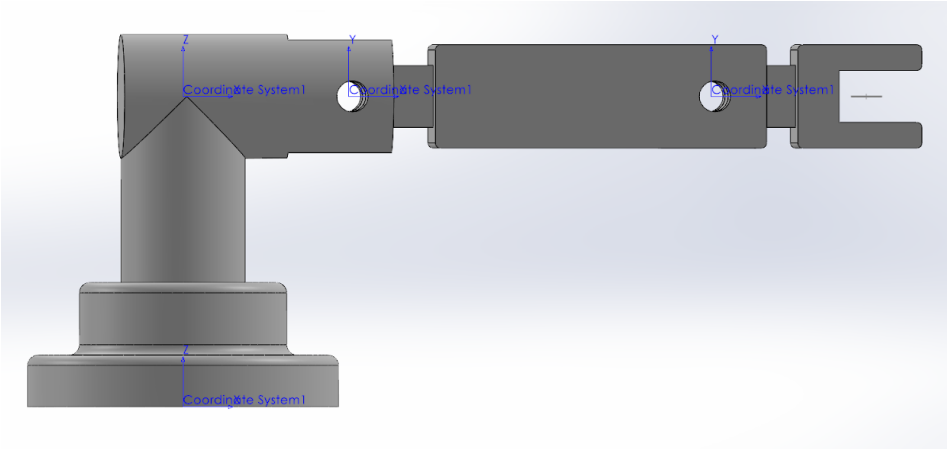
LAB3 (Romeona Robot Gazabo)

Group 23

การสร้างแบบจำลองหุ่นยนต์ที่เป็นแขนกล

หุ่นยนต์แขนกลที่สร้างไว้เป็น 3 DOF และมีการกำหนด Coordinate frame ของแต่ละ Joint โดยมีรูปดังนี้



$i-1$	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

$$\begin{bmatrix} 0 & 0 & 1 & 0.16 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_e^3 =$$

โดย type of joint ทั้งหมดเป็น revolute joint

Gazebo และการแสดงผลการจำลองหุ่นยนต์

ในการจำลองหุ่นยนต์แขนกล 3 DOF ใน Gazebo เราจึงเขียนไฟล์ URDF เพื่อให้ CAD ใน Gazebo สามารถแสดงรูปร่างและการเคลื่อนไหวได้อย่างถูกต้อง

ในการสร้าง coordinate frame เราจะสร้างไว้ที่ joint ใน CAD ดังนั้น เราจึงต้องกำหนด <origin> ของแต่ละ joint 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 และไม่มีการหมุน 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 _3 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 เรากำหนดให้แกนของ end_effector มีแกน x ชี้ไปข้างหน้า แกน z ชี้ขึ้น และแกน y ชี้ออกมาจากหน้าจอ ดังนั้นจึงต้องทำการการหมุนตามแกน 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 ”

ซึ่งในการจำลองหุ่นยนต์ใน Gazebo จะมีการจำลองตามหลักฟิสิกส์ ดังนั้นจึงจำเป็นต้องใส่ค่า mass และค่า inertia ของแต่ละ link ให้ถูกต้อง ดังนี้

```
  <link name="link_0">
            <inertial>
```

```
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <mass value="4.91" />
            <inertia ixx="0.03" ixy="0.00" ixz="0.00" iyy="0.03"
iyz="0.00" izz="0.05" />
        </inertial>
    </link>
```

หลังจากใช้ Solidworks ในการคำนวณหาค่า mass และค่า inertia ของ link 0 จะได้ค่ามวลเท่ากับ 4.91 kg และได้ค่าความเฉื่อย ixx = 0.03 ixy = 0.00 ixz = 0.00 iyy = 0.03 iyz = 0.00 izz = 0.05

```
<link name="link_1">
              <inertial>
            <origin xyz="0 0 0.0" rpy="0 0 0"/>
            <mass value= "4.56" />
            <inertia ixx="0.04" ixy="0.00" ixz="0.01" iyy="0.05"
iyz="0.00" izz="0.02" />
        </inertial>
    </link>
```

หลังจากใช้ Solidworks ในการคำนวณหาค่า mass และค่า inertia ของ link 1 จะได้ค่ามวลเท่ากับ 4.56 kg และได้ค่าความเฉื่อย ixx = 0.04 ixy = 0.00 ixz = 0.01 iyy = 0.05 iyz = 0.00 izz = 0.02

```
<link name="link_2">
        <inertial>
            <origin xyz="0 0 0.0" rpy="0 0 0"/>
            <mass value="3.56" />
            <inertia ixx="0.04" ixy="0.00" ixz="0.00" iyy="0.01"
iyz="0.00" izz="0.04" />
        </inertial>
    </link>
```

