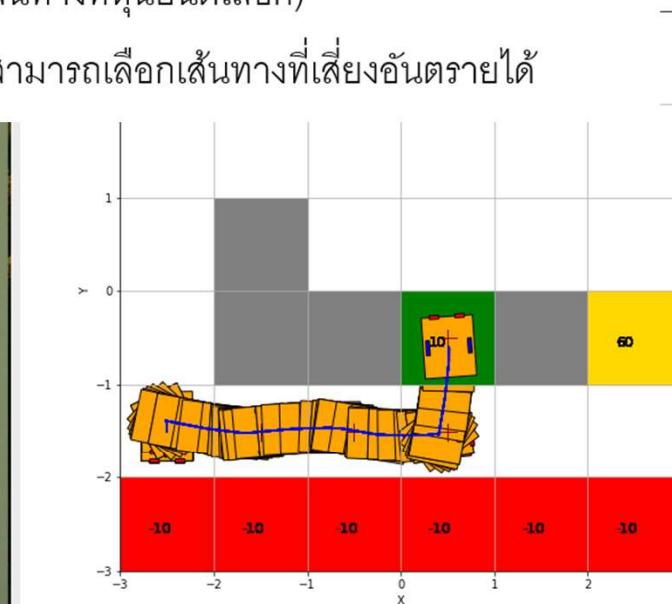
- ตัวประกอบส่วนลดมีค่ามากเข้าใกล้ 1 ดังนั้นหุ่นยนต์สามารถเดินทางได้ไกลโดยร่วงวัดจะเสื่อมราคาลงไม่มาก
- หุ่นยนต์มีความแม่นยำในการเคลื่อนที่สูง สามารถเลือกเส้นทางที่เสี่ยงอันตรายได้



กรณี 2:  $\gamma = 0.3$ ,  $P_a = 0.9$

