Comparacion del codigo de control de robots seriales

Rocio Fabiola Romero Bernal

1 de diciembre de 2025

1. Ensayo sobre los códigos

Estos dos códigos representan dos niveles distintos de complejidad en el control de manipuladores. El programa del robot SCARA se enfoca en un onjetivo, hacer que el efector final se mueva en una línea recta entre dos puntos del plano. Para lograrlo, se introduce un parámetro que actúa como tiempo normