Alumno: Flores Pradel Mauricio
Materia: Robótica
Grupo:01

Comparativa de códigos scara_tray_line_py.py y dofbot_sequence_py.py

Dofbot_sequence_py.py:

Comenzando con el análisis para el código dofbot_sequence_py.py, el código define un nodo ROS 2 llamado dofbot_tray_control_node, encargado de controlar tanto los motores articulares del brazo como el gripper (pinza).

Se publica en dos tópicos:

- /dofbot_trajectory_controller/joint_trajectory → controla los 5 ejes del brazo.
- /dofbot_gripper_controller/joint_trajectory → controla la apertura/cierre del gripper.

El nodo ejecuta una secuencia automatizada de movimientos (trayectoria predefinida) que simula una tarea de “tomar un objeto y colocarlo en otro punto”.

En la clase principal a lo referido como:

```
class DofbotControlNode(Node):
    def __init__(self):
        super().__init__("dofbot_tray_control_node")
```

Crea el nodo con el nombre que se ve en la imagen e inicializa variables, tópicos y temporizadores

```
topic_dofbot_ = "/dofbot_trajectory_controller/joint_trajectory"
topic_gripper_ = "/dofbot_gripper_controller/joint_trajectory"
self.dofbot_publisher_ = self.create_publisher(
    JointTrajectory, topic_dofbot_, 10)
self.dofbot_joints_ = ['arm_joint_01', 'arm_joint_02',
                       'arm_joint_03', 'arm_joint_04', 'arm_joint_05']

self.gripper_publisher_ = self.create_publisher(
```

En este apartado se crea dos publicadores con colas de 10 mensajes, mientras que los mensajes JointTrajectory contiene una lista de nombres de articulaciones y una lista de puntos (refereidos a las posturas)

```
self.dofbot_joints_ = ['arm_joint_01', 'arm_joint_02',
                       'arm_joint_03', 'arm_joint_04', 'arm_joint_05']

self.gripper_publisher_ = self.create_publisher(
    JointTrajectory, topic_gripper_, 10)
self.gripper_joints_ = ['grip_joint', 'rfinger_joint_01',
                        'rfinger_joint_02', 'lfinger_grip_joint_01',
                        'lfinger_grip_joint_02', 'lfinger_grip_joint_03']
```

En esta parte del código podemos observar las listas de articulaciones, las cuales deben coincidir con los nombres definidos en el controlador del hardware del robot

De igual manera tenemos un temporizador, cada 0.5 segundos se llama a timer_callback(). Sin embargo tambien existen pauesas con time.sleep().

Continuando con el código observamos una función llamada timer_callback(). Este método ejecuta una máquina de estados mediante la variable self.lamda_ que avanza de 0 a 9. Donde cada valor representa una etapa diferente del movimiento del robot.

Estado	Acción

0	Abrir gripper
1	Cerrar gripper
2	Volver a abrir gripper
3	Primer postura
4	Cerrar gripper
5	Segunda postura
6	Tercera postura
7	Bajar brazo
8	Abrir gripper
9	Posición de reposo

Donde cada bloque:

- Crea mensajes JointTrajectory y JointTrajectoryPoint.
- Define posiciones de articulaciones.
- Publica el mensaje en el tópico correspondiente.
- Espera unos segundos (time.sleep) para permitir el movimiento.

Para la función definida como dofbot_ink(), es referida a la cinemática inversa del robot dofbot.

```
def dofbot_ink(x_P, y_P, z_P, theta_1_P, theta_g):
    # Parametros
    z_0_1 = 0.105
    L_1 = 0.084
    L_2 = 0.084
    L_3 = 0.115
```

Cálcula los ángulos de las articulaciones dadas tomando en cuenta una posición desdeada del efector final, la orientacion del efector y el ángulo de la garra. De igual manera se define las longitudes de eslabones del brazo.

El método aplica fórmulas trigonométricas:

- theta_1 = atan2(y_P, x_P) calcula el giro de la base.
- Usa ley de cosenos y senos (acos, asin) para obtener theta_2, theta_3, theta_4.
- Devuelve un vector de 5 ángulos [θ1, θ2, θ3, θ4, θ5].

De igual manera tenemos una función gripper_state(theta), la cual devuelve las posiciones angulares necearias para abrir o cerrar la pinza, donde los pares de dedos del gripper giran en direcciones opuestas para abrir/cerrar simétricamente.

Finalmente la función principal main(). Inicializa ROS 2, ejecuta el nodo y espera eventos (mantiene el programa activo). Lo que podemos concluir es que el código demuestra el control por trayecotiras articuladas de un manipulador tipo DOFBOT.

Alumno: Flores Pradel Mauricio
Materia: Robótica
Grupo:01

scara_tray_line_py.py:

Este código corresponde a un nodo ROS 2 que controla un robot tipo SCARA (Selective Compliance Assembly Robot Arm). Este programa define un nodo llamado scara_tray_line_node que genera una trayectoria en línea recta para un robot SCARA.