ค่าพารามิเตอร์นี้แสดงนัยว่า

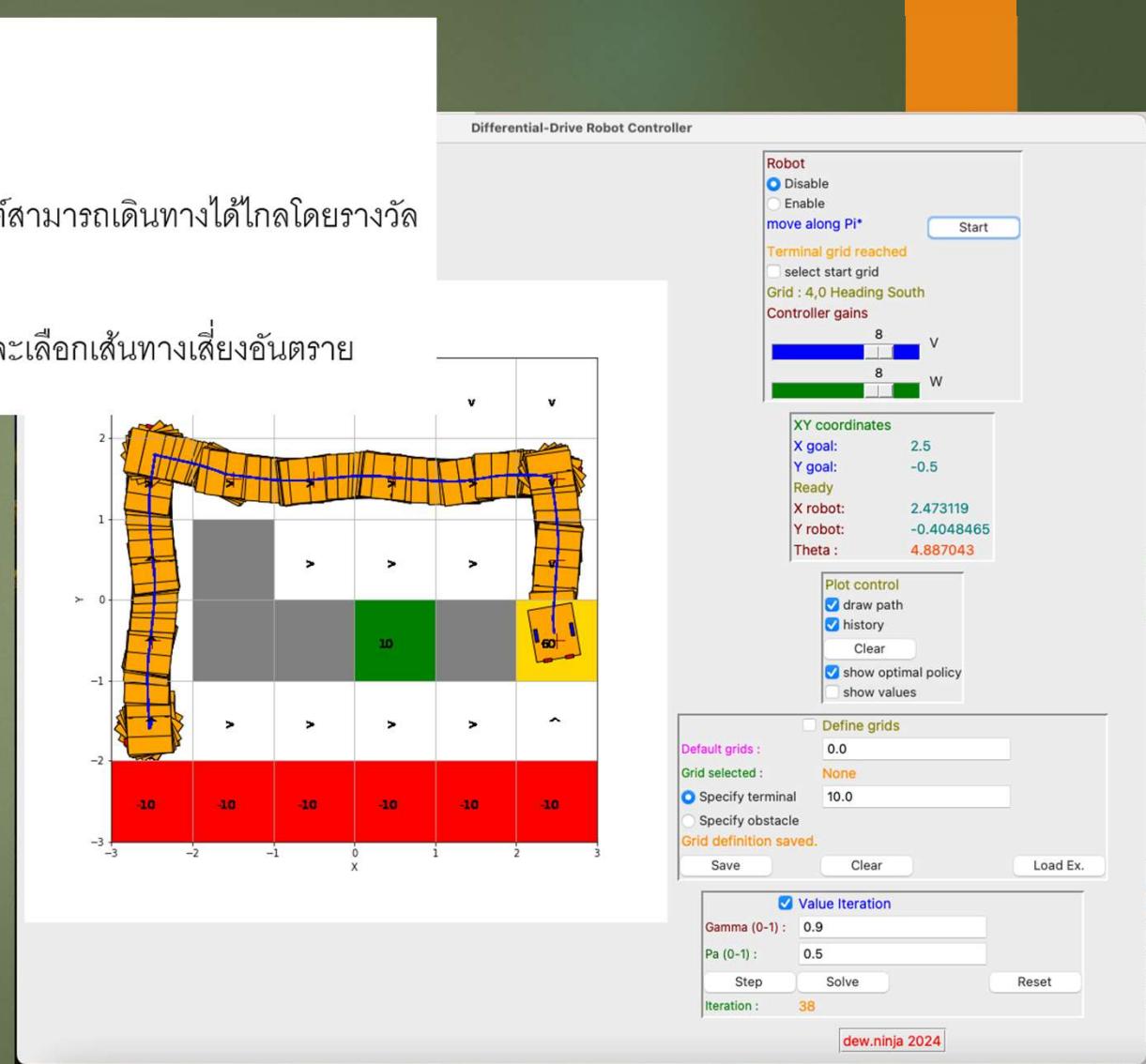
- ตัวประกอบส่วนลดมีค่าเพียง 0.3 กล่าวคือรังวัลจะลดค่าลงอย่างรวดเร็วตามเวลา (เวลาในการเดินทางขึ้นกับจำนวนกริดในเส้นทางที่หุ่นยนต์เลือก)
- หุ่นยนต์มีความแม่นยำในการเคลื่อนที่สูง สามารถเลือกเส้นทางที่เสี่ยงอันตรายได้



