Universidad Nacional Autónoma de México

Facultad de Ingeniería
Departamento de Ingeniería Mecatrónica
Robótica

Investigación

Comparación del código de control de robots seriales

Profesor:

Ing. Erik Peña Medina

Alumna:

Mujica Zeballos Carla

Grupo: 1 - Semestre: 2026-1

1. scara_tray_line_py.py

```
#Módulos para ROS 2
import rclpy
from rclpy.node import Node

#Importa los tipos de mensaje JointTrayectory y JointTrayectoryPoint para enviar
comandos de posición
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint

#Importa el mensaje Duration para especificar tiempos
from builtin_interfaces.msg import Duration

#Importa librerías de Python y funciones matemáticas paa calcular la cinemática inversa
import time
from math import cos, sin, acos, asin, atan2, sqrt

#Definimos el Nodo
class ScaraTrayLineNode(Node):
    def __init__(self):
        #Inicializamos el nodo
        super().__init__("scara_tray_line_node")
        #Colocamos el nombre del tópico donde se publicarán los comandos de trayectoria
        topic_name = "/scara_trajectory_controller/joint_trajectory"

        #Nombre de las articulaciones del robot
        self.joints_ = ['link_1_joint', 'link_2_joint', 'link_3_joint']

        #Parámetro de interpolación para el avance de la trayectoria
        self.lamda_ = 0

        #Tiempo total de ejecución de la trayectoria
        self.Tiempo_ejec_ = 10
```

```
    #Creación del Publicador en JointTrayectory
    self.scara_tray_pub_ = self.create_publisher(
        JointTrajectory, topic_name, 10)

    #Creación del temporizador que llama a la función trayectory_cbck cada
    1 segundo
    self.tray_timer_ = self.create_timer(
        1, self.trayectory_cbck)

    #Mensaje de información al iniciar el nodo
    self.get_logger().info(
        'Scara activo, trayectoria linea recta')

#Función Callback del temporizador que se ejecuta cada vez que el temporizador
dispara
def trayectory_cbck(self):
    trayectory_msg = JointTrajectory()

    #Asigna los nombres de las articulaciones al mensaje
    trayectory_msg.joint_names = self.joints_

    #Crea un nuevo punto de trayectoria
    point = JointTrajectoryPoint()

    #Lógica de control de la Trayectoria
    if self.lamda_ <= self.Tiempo_ejec_:
        #Posición Inicial
        x_1 = 0.1
        y_1 = 0.6
        theta_1 = 0

        #Posición final
        x_2 = 0.3
```

```
            y_2 = -0.6
            theta_2 = 1.57

            #Calcula las posiciones angulares de las articulaciones
            solucion = invk_sol(self.lamda_, x_1, y_1,theta_1, x_2, y_2,theta_2)

            #Asigna la solución al punto de trayectoria
            point.positions = solucion

            #Define el tiempo en que el robot debe alcanza esta posición
            point.time_from_start = Duration(sec=1)
            trayectory_msg.points.append(point)

            #Publica el comando de trayectoria al controlador del robot
            self.scara_tray_pub_.publish(trayectory_msg)

            #Muestra la postura actual publicada
            self.get_logger().info("Postura actual {}".format(solucion))
            time.sleep(2)

            #Incrementa el parámetro deinterpolación para el siguiente ciclo
            self.lamda_ += 1

        #Si la trayectoria ha terminado establece las posiciones e (0, 0, 0)
        elif self.lamda_ > 10:
            link_1_joint = 0
            link_2_joint = 0
            link_3_joint = 0

            #Devuelve la posición pero no la publica
            return [float(link_1_joint), float(link_2_joint), float(link_3_joint)]

#calcula las posiciones de las articulaciones para una posición cartesiaa utilizando interpo
def invk_sol(param,x_in, y_in, theta_in, x_fin, y_fin, theta_fin):
```

```
    #Parámetro de tiempo total de ejecución
    Tiempo_ejec_ = 10
    #Longitudes de los eslabones del robot SCARA
    L_1 = 0.5
    L_2 = 0.5
    L_3 = 0.3

    #Interpolación lineal
    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)

    #Proyección al centro de la articulación
    x_3 = x_P - L_3*cos(theta_P)
    y_3 = y_P - L_3*sin(theta_P)

    #Cálculo de angulos
    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(
    theta_1 = beta - psi
    theta_3 = theta_P -theta_1 -theta_2

    #Retorna los ángulos de las articulaciones como una lista de flotantes
    return [float(theta_1), float(theta_2), float(theta_3)]

#FUNCIÓN DE EJECUCIÓN PRINCIPAL
def main(args=None):
    rclpy.init(args=args)
    node = ScaraTrayLineNode()
    rclpy.spin(node)
    rclpy.shutdown()
if __name__ == "__main__":
    main()
```

