

LAB2 (Romeona Robot)

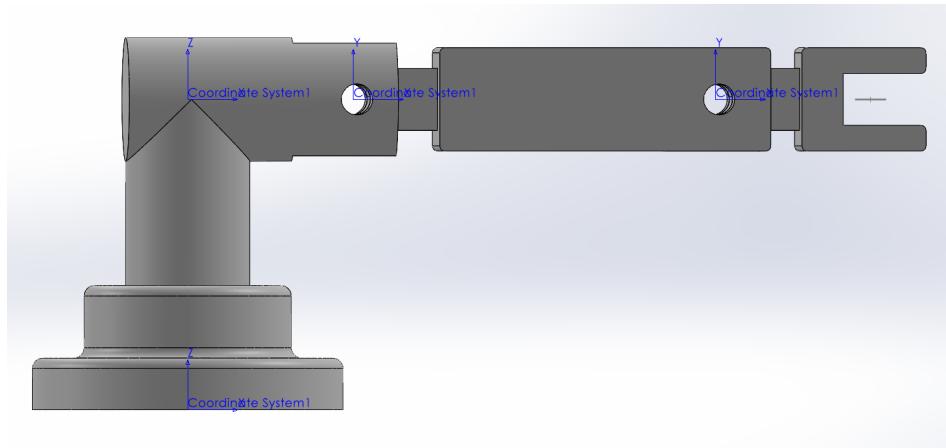
Group 23

Lab 2 จะประกอบไปด้วย 3 packages คือ

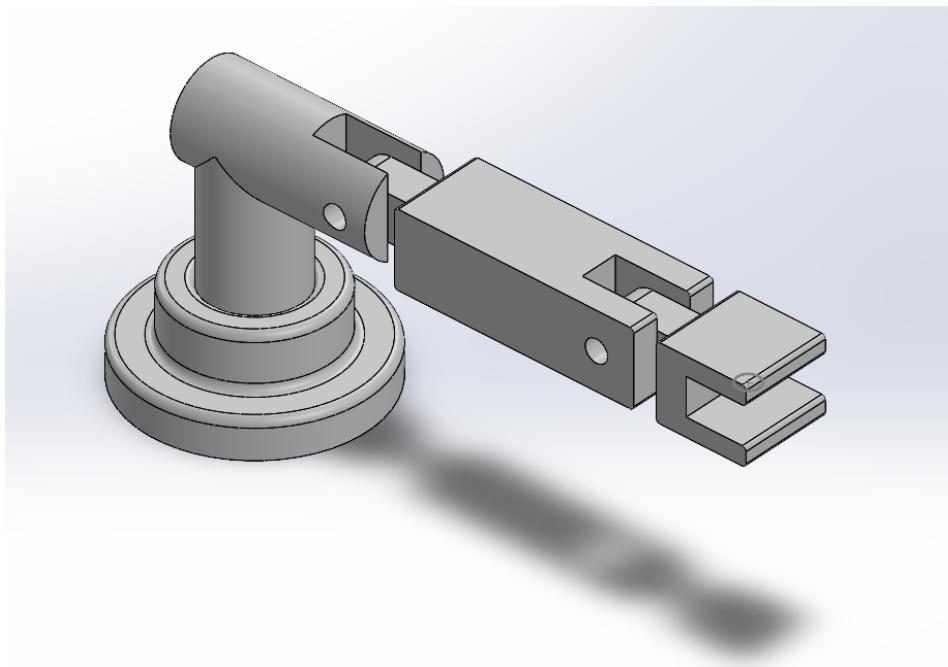
- romeona_description
- romeona_kinematics
- romeona_kinematics_interfaces

การสร้างแบบจำลองหุ่นยนต์ที่เป็นแขนกล

หุ่นยนต์แขนกลที่สร้างไว้เป็น 3 DOF และมีการกำหนด Coordinate frame ของแต่ละ Joint โดยมีรูปดังนี้

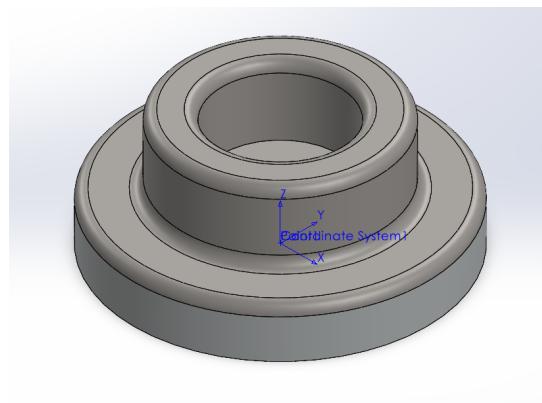


รูป cad 3D Front View

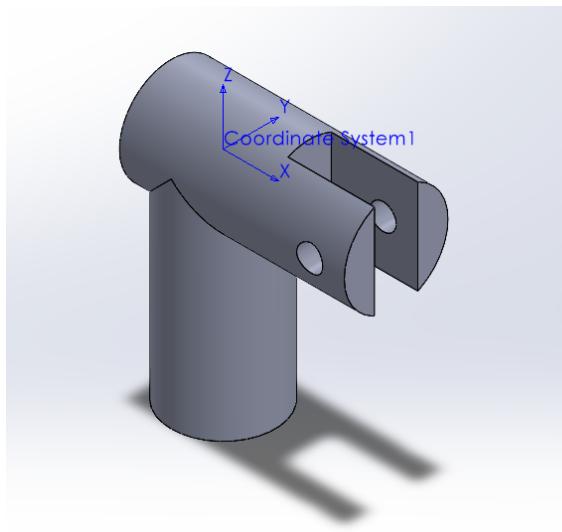


รูป cad 3D Isometric View

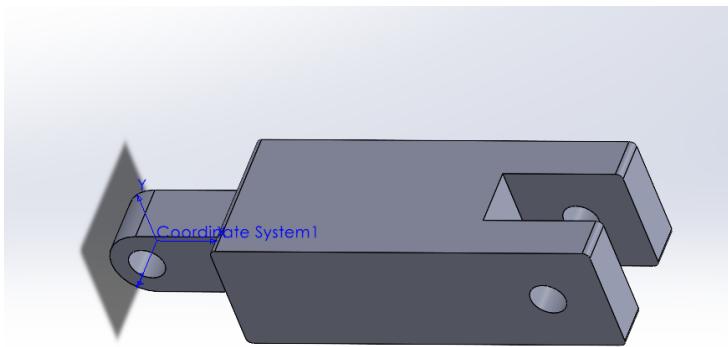
โดยมีองค์ประกอบเป็น



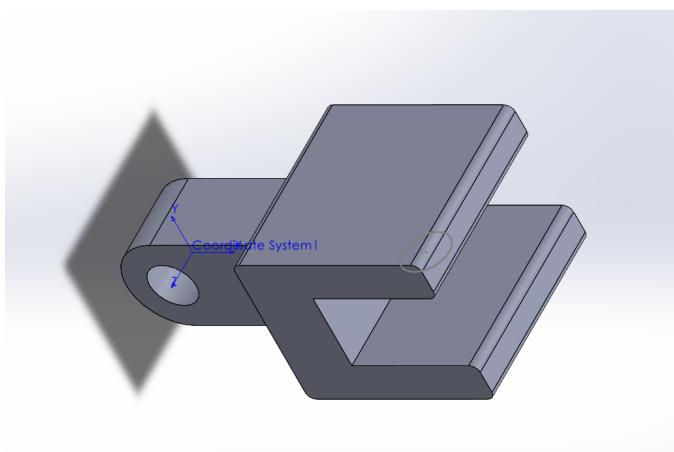
base



link_1



link_2



end-effector

หุ่นยนต์แขนกลมี DH-Parameter ดังนี้

$i - 1$	i	a_i	α_i	d_i	θ_i
0	1	0	0	0.3	0
1	2	0.16	$\frac{\pi}{2}$	0	0
2	3	0.35	0	0	0
3	end-eff			H_e^3	

a_i คือ link length

α_i คือ link twist

d_i คือ link offset

θ_i คือ joint offset

$$H_e^3 = \begin{bmatrix} 0 & 0 & 1 & 0.16 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

โดย type of joint ทั้งหมดเป็น revolute joint

Package : romeona_description

Package นี้ไว้สำหรับแสดงผลการจำลองทุนยนต์แขนกล 3 DOF ใน rviz ซึ่งเราจะเขียน romeona.xacro ขึ้นมาเพื่อให้ CAD ที่เราทำแสดงผลออกไปบน rviz ได้อย่างถูกต้อง

ในการสร้าง coordinate frame เราจะสร้างไว้ที่ joint ใน CAD ดังนั้น เราจึงต้องกำหนด <origin> ของแต่ละ joint ใน romeona.xacro ดังนี้

```
<joint name="joint_1" type="revolute">
  <parent link="base_link"/>
  <child link="link_1"/>
  <origin rpy="0 0 0" xyz="0 0 0.3"/>
  <axis xyz="0 0 1"/>
  <limit lower="-3.14" upper="3.14" effort="0.5" velocity="0.5"/>
</joint>
```

joint_1 จะเชื่อมต่อระหว่าง base_link กับ link_1 โดย link_1 จะถูกเลื่อนไปตามแนวแกน z เป็นระยะ 0.3 m และมีการหมุน จึงกำหนดค่า origin เป็น rpy = “0 0 0” xyz = “0 0 0.3” และ joint_1 หมุนตามแกน Z กำหนดให้ค่า axis เป็น xyz = “0 0 1”

```
<joint name="joint_2" type="revolute">
  <parent link="link_1"/>
  <child link="link_2"/>
  <origin rpy="1.57079632679 0 0" xyz="0.16 0 0"/>
  <axis xyz="0 0 1"/>
  <limit lower="-3.14" upper="3.14" effort="0.5" velocity="0.5"/>
</joint>
```

joint_2 จะเชื่อมต่อระหว่าง link_1 กับ link_2 โดย link_2 จะถูกเลื่อนตามแนวแกน x เป็นระยะ 0.16 m และมีการหมุนตามแกน Roll ไป $\frac{\pi}{2}$ หรือเท่ากับ 1.57

จึงกำหนดค่า origin เป็น rpy = “1.57 0 0” xyz = “0.16 0 0” และ joint_2 หมุนตามแกน Z จึงกำหนดให้ค่า axis เป็น xyz = “0 0 1”

```
<joint name="joint_3" type="revolute">
```

```

<parent link="link_2"/>
<child link="link_3"/>
<origin rpy="0 0 0" xyz="0.35 0 0"/>
<axis xyz="0 0 1"/>
<limit lower="-3.14" upper="3.14" effort="0.5" velocity="0.5"/>
</joint>

```

joint _2 จะเชื่อมต่อระหว่าง link_2 กับ link_3 โดย link_3 จะถูกเลื่อนตามแนวแกน x เป็นระยะ 0.35 m และไม่มีการหมุนแกน
จึงกำหนดค่า origin เป็น

`rpy = " 0 0 0 " xyz = " 0.35 0 0 "` และ joint_3 หมุนตามแกน Z
จึงกำหนดให้ค่า axis เป็น xyz = “ 0 0 1 ”

```

<joint name="joint_eff" type="fixed">
  <parent link="link_3"/>
  <child link="end_effector"/>
  <origin rpy="-1.57 0 0" xyz="0.16 0 0"/>
  <axis xyz="0 0 1"/>
  <limit lower="-3.14" upper="3.14" effort="0.5" velocity="0.5"/>
</joint>

```

joint _eff จะเชื่อมต่อระหว่าง link_3 กับ end_effector โดย end_effector จะถูกเลื่อนตามแนวแกน x = 0.16 m เรากำหนด
ให้แกนของ end_effector มีแกน x ชี้ไปข้างหน้า แกน z ชี้ขึ้น และแกน y ชี้ออกมาจากหน้าจอ ดังนั้นจึงต้องทำการการหมุนตาม
แกน Roll ไป $-\frac{\pi}{2}$ หรือเท่ากับ -1.57

จึงกำหนดค่า origin เป็น `rpy="-1.57 0 0" xyz="0.16 0 0"` และ joint_eff หมุนตามแกน Z จึงกำหนดให้ค่า axis เป็น xyz = “
0 0 1 ”

การเขียน display.launch.py

- เพิ่ม node rviz2 และเรียกใช้ romeona.rviz
- เพิ่ม node robot_state_publisher และเรียกใช้ romeona.xacro เพื่อเป็นโมเดลในการจำลอง

CMakeList

เราที่จะทำการ add folders ที่อยู่ใน package นี้เข้าไปทั้งหมด

```

install(DIRECTORY
  # add directories here

```

```
scripts
config
launch
meshes
robot
DESTINATION share/${PROJECT_NAME})

ament_package()
```

Package : romeona_kinematics

ใน folder scripts จะมีไฟล์ kinematics_server.py ซึ่งเราจะสร้าง Node ชื่อ kinematics_server

```

class Kinematics_Server(Node):
    def __init__(self):
        super().__init__('kinematics_server')
        # get the rate from argument or default
        self.rate = 10.0

```

กำหนด rate ในการ publish คือ 10 Hz

```

# Create Service
self.joint_states = self.create_serviceGetPosition,'set_joint',self.set_joint_callback)
self.joint_states_ik = self.create_service(SolveIK,'solve_ik',self.set_joint_callback_ik)

```

สร้าง Service ขึ้นมา 2 ตัว

1. joint_states เป็น service ที่รับค่า q (มุมของแต่ละ joint) ซึ่งจะ set_joint โดยเรียก service server ชื่อ GetPosition ผ่านฟังก์ชัน set_joint_callback
2. joint_states_ik เป็น service ที่รับค่า x y z (ตำแหน่งของ End-effector เทียบกับ Base ของหุ่นยนต์) ซึ่งจะ solve_ik โดยเรียก service server ชื่อ SolveIK ผ่านฟังก์ชัน set_joint_callback_ik

```

# Create Publisher
self.command_publisher = self.create_publisher(JointState,'/joint_states',10)

# Publish
timer_period = 1/self.rate
self.timer = self.create_timer(timer_period,self.timer_callback)

```

สร้าง publisher โดยมี topic ชื่อ joint_states และส่ง message ชนิด JointState

ในการ publish เราจะทำการส่งค่าด้วยความถี่ 10 Hz หรือคือ 0.1 second ผ่านฟังก์ชัน timer_callback

```

# additional attributes
self.joint_state_position = JointState()
self.joint_state_position.name = ['joint_1','joint_2','joint_3']
self.joint_state_position.position = [0., 0., 0.]

```

สร้างตัวแปรชื่อ joint_state_position กำหนดเป็น message JointState และทำการ set ค่า name ซึ่งเป็นชื่อเดียวกับ joint ที่เราตั้งไว้ในไฟล์ romeona.xacro กับ position ซึ่งเป็นค่า default ของมุม joint คือ [0,0,0] (rad)

JointState() จะสามารถเก็บข้อมูลได้ดังต่อไปนี้

```
std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort
```

ฟังก์ชัน set_joint_callback

```
def set_joint_callback(self,request:GetPosition.Request,response:GetPosition.Response):
    self.joint_state_position.position[0] = request.q.position[0]
    self.joint_state_position.position[1] = request.q.position[1]
    self.joint_state_position.position[2] = request.q.position[2]

    fk = forward_kinematics(self.joint_state_position.position)
    position = Point()
    position.x = fk[0][3]
    position.y = fk[1][3]
    position.z = fk[2][3]
    response.p = position

    return response
```

ฟังก์ชันนี้จะทำการรับค่า request และแสดง response จาก service server ชื่อ GetPosition

request : จะรับค่าเข้ามาในรูปแบบของ sensor_msgs/JointState ในที่นี่เราจะรับค่า request แค่ position ของแต่ละ joint (joint configuration) เมื่อรับค่าผ่านทาง service เข้ามาแล้วจะทำการเก็บค่าไว้ที่ตัวแปร joint_state_position ค่าที่รับเข้ามายังเป็น rad ของแต่ละ joint

response : เราจะ response ค่าที่เป็นตำแหน่งของ end-effector คือ x y z ดังนั้น เราจึงต้องนำค่า rad ที่รับเข้ามาไปทำการ forward kinematic เพื่อให้ได้ตำแหน่งของ end-effector เทียบกับฐานของหุ่นยนต์แขนกล เมื่อได้มาซึ่งตำแหน่ง x y z แล้ว ทำการสร้างตัวแปรชื่อ position เป็น massage ชนิด Point() ซึ่งจะทำการเก็บตำแหน่งของ end-effector เทียบกับฐานของหุ่นยนต์แขนกลไว้

Point() จะสามารถเก็บข้อมูลได้ดังต่อไปนี้

Compact Message Definition

```
float64 x
float64 y
float64 z
```

Forward kinematics

```
import numpy as np
import math

def forward_kinematics(q):
    joint_config = q
    n_joint = 3
    DH_pam = [[0 ,0 ,0.3 ,0],
               [0.16 ,math.pi/2 ,0 ,0],
               [0.35 ,0 ,0 ,0]]

    H = np.identity(4)

    H_end_eff = [[0 , 0 ,1 ,0.16],
                  [1 , 0 ,0 ,0],
                  [0 ,-1 ,0 ,0],
                  [0 , 0 ,0 ,1]]

    for i in range(n_joint):
        rad = joint_config[i]
        c ,s = np.cos(rad) , np.sin(rad)
        c_x ,s_x = np.cos(DH_pam[i][1]) , np.sin(DH_pam[i][1])
        c_z ,s_z = np.cos(DH_pam[i][3]) , np.sin(DH_pam[i][3])

        T_x = [[1 ,0 ,0 ,DH_pam[i][0]],
                [0 ,1 ,0 ,0],
                [0 ,0 ,1 ,0],
                [0 ,0 ,0 ,1]]

        R_x = [[1 , 0 , 0 ,0],
                [0 , c_x ,-s_x ,0],
                [0 , s_x , c_x ,0],
                [0 , 0 , 0 ,1]]

        T_z = [[1 ,0 ,0 ,0],
                [0 ,1 ,0 ,0],
                [0 ,0 ,1 ,DH_pam[i][2]],
```

```

[0 ,0 ,0 ,1]]

R_z = [[c_z ,-s_z ,0 ,0],
        [s_z , c_z ,0 ,0],
        [0 , 0 ,1 ,0],
        [0 , 0 ,0 ,1]]

H_j = [[c , -s ,0 ,0],
        [s , c ,0 ,0],
        [0 , 0 ,1 ,0],
        [0 , 0 ,0 ,1]]

H = H @ T_x @ R_x @ T_z @ R_z @ H_j

H = H@H_end_eff
return H