กรณี 3:  $\gamma = 0.9$ ,  $P_a = 0.5$

ค่าพารามิเตอร์นี้แสดงนัยว่า

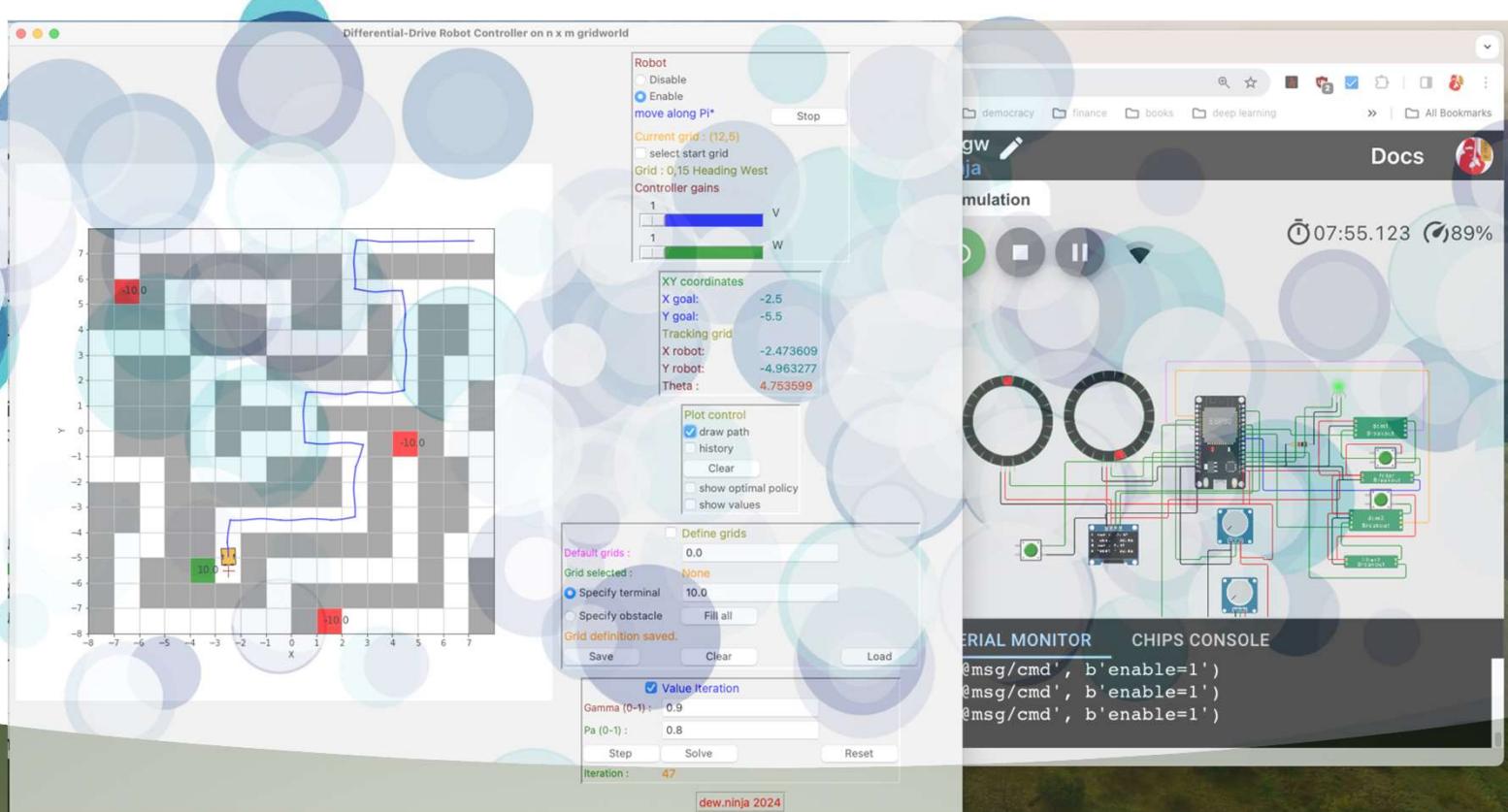
- ตัวประกอบส่วนลดมีค่ามากเข้าใกล้ 1 ดังนั้นหุ่นยนต์สามารถเดินทางได้ไกลโดยร่วงวัดจะเสื่อมราคาลงไม่มาก
- หุ่นยนต์มีความแม่นยำในการเคลื่อนที่ต่ำ ไม่เหมาะสมที่จะเลือกเส้นทางเสี่ยงอันตราย



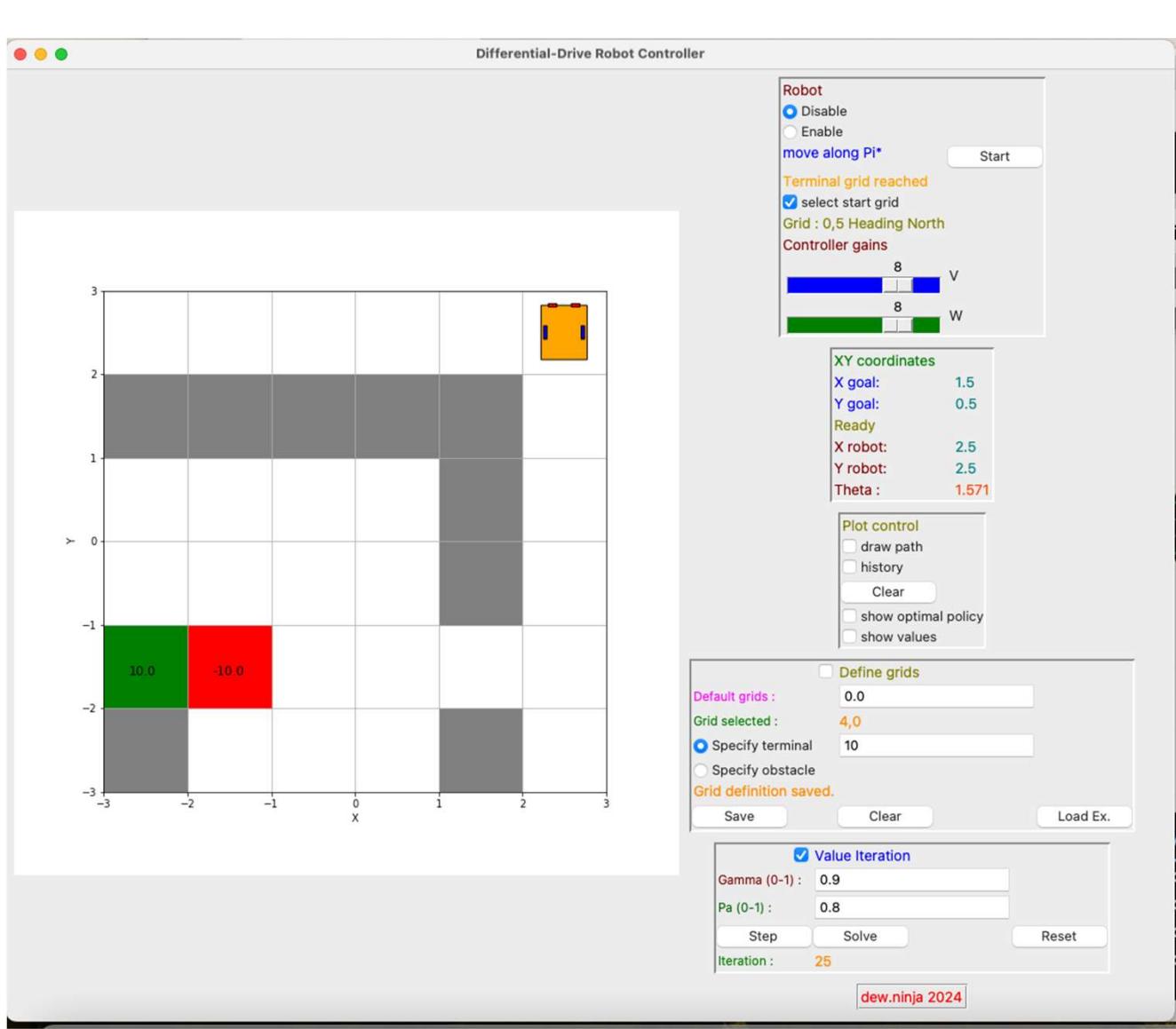


ทิ้งให้เป็นแบบฝึกหัดสำหรับผู้อ่านสำหรับกรณี 4 คือกำหนด  $\gamma = 0.3$ ,  $P_a = 0.5$  ก่อนที่จะทดลองบนกริดเวิลด์ ลองคาดเดาว่าหุ่นยนต์สัมจะเลือกนโยบายเหมาะสมที่สุดอย่างไร