2. dofbot_test_py.py

```
#Módulos de ROS
import rclpy
from rclpy.node import Node
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from builtin_interfaces.msg import Duration

#Importando funciones y liberías para cálculos de cinemática inversa
import time
from math import cos, sin, acos, asin, atan2, sqrt

#Definición del nodo de control
class DofbotControlNode(Node):
    def __init__(self):

        #Inicializa el nodo
        super().__init__("dofbot_tray_control_node")

        #Parámetro de estado para secuenciar los moviminetos
        self.lamda_ = 0

        #Definimos el tópico para enviar comandos de trayectoria
        topic_dofbot_ = "/dofbot_trajectory_controller/joint_trajectory"

        #Definimos el publicador de mensajes Joint Trayectory
        self.dofbot_publisher_ = self.create_publisher(JointTrajectory, topic_dofbot_, 10)

        #Nombramos las 5 articulaciones del Dofbot a controlar
        self.dofbot_joints_ = ['arm_joint_01', 'arm_joint_02', 'arm_joint_03', 'arm_joint_04

        #Crea un temporizador que llama a "timer_callback" cada 0.5 segundos
        self.timer_ = self.create_timer(0.5, self.timer_callback)
```

```
        #Mensaje informativo
        self.get_logger().info('Nodo de control del dofbot en funcionamiento')

    #Inicializa el callback del timer
    def timer_callback(self):
        dofbot_msg = JointTrajectory()
        dofbot_msg.joint_names = self.dofbot_joints_
        dofbot_point = JointTrajectoryPoint()

        #Fase 1, primer punto
        if self.lamda_ == 0:
            x_1 = 0.2
            y_1 = 0.0
            z_1 = 0.05

            #Orientación del efector final
            theta_p_1 = 3.1416*(3/4)
            theta_g_1 = 0.0

            #Llama a la función de cinemática
            solution_pos = dofbot_ink(x_1, y_1, z_1, theta_p_1, theta_g_1)

            #Asigna posición y define el tiempo de inicio para el controlador
            dofbot_point.positions = solution_pos
            dofbot_point.time_from_start = Duration(sec=2)
            dofbot_msg.points.append(dofbot_point)

            #Publica la posición
            self.dofbot_publisher_.publish(dofbot_msg)
            self.get_logger().info('poture {}'.format(solution_pos))

            #Espera 15 segundos
            time.sleep(15)
```

```
        self.lamda_ += 1

    #Fase 2: Segundo punto (lambda=1)
    elif self.lamda_ == 1:
        x_2 = 0.05
        y_2 = -0.15
        z_2 = 0.11
        theta_p_2 = 3.1416*(3/4)
        theta_g_2 = 0

        solution_pos = dofbot_ink(x_2, y_2, z_2, theta_p_2, theta_g_2)

        dofbot_point.positions = solution_pos
        dofbot_point.time_from_start = Duration(sec=2)
        dofbot_msg.points.append(dofbot_point)
        self.dofbot_publisher_.publish(dofbot_msg)
        self.get_logger().info('poture {}'.format(solution_pos))

        #Espera 15 segundos
        time.sleep(15)
        self.lamda_ += 1

    Fase 3; Tercer punto (lambda = 2)
    elif self.lamda_ == 2:

        # punto de de posicion del objeto
        x_3 = 0.15
        y_3 = 0.15
        z_3 = 0.10
        theta_p_3 = 3.1416*(3/4)
        theta_g_3 = 0
        solution_pos = dofbot_ink(x_3, y_3, z_3, theta_p_3, theta_g_3)

        dofbot_point.positions = solution_pos
```

```
            dofbot_point.time_from_start = Duration(sec=2)
            dofbot_msg.points.append(dofbot_point)

            self.dofbot_publisher_.publish(dofbot_msg)
            self.get_logger().info('poture {}'.format(solution_pos))

            #Espera 15 segundos para alcanzar la posición
            time.sleep(15)
            self.lamda_ += 1

        #Fase 4: Tercer punto (lambda = 3)
        elif self.lamda_ == 3:

            #Establece todas las articulaciones en posición de reposo
            solution_pos = [ float(0.0), float(0.0), float(0.0), float(0.0), float(0.0)]
            dofbot_point.positions = solution_pos
            dofbot_point.time_from_start = Duration(sec=2)
            dofbot_msg.points.append(dofbot_point)
            self.dofbot_publisher_.publish(dofbot_msg)
            self.get_logger().info('poture {}'.format(solution_pos))

            #Espera 10 segundos antes de que el callback se ejecute de nuevo
            time.sleep(10)

#Calcula los 5 ángulos de las articulaciones
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

    #Cálculo de Theta 1
    theta_1 = atan2(y_P, x_P)
```

```
    #proyección al plano de movimiento
    aux_x = sqrt(pow(x_P, 2) + pow(y_P, 2)) - L_3*sin(theta_1_P)
    aux_z = z_P - z_0_1 -L_3*cos(theta_1_P)

    #Magnitud del vector resultante
    norm_4_P = sqrt(pow(aux_z, 2)+pow(aux_x, 2))

    #Ángulo epsilon entre el vector norm_4_P y el eje Z
    epsilon = acos(aux_z/norm_4_P)

    #Ángulo alpha entre el triángulo inverso en los eslabones
    alpha = acos((pow(L_1, 2)+pow(norm_4_P, 2)-pow(L_2, 2))/(2*L_1*norm_4_P))

    #Cálculo de Theta 2
    theta_2 = epsilon - alpha

    #Cálculo de Theta 3
    theta_3 = 3.1416 - asin((sin(alpha)*sqrt(pow(aux_x, 2) + pow(aux_z, 2)))/(L_2))

    #Cálculo de Theta 4
    theta_4 = theta_1_P - theta_2 - theta_3

    #Cálculo de Theta 5
    theta_5 = theta_g
    return [ float(theta_1), float(-theta_2), float(-theta_3), float(-theta_4), float(theta_

#Función de ejecución principal
def main(args=None):
    rclpy.init(args=args)
    node = DofbotControlNode()
    rclpy.spin(node)
    rclpy.shutdown()
```

```
if __name__ == "__main__":
    main()
```