El nodo:

- Calcula la cinemática inversa del SCARA.
- Publica los ángulos articulares que permiten que el efector final se mueva desde un punto inicial hasta un punto final de manera lineal y suave en el espacio cartesiano.

```
super().__init__("scara_tray_line_node")
topic_name = "/scara_trajectory_controller/joint_trajectory"
self.joints_ = ['link_1_joint', 'link_2_joint', 'link_3_joint']
```

Se crea un constructor, define el nodo con el nombre que se muestra en la imagen, publica en el tópico donde escucha el controlador del SCARA y declara las tres articulaciones del robot, dos rotacionales (link 1 y 2) y una lineal (link 3).

```
self.lamda_ = 0
self.Tiempo_ejec_ = 10
```

Se colocan parámetro de ejecución, donde tenemos a lamda_ que es el parámetro incremental que controla el avance de la trayectoria, mientras que Tiempo_ejec_ es el número total de pasos en los que se completará la trayectoria.

```
self.scara_tray_pub_ = self.create_publisher(
    JointTrajectory, topic_name, 10)
self.tray_timer_ = self.create_timer(
    1, self.trayectory_cbck)
```

De igual manera tenemos un publicador y un temporizador, donde el publicador se encarga de enviar comandos de trayectoria, mientras que el temporizador ayuda a llamar cada segundo a la función trayectory_cbck() para publicar la siguiente postura intermedia.

Comenzando con las funciones tenemos:

```
def trayectory_cbck(self):
    trayectory_msg = JointTrajectory()
    trayectory_msg.joint_names = self.joints_
    point = JointTrajectoryPoint()
```

La creación del mensaje ROS, el cual prepara el mensaje con los nombres de las articulaciones, de igual manera crea un punto de trayectoria.

Alumno: Flores Pradel Mauricio
Materia: Robótica
Grupo:01

```
if self.lamda_ <= self.Tiempo_ejec_:
    x_1 = 0.1
    y_1 = 0.6
    theta_1 = 0
    x_2 = 0.3
    y_2 = -0.6
    theta_2 = 1.57
```

Se tiene un movimiento lineal, donde se define el punto inicial (x_1, y_1, θ_1) y el punto final (x_2, y_2, θ_2). El robot se moverá desde el extremo superior izquierdo hasta el inferior derecho del plano de trabajo.

```
point.positions = solucion
point.time_from_start = Duration(sec=1)
trayectory_msg.points.append(point)
self.scara_tray_pub_.publish(trayectory_msg)
```

Posteriormente tenemos la publicación, donde se asigna las posiciones angulares calculadas, indica que debe ejecutarse en 1 segundo y publica el mensaje al controlador del SCARA.

```
time.sleep(2)
self.lamda_ += 1
```

Se genera un avance temporal, el caul pausa 2 segundos antes del seguimiento del movimiento e incremencra el parámetro para avanzar al siguiente punto de línea.

Posteriormente tenemos la función invk_sol() referida a la cinemática del SCARA, donde primero se determina los ángulos de las articulaciones para que el efector final del SCARA siga una linea recta en el plano XY. Se generan parámetros con las longitudes de los eslabones y se define el tiempo de ejecución. Se hace una interpolación lineal para obtener un punto intermedio entre el inicio y el final. Se realiza la cinemática inversa.

```
def invk_sol(param,x_in, y_in, theta_in, x_fin, y_fin, theta_fin):
    Tiempo_ejec_ = 10
    L_1 = 0.5
    L_2 = 0.5
    L_3 = 0.3
    x_P = x_in + (param/Tiempo_ejec_)*(x_fin - x_in)
    y_P = y_in + (param/Tiempo_ejec_)*(y_fin - y_in)
    theta_P = theta_in + (param/Tiempo_ejec_)*(theta_fin - theta_in)
    x_3 = x_P - L_3*cos(theta_P)
    y_3 = y_P - L_3*sin(theta_P)
    theta_2 = acos((pow(x_3, 2)+pow(y_3,2)-pow(L_1, 2)-pow(L_2, 2))/(2*L_1*L_2))
    beta = atan2(y_3, x_3)
    psi = acos((pow(x_3, 2)+pow(y_3,2)+pow(L_1, 2)-pow(L_2, 2))/(2*L_1*sqrt(pow(x_3, 2)+pow(y_3,2))))
    theta_1 = beta - psi
    theta_3 = theta_P -theta_1 -theta_2
    return [float(theta_1), float(theta_2), float(theta_3)]
```

Por ultimo en la función principal, se inicializa el entorno ROS 2, crea el incio del nodo y entra en un bucle de espera para mantener el nodo activo.

Alumno: Flores Pradel Mauricio
Materia: Robótica
Grupo:01

Comparativa entre los dos códigos

Aspecto	Scara_tray_line_py.py	Dofbot_sequence_py.py
Grados de libertad	3 (2 rotacionales y 1 lineal)	5 (totalmente articulado)
Trayectoria	Lineal cartesiana (interpolación entre puntos)	Secuencia discreta de posiciones predefinidas
Control del gripper	No tiene gripper (solo brazo)	Incluye apertura/cierre de pinza
Cinemática inversa	Ecuaciones simplificadas (2R + 1P)	Más compleja, involucra orientación y compensación en z
Movimiento	Continuo y suave	Por etapas (tareas pick and place)
Aplicación típica	Soldadura, ensamblaje, transporte horizontal	Manipulación precisa, laboratorio o aprendizaje

Los dos códigos utilizan ROS 2 para controlar robots manipuladores, pero tienen diferencias en cuanto a propósito y complejidad: el programa del DOFBOT emplea cinco grados de libertad y control de gripper para ejecutar una serie discreta de movimientos tipo pick-and-place, realizando cálculos completos de cinemática inversa con el fin de desplazar el efector a diferentes posiciones tridimensionales. Por otro lado, el código del SCARA produce un movimiento lineal continuo entre dos puntos en un plano usando interpolación cartesiana y simplificada cinemática inversa de tres grados de libertad, sin gripper. En conclusión, el DOFBOT se enfoca en manipular con exactitud en múltiples etapas, mientras que el SCARA se concentra en un desplazamiento suave y cíclico dentro de un plano de trabajo. Estos son dos enfoques diferentes del control por trayectorias en ROS 2.