## Exercise B.2

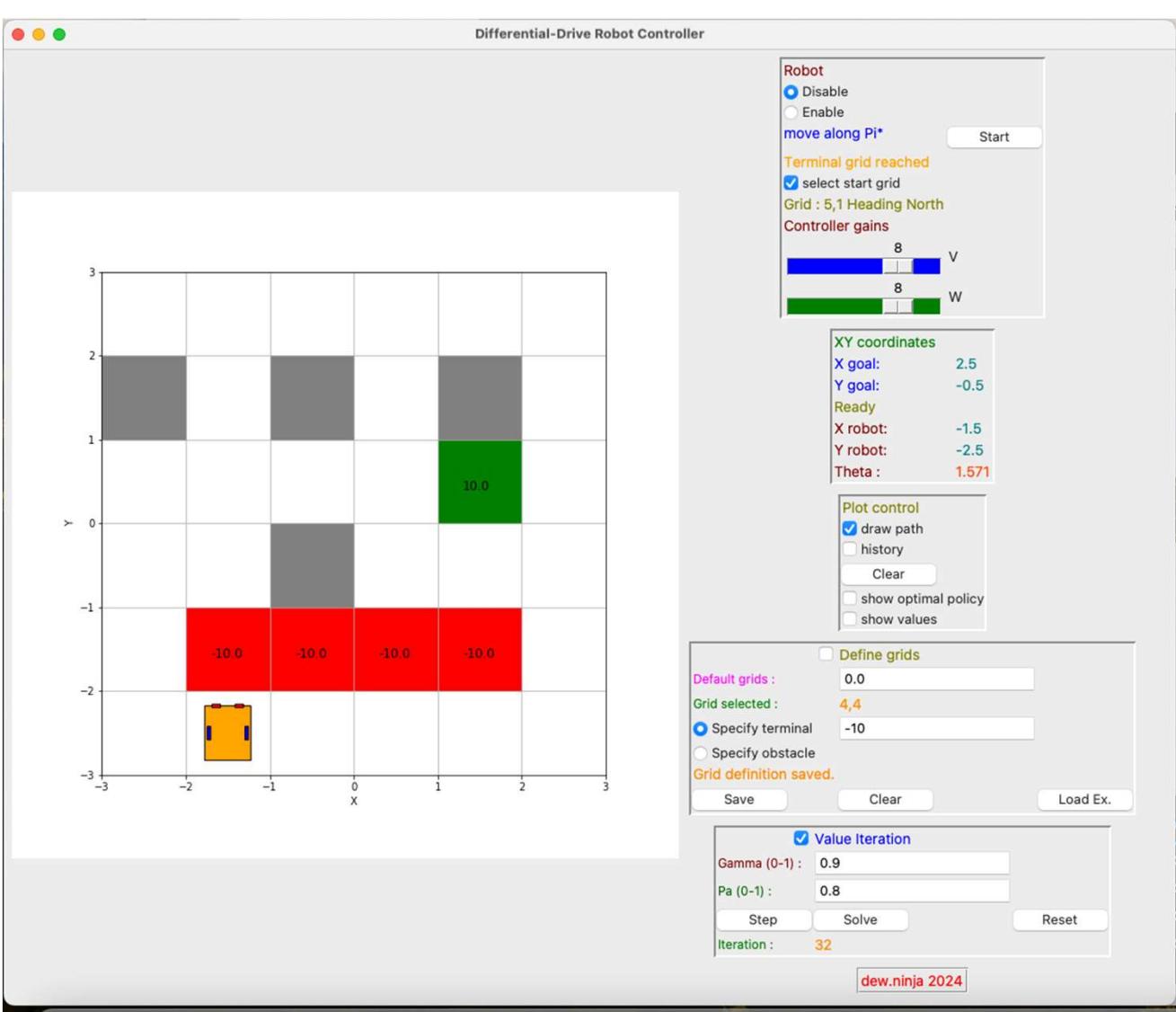


ตัวอย่างโจทย์ปัญหาการวางแผนเส้นทางเดินหุ่นยนต์



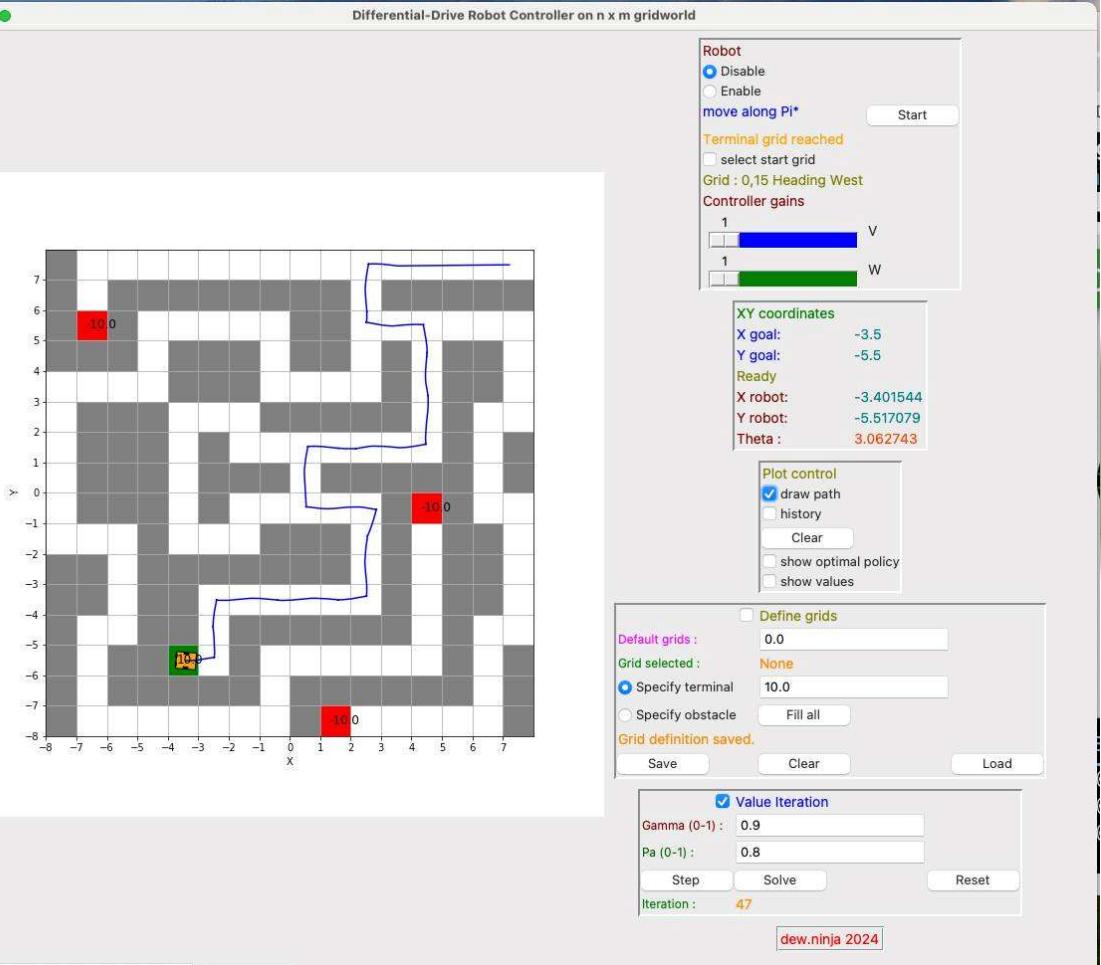
หุ่นยนต์นำหนังสือพิมพ์  
มาให้เจ้าของที่เลี้ยงเมว  