```

สร้างฟังก์ชัน forward_kinematics โดยมี input คือ q (joint configuration) ซึ่งเป็น array เข้ามา โดยที่ภายใน array ของ q จะเป็นค่าของมุมในแต่ละ joint (rad) ที่หมุนไป กำหนดให้ตัวแปร DH-param คือ ค่า DH-Parameter และกำหนดค่า H_end_eff คือ ค่า H_e^3 (homogeneous matrix transformation ของ end-effector เทียบกับ link_3) ทำการ for loop ตามจำนวน joint ซึ่งเรากำหนดให้มี 3 joint เพื่อทำการหาค่า H_i^{i-1}

โดยที่ $H_i^{i-1} = Trans_x(a_i) * Rot_x(\alpha_i) * Trans_z(d_i) * Rot_z(\theta_i) * H_j$
 ซึ่ง H_j คือการเคลื่อนที่ของตัวทำงานส่วนปลายของหุ่นยนต์ ซึ่งหุ่นยนต์ของเรามาเคลื่อนที่แบบ revolute ตามแกน z ดังนั้น $H_j =$

$Rot_z(q[i])$

ดังนั้น $H_e^3 = H_{end_eff} * H_i^{i-1}$ และส่งค่าอกมาเป็น matrix ของ H_e^3

ฟังก์ชัน set_joint_callback_ik (สำหรับ Inverse Kinematic)

```

# Inverse Kinematics
def set_joint_callback_ik(self,request:SolveIK.Request,response:SolveIK.Response):
    self.joint_state_position.position[0] = request.point.x
    self.joint_state_position.position[1] = request.point.y
    self.joint_state_position.position[2] = request.point.z
    gamma = request.gamma

    ik,flag = inverse_kinematics(self.joint_state_position.position, gamma)
    self.joint_state_position.position = [ik[0],ik[1],ik[2]]

    response.q = self.joint_state_position
    response.flag.data = flag

    return response

```

ฟังก์ชันนี้จะทำการรับค่า request และแสดง response จาก service server ชื่อ SolveIK

request : จะรับค่าเข้ามา 2 ค่า คือ geometry_msgs/Point ซึ่งจะรับค่า request เป็นตำแหน่ง x y z ของ end-effector
เทียบกับฐานของหุ่นยนต์แขนกล และ int32[2] ซึ่งเป็น array ของค่า arm configuration จะมีค่า 1 กับ -1 ซึ่งจะเป็นการกำหนด
ว่าแขนกลจะหันหน้า หันหลัง หรือ โค้งขึ้น โค้งลง
response : เราจะ response ค่าที่เป็นมุมของแต่ละ joint ที่ต้องทำเพื่อให้ได้ตำแหน่ง x y z ที่เรา request ไป ดังนั้น เราจึงต้อง¹
นำตำแหน่ง x y z ที่รับเข้ามาไปทำการ inverse kinematic เพื่อให้มุมของแต่ละ joint (rad) ที่ต้องหมุนซึ่งเป็นข้อความชนิด
JointState

Inverse kinematic

```

import numpy as np
import math

def inverse_kinematics(p,gamma):
    # setup parameters
    flag = True
    x = p[0]
    y = p[1]
    z = p[2]
    g1 = gamma[0]
    g2 = gamma[1]
    h = 0.3
    l_1 = 0.16

```

```

l_2 = 0.35
l_3 = 0.16

# Q1
s_1 = y/g1
c_1 = x/g1
q_1 = np.arctan2(s_1,c_1)

# Q3
c_3 = ((g1*((x**2+y**2)**0.5) - l_1)**2 + (z-h)**2 - l_2**2 - l_3**2)/(2*l_2*l_3)
# c_3 = round(c_3,13)
if -1 <= c_3 and c_3 <=1 :
    s_3 = g2*((1-c_3**2)**0.5)
    q_3 = np.arctan2(s_3,c_3)
else:
    flag = False
return [0.,0.,0.],flag

#Q2
c_2 = (l_2 + l_3*c_3)*(g1*((x**2+y**2)**0.5)-l_1) + (l_3*s_3)*(z-h)
s_2 = (-l_3*s_3)*(g1*((x**2+y**2)**0.5)-l_1) + (l_2 + l_3*c_3)*(z-h)
q_2 = np.arctan2(s_2,c_2)
return [q_1,q_2,q_3],flag

```

สร้างฟังก์ชัน inverse_kinematics ที่รับค่าตัวแปรสองตัวคือ

1. ค่า array ของ p ซึ่งเป็นค่าตำแหน่งของ end-effector เทียบกับฐานของหุนยนต์แขนกล ซึ่งภายใน array ของ p จะประกอบด้วย [x,y,z]
2. ค่า gamma ซึ่งมีค่า {-1,1}

ทำการแทนค่าตามสมการเพื่อหา Q1

ในการหาค่า Q2 กับ Q3 จะต้องทำการตรวจสอบว่าค่า c_3 มีค่าอยู่ระหว่าง -1 ถึง 1 หรือไม่ก่อนแทนค่าลงสมการ และสังเคราะห์มาเป็น array([q_1,q_2,q_3]) คือ mutable ที่ต้องทำเพื่อให้ได้ตามตำแหน่ง x,y,z ที่เรารับ request เข้ามา และ flag ที่จะเป็นตัวบอกว่าแขนของหุนยนต์สามารถเคลื่อนที่ไปยังตำแหน่งนั้นๆได้หรือไม่

CMakeList

```

# Install Python executables
install(PROGRAMS
    scripts/kinematics_server.py
    DESTINATION lib/${PROJECT_NAME}
)

#####
##### INSTALL LAUNCH, ETC #####
install(DIRECTORY
    # add directories here
    launch
    scripts
    romeona_kinematics
    DESTINATION share/${PROJECT_NAME})

ament_package()

```

ทำการ add kinematic_server.py และ folder ที่อยู่ใน package นี้ลงไป

Package : romeona_kinematics_interfaces

ภายใน package นี้จะมี folder srv ซึ่งเก็บไฟล์ GetPosition.srv และ SolveIK.srv โดยที่ทั้งคู่คือ Service server

GetPosition.srv

```

# request
sensor_msgs/JointState q
---

# response
geometry_msgs/Point p