หลังจากใช้ Solidworks ในการคำนวณหาค่า mass และค่า inertia ของ link 2 จะได้ค่ามวลเท่ากับ 3.56 kg และได้ค่าความเฉื่อย ixx = 0.04 ixy = 0.00 ixz = 0.00 iyy = 0.01 iyz = 0.00 izz = 0.04

```
  <link name="link_3">
        <inertial>
            <origin xyz="0 0 0.0" rpy="0 0 0"/>
            <mass value="1.23" />
```

```
            <inertia ixx="0.00" ixy="0.00" ixz="0.00" iyy="0.00"
iyz="0.00" izz="0.00" />
        </inertial>
    </link>
```

หลังจากใช้ Solidworks ในการคำนวณหาค่า mass และค่า inertia ของ link 3 จะได้ค่ามวลเท่ากับ 1.23 kg และได้ค่าความเฉื่อย ixx = 0.00 ixy = 0.00 ixz = 0.00 iyy = 0.00 iyz = 0.00 izz = 0.00

การรับค่า Imu

ในการรับค่า Imu เราจะใช้ xicro package ในการเชื่อมต่อ arduino เข้ากับ ros2 โดยทำการ generate library หลังจากนั้นทำการ generate node แล้วอัปโหลด code ผ่าน arduino **ซึ่งค่าผลลัพธ์ที่ได้ออกมาเป็น** angular_velocity ซึ่งเป็นความเร็วเชิงมุม และ linear_acceleration ซึ่งเป็นค่าความเร่ง โดยค่าที่ได้จะถูกรับค่าไปใช้ใน Node ที่ชื่อ Trajectory_publisher ผ่านตัว subscription

```
- 0.0
- 1.0
angular_velocity:
  x: 0.051132682710886
  y: -0.023435812443494797
  z: -0.04048004001379013
angular_velocity_covariance:
- 1.0
- 0.0
- 0.0
- 0.0
- 1.0
- 0.0
- 0.0
- 0.0
- 1.0
linear_acceleration:
  x: 0.1149216815829277
  y: -0.1472434103488922
  z: -9.97065258026123
linear_acceleration_covariance:
- 1.0
- 0.0
- 0.0
- 0.0
```

การทำ Calibrate

ค่าความเร่งที่ได้จาก Imu เมื่อมีการแปลงค่าเป็นตำแหน่ง จะพบว่ามีค่า noise สะสมไปเรื่อยๆ ดังนั้นเราจึงต้องทำการ Calibrate เพื่อลดค่า noise

โดยเราจะทำภายใน Node ชื่อ Trajectory_publisher

```python
class Trajectory_publisher(Node):
    def __init__(self):
        super().__init__('trajectory_publsiher_node')
        publish_topic = "/joint_trajectory_position_controller/joint_trajectory"

        # create publisher and publishtopic to gazebo
        self.trajectory_publihser = self.create_publisher(JointTrajectory,publish_topic, 10)

        # subscription from IMU
        self.imu_sub = self.create_subscription(Imu,'/Imu_arduino',self.subscription_callback,10)
```

```
        # timer
        self.timer_period = 0.1
        self.timer = self.create_timer(self.timer_period, self.timer_callback)
```

สร้าง subscription เพื่อรับค่า Imu_arduinoซึ่งเป็นข้อความประเภท Imu และเก็บข้อมูลผ่านฟังก์ชัน subsription_callback

ซึ่งภายในฟังก์ชั่น subscription จะประกอบไปด้วย การทำ integral ความเร่งเชิงเส้น เพื่อหาความเร็วเชิงเส้น แล้ว integral ความเร็วเชิงเส้นหาระยะเชิงเส้น และการทำ Calibrate

สร้าง publisher โดยมี topic ชื่อ publish_topic และส่ง massage ชนิด JointTrajectory โดยในการ publish เราจะทำการส่งค่าครั้งละ 0.1 second ผ่านฟังก์ชั่น timer_callback

```
# user variables
        self.joints = ['joint_1','joint_2','joint_3']
        self.start_positions = [0.0, 0.0, 0.0]
        self.goal_positions = [0.0, 0.0, 0.0]
        self.setpoint_position = self.start_positions
        self.i = 0
        self.vel_init = [0.0, 0.0, 0.0]
        self.pos_init = [0.0, 0.0, 0.0]
        self.is_cal = True
        self.cal_ang = [0, 0, 0]
        self.cal_acc = [0, 0, 0]

    def subscription_callback(self,msg):

        self.imu_angular_vel = [msg.angular_velocity.x - self.cal_ang[0],
                                msg.angular_velocity.y - self.cal_ang[1],
                                msg.angular_velocity.z - self.cal_ang[2]]
        self.imu_linear_acc = [msg.linear_acceleration.x - self.cal_acc[0],
                               msg.linear_acceleration.y - self.cal_acc[1],
                               msg.linear_acceleration.z - self.cal_acc[2]]
        if self.is_cal:
            self.is_cal, self.cal_ang, self.cal_acc =
call_json(self.imu_angular_vel, self.imu_linear_acC)
```

ทำการกำหนดตัวแปรขึ้นมาเพื่อใช้เก็บข้อมูล โดยกำหนดให้ is_cal = True , cal_ang = [0,0,0] ซึ่งกำหนดให้เป็น array ที่เก็บค่าเฉลี่ยของความเร็วเชิงมุมของทั้ง 3 แกน , cal_acc = [0,0,0] ซึ่งกำหนดให้เป็น array ที่เก็บค่าเฉลี่ยของความเร่งของทั้ง 3 แกน

ซึ่งภายในฟังก์ชัน subsription_callback ทำกับรับข้อความเข้ามาที่เป็นประเภท Imu() ซึ่งสามารถเก็บค่าได้ดังนี้

```
std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
```

โดยทำการเก็บค่า angular_velocity - cal_ang ซึ่งเป็นความเร็วเชิงมุมของแต่ละแกนลบกับความเร็วเชิงมุมเฉลี่ยของแต่ละแกน

แล้วทำการเก็บไว้ในตัวแปร imu_angular_vel

และทำการเก็บค่า linear_acceleration - cal_acc ซึ่งเป้นความเร่งของแต่ละแกนลบกับความเร่งเฉลี่ยของแต่ละแกน

แล้วทำการเก็บไว้ในตัวแปร imu linear_acc

ถ้า is_cal มีค่าเท่ากับ True จะส่งค่า imu_angular_vel และ imu linear_acc เข้าไปในฟังก์ชั่น call_json

```
def call_json(l1,l2):
    flag = True
    cal_an = [0.0, 0.0, 0.0]
    cal_ac = [0.0, 0.0, 0.0]
    # read json
    f = open('data.json')

    try :
        data = json.load(f)
        if len(data['angular']) >= 1000:
            flag = False
            for i in range(len(cal_ac)):
                cal_an[i] =
sum(np.array(data['angular'])[:,i])/len(np.array(data['angular'])[:,i])
                cal_ac[i] =
sum(np.array(data['linear'])[:,i])/len(np.array(data['linear'])[:,i])
            # print(cal_ac, cal_an)
        else:
            data["angular"].append(l1)
            data["linear"].append(l2)
    except JSONDecodeError:
        data = {"angular": [l1], "linear": [l2] }

    f.close()
    # Data to be written
    with open("data.json", "w") as outfile:
        json.dump(data, outfile)

    print(len(data["linear"]))
    return flag, cal_an, cal_ac
```

โดยภายในฟังก์ชัน call_json จะทำการอ่านค่าในไฟล์ json ที่เป็น angular เมื่ออ่านค่าได้เท่ากับ 1000 ตัวจะกำหนดค่า flag = False แล้วจะทำการหาค่าเฉลี่ยของความเร็วเชิงมุม และ ค่าเฉลี่ยของความเร่ง และทำการเก็บไว้ในตัวแปร cal_an และตัวแปร cal_ac ตามลำดับ แล้วทำการ return ค่าออกมา 3 ตัวได้แก่ flag , cal_an, cal_ac

```
if self.is_cal:
            self.is_cal, self.cal_ang, self.cal_acc =
call_json(self.imu_angular_vel, self.imu_linear_acc)
```

โดยค่าที่ return ออกมา 3 ตัวได้แก่ flag , cal_an, cal_ac จะถูกเก็บไว้ในตัวแปร is.cal , cal.ang และ cal_acc ตามลำดับ

Integral acceleration and velocity

```
else:
    # self.imu_angular_vel = list(map(lambda x:round(x,2), self.imu_angular_vel))

    vel = get_linear_vel(self.imu_linear_acc, self.vel_init, self.timer_period)
    scale = 0.2
    self.vel_init[0] = vel[0]*scale
    self.vel_init[1] = vel[1]*scale
    self.vel_init[2] = vel[2]*scale
    # print('v =', self.vel_init)

    # pos = get_position(self.vel_init, self.timer_period)
    # self.pos_init[0] += round(pos[0],2)
    # self.pos_init[1] += round(pos[1],2)
    # self.pos_init[2] += round(pos[2],2)


    # stupid ver.
    pos = get_position(self.imu_angular_vel , self.timer_period)
    self.pos_init[0] += round(pos[0],2)
    self.pos_init[1] += round(pos[1],2)
    self.pos_init[2] += round(pos[2],2)
```

Function get_linear_vel จะรับความเร่งเชิงเส้น ความเร็วเริ่มต้น และระยะเวลา เพื่อ integral หาความเร็วเชิงเส้น

```python
def get_linear_vel(acc, v_0, dt):
    v = [0, 0, 0]
    for i in range(len(v_0)):
        v[i] = v_0[i] + acc[i]*dt

    return v
```

Function get_linear_vel จะรับความเร็วเชิงเส้น และระยะเวลา เพื่อ integral หาความระยะเชิงเส้น

```python
def get_position(vel, dt):
    p = [0, 0, 0]
    for i in range(len(vel)):
        p[i] =  vel[i]*dt

    return p
```

```python
self.goal_positions = self.pos_init.copy()
```

บันทึกข้อมูลตำแหน่งของ pos_init.copy() ใส่ goal_positions

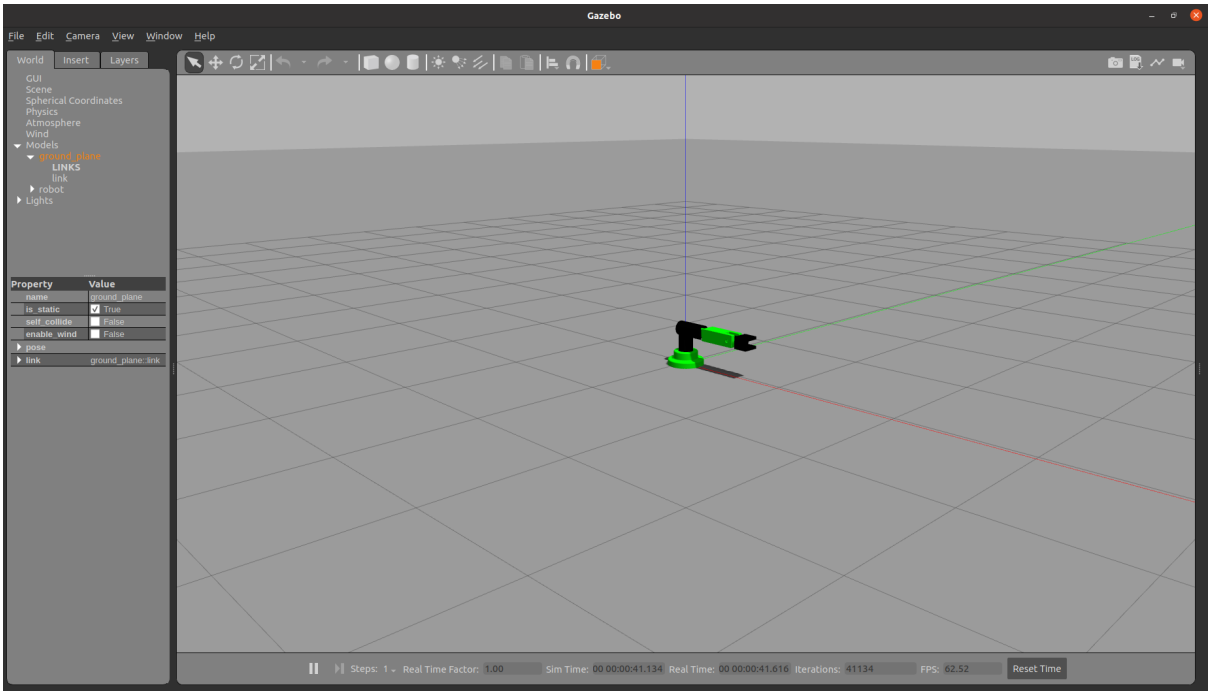
Publish message

```python
def timer_callback(self):


    self.setpoint_position = self.goal_positions

    bazu_trajectory_msg = JointTrajectory()
    bazu_trajectory_msg.joint_names = self.joints
    ## creating a point
    point = JointTrajectoryPoint()
    # point.positions = self.self.cur_pos
    point.positions = self.setpoint_position
    point.time_from_start = Duration(sec=1)
    ## adding newly created point into trajectory message
    bazu_trajectory_msg.points.append(point)
    # point.positions = self.goal_positions
    # point.time_from_start = Duration(sec=8)
    # bazu_trajectory_msg.points.append(point)
    self.trajectory_publihser.publish(bazu_trajectory_msg)
    # print(self.setpoint_position)
```

Function timer_callback จะบันทึกค่า goal_positions ใส่ setpoint_position โดยจะกำหนดให้ bazu_trajectory_msg.joint_names ตามชื่อ joints ที่ได้กำหนด และ กำหนดให้ point.positions ประเภท JointTrajectoryPoint() เป็น self.setpoint_position จากนั้นส่ง message bazu_trajectory_msh ที่ถูกเพิ่ม ข้อมูลประเภท point เข้าไปใน Gazebo เพื่อแสดงผล


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
ground_plane
LINKS
link
robot
Lights
Property Value
name ground_plane
is_static True
self_collide False
enable_wind False
pose
link ground_plane::link
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:41.134 Real Time: 00 00:00:41.616 Iterations: 41134 FPS: 62.52
Reset Time