ในห้อง

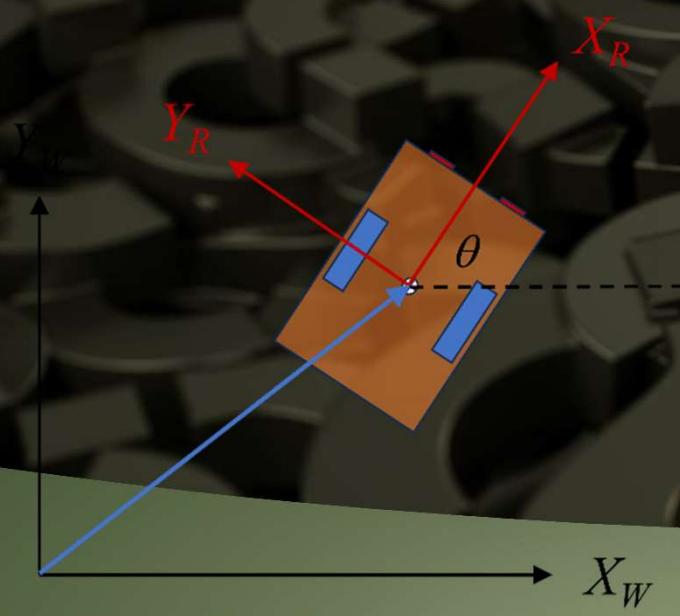
# หุ่นยนต์บริการ อาหาร ให้ลูกค้า



# แก้ปัญหามaze โดย สร้างเป็นกริดเวลเดอร์ nxn

ddtk\_gwnxn.ipynb





## Part B Summary & Questions

ภาคบ่าย (1:00 – 4:00 PM)