```

สร้าง service server ชื่อ GetPosition.srv ซึ่งอยู่ใน romeona_kinematics_interfaces/srv/GetPosition.srv เพื่อเป็นตัวส่งค่าให้กับฟังก์ชัน set_joint_callback โดยสร้าง request ของตัว service เป็นชนิด sensor_msgs/JointState ชื่อ q ซึ่งจะใช้เป็น

ตัวเก็บค่าของมุมในแต่ละ joint และสร้าง response ของตัว service เป็นชนิด geometry_msgs/Point ซึ่ง p ซึ่งเป็นตัวส่งค่า ตำแหน่ง x,y,z ของ end-effector เทียบกับฐานของหุ่นยนต์แขนกล

SolveIK.srv

```
geometry_msgs/Point point
int32[2] gamma
---

sensor_msgs/JointState q
std_msgs/Bool flag
```

สร้าง service server ชื่อ SolveIK.srv ซึ่งอยู่ที่ไฟล์เดอร์เดียวกัน เพื่อใช้เป็นตัวส่งค่าให้กับฟังก์ชัน set_joint_callback_ik โดยสร้าง request ของตัว service 2 ตัว คือ

1. ชนิด geometry_msgs/Point ชื่อ point
2. ชนิด int32[2] ชื่อ gamma

และสร้าง response ของตัว service 2 ตัว คือ

1. ชนิด sensor_msgs/JointState ชื่อ q
2. ชนิด std_msgs/Bool ชื่อ flag

CMakeList

```
find_package(rosidl_default_generators REQUIRED)
find_package(geometry_msgs)
find_package(std_msgs)
find_package(sensor_msgs)

rosidl_generate_interfaces(${PROJECT_NAME}
    "srv/GetPosition.srv"
    "srv/SolveIK.srv"
    DEPENDENCIES geometry_msgs
    DEPENDENCIES std_msgs
    DEPENDENCIES sensor_msgs
```

เพิ่ม package ของ message ที่เราใช้ใน service เข้าไปคือ

- geometry_msgs
- sensor_msgs

เพิ่มไฟล์ service server เข้าไปคือ

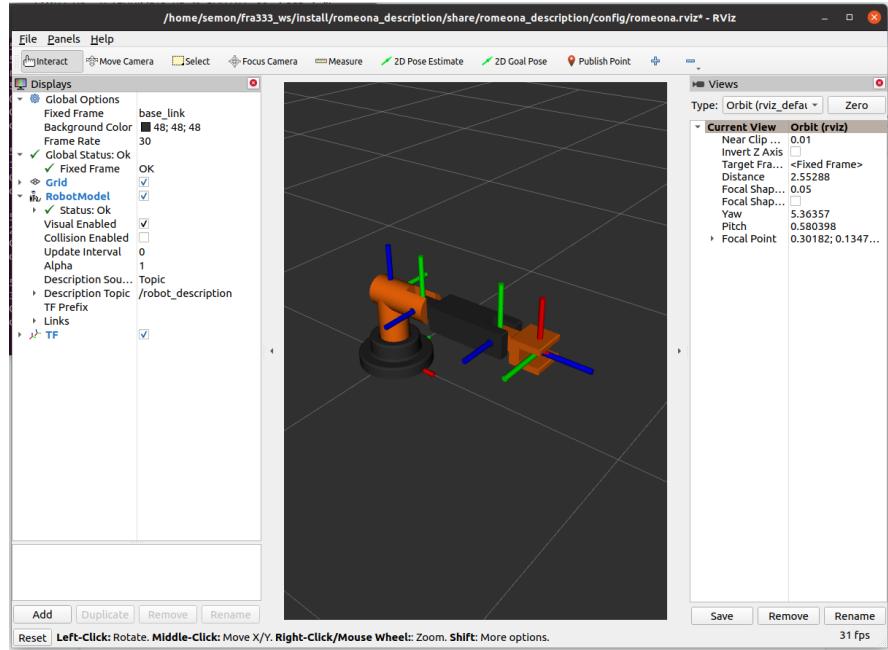
- GetPosition.srv
- SolveIK.srv

Launch Package

เมื่อ launch package ด้วย command

-> *ros2 launch romeona_kinematics display_with_kinematics_server.launch.py*

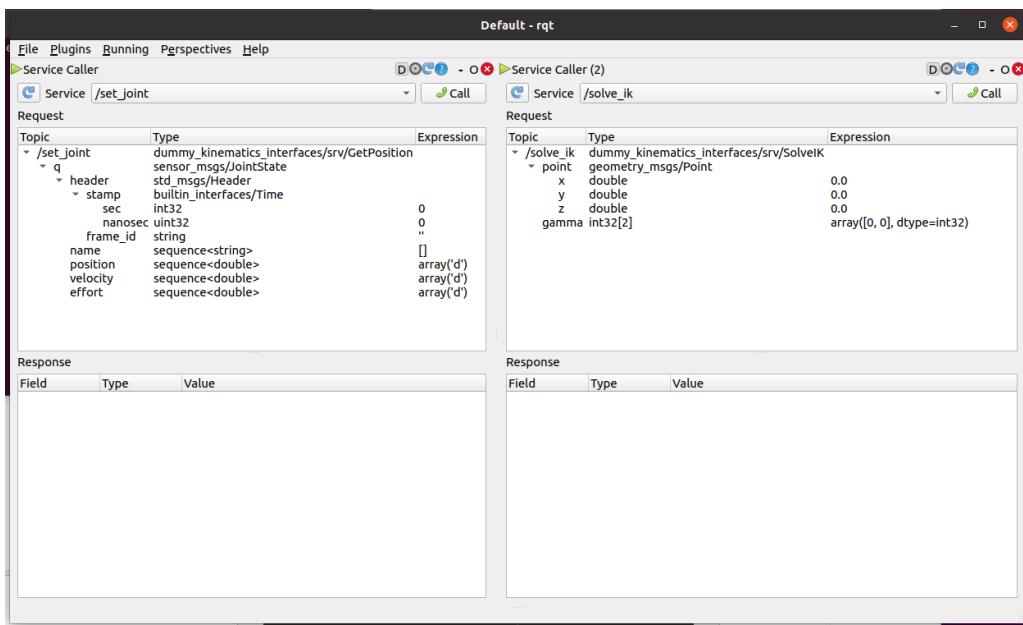
จะปรากฏหน้าต่าง rviz ของหุ่นยนต์แขนกลของเราระบุมา ดังนี้



และเราสามารถ call service ผ่าน command

-> rqt

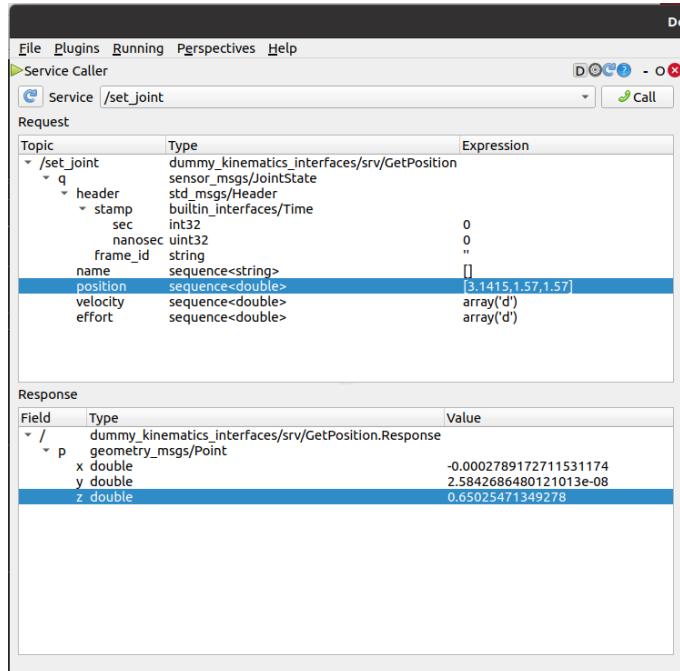
จะปรากฏหน้าต่าง ดังนี้



จะเลือก service `/set_joint` ที่เป็นการ request ค่าเป็นมุม (rad) ของแต่ละ joint (q) ที่จะนำค่าดังกล่าวไปผ่านการทำ Forward Kinematic เพื่อให้ได้ตำแหน่งของ end-effector เทียบกับฐานของหุ่นยนต์ (x, y, z) และเพิ่ม service `/solve_ik` ที่เป็นการ request ค่าเป็นตำแหน่ง end-effector เทียบกับฐานของหุ่นยนต์ (x, y, z) และ pose (γ) นำค่าดังกล่าวไปผ่านการทำ Inward Kinematic เพื่อให้ได้มุม (rad) ของแต่ละ joint (q) ที่ต้องทำ เพื่อให้ได้ตำแหน่งที่ต้องการและมี

ค่า flag ที่จะ return ออกมามาเป็น True กับ False (*True* กรณีที่หุ่นยนต์แขนกลสามารถไปที่ตำแหน่งนั้นได้ และ *False* กรณีที่หุ่นยนต์แขนกลไม่สามารถไปที่ตำแหน่งนั้นได้)

การใช้ service



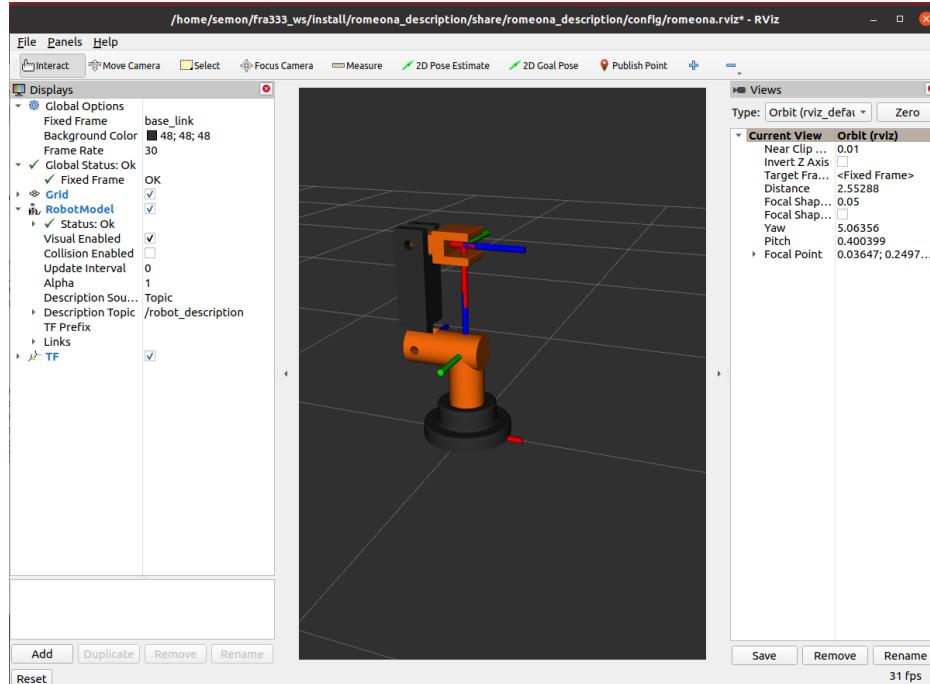
ใน service */set_joint* ใส่ค่า \hat{q} ดังนี้ $[3.1415, 1.57, 1.57]$ ซึ่งเป็นมุมของแต่ละ joint ในหน่วย *rad* เมื่อ call แล้วจะได้ตำแหน่งของ end-effector เทียบกับฐานของหุ่นยนต์ (x, y, z) ผ่านทาง response ดังนี้

$$x = -0.0002789172711531174 \text{ m}$$

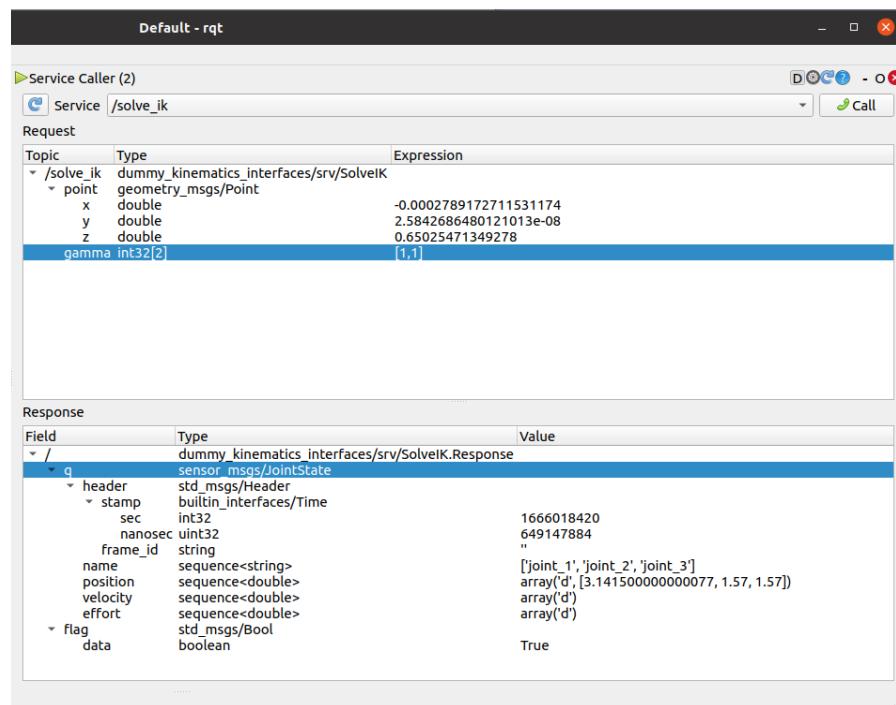
$$y = 2.5842686480121013e - 08 \text{ m}$$

$$z = 0.65025471349278 \text{ m}$$

หุ่นยนต์แขนกลจะมีลักษณะดังภาพนี้



ใน service `/solve_ik` นำค่า (x , y , z) ซึ่งได้จากการทำ Forward Kinematic ไปท่า Inverse Kinematic ผ่าน service นี้ และใส่ค่า γ ด้วยค่า [1,1] จะได้ผลดังนี้



จะได้รูปที่แต่ละ joint ต้องทำได้ผ่าน response ดังนี้

-> `array('d', [3.141500000000077, 1.57, 1.57])` ซึ่งหมายความว่า

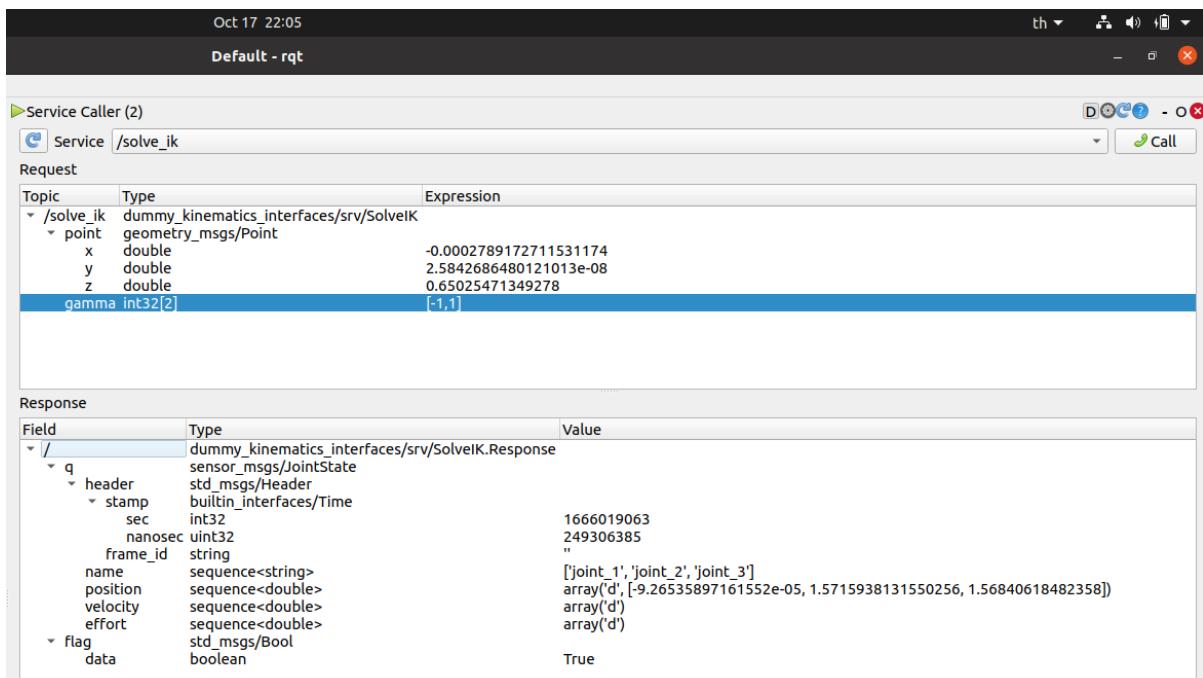
$$q_1 = 3.141500000000077 \text{ rad}$$

$$q_2 = 1.57 \text{ rad}$$

$$q_3 = 1.57 \text{ rad}$$

ซึ่งมีค่าตรงกับที่เราป้อนค่าไปใน service /set_joint หุ่นยนต์จะมี pose เหมือนเดิม

แต่ถ้าใส่ค่า γ ตัวย่อ [-1,1] จะได้ผลดังนี้



จะได้รูปที่แสดง joint ต้องทำได้ผ่าน response ดังนี้

$\rightarrow \text{array}('d', [-9.26535897161552e-05, 1.5715938131550256, 1.56840618482358])$ ซึ่งหมายความว่า

$$q_1 = -9.26535897161552e-05 \approx 0 \text{ rad}$$

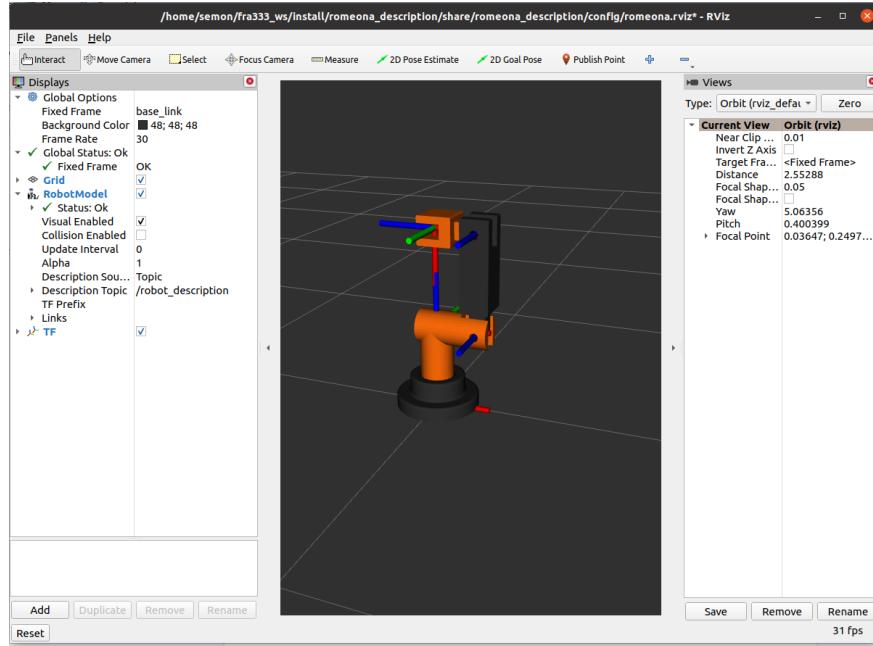
$$q_2 = 1.5715938131550256 \approx 1.57 \text{ rad}$$

$$q_3 = 1.56840618482358 \approx 1.57 \text{ rad}$$

และ

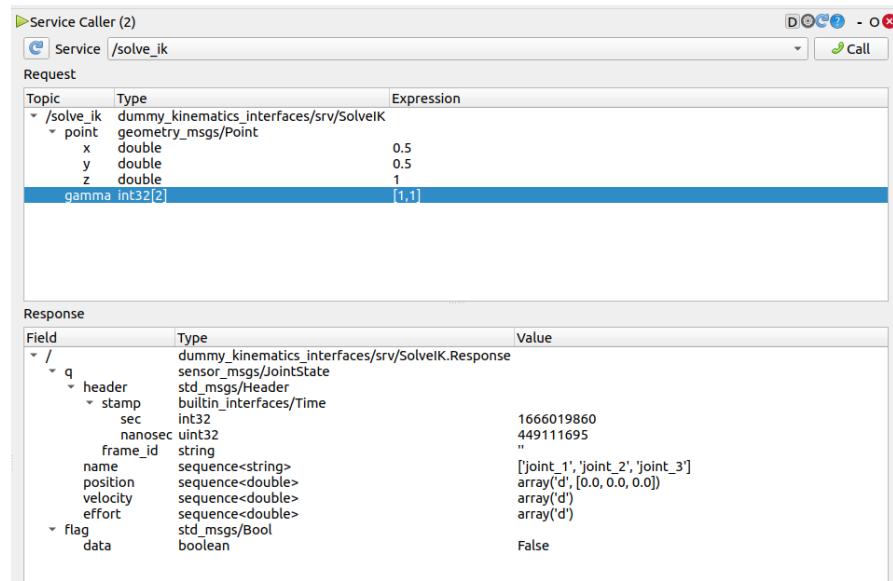
$flag = True$ ซึ่งแสดงว่าหุ่นยนต์แขนกลสามารถทำงานใน task space นี้ได้

หุ่นยนต์จะมี pose ที่เปลี่ยนแปลงไป ดังนี้

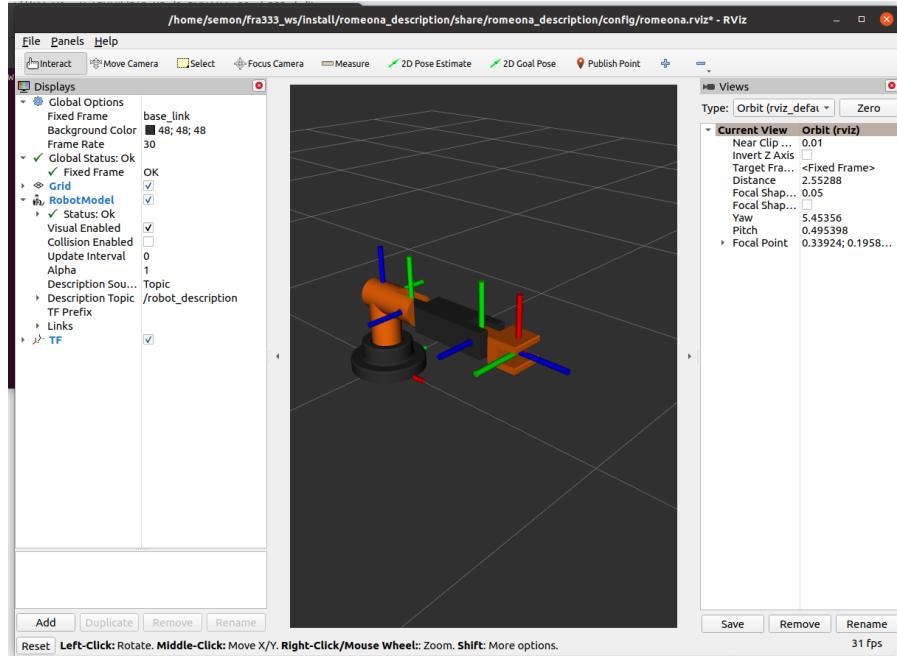


ที่ link_1 จะถูกหมุนไปด้วย 3.14 rad ทำให้หุ่นยนต์แขนกลมีทิศทางที่กลับด้านเมื่อเทียบกับก่อนหน้านี้ เนื่องจากการใส่ค่า γ เป็นการกำหนด pose ของหุ่นยนต์แขนกลนั้นเอง

แต่ถ้าเราใส่ค่าตำแหน่ง (x, y, z) ใน service `/solve_ik` ที่หุ่นยนต์แขนกลไม่สามารถทำ task นั้นได้ จะได้ผลลัพธ์ดังนี้
เช่น ใส่ค่า $(x, y, z) = (0.5, 0.5, 1)$ จะได้ผลลัพธ์ดังนี้



ซึ่งหุ่นยนต์แขนกลไม่สามารถทำงานใน task space นี้ได้ เราจึงกำหนดให้หุ่นยนต์แขนกลกลับมาที่ Initial position ตามโจทย์และ trigger $flag = False$ ซึ่งจะได้ pose ของหุ่นยนต์แขนกลดังนี้



ดังนั้นจะสรุปได้ว่า หุ่นยนต์แขนกลจะมี task space ที่สามารถทำงานได้อยู่ ในการใส่ค่าเพื่อทำ Forward kinematic เราสามารถใส่ค่า (x, y, z) ที่เกินขอบเขตการทำงานของหุ่นยนต์ แขนกลได้เนื่องมาจากข้อจำกัดด้านความยาวของ link นั้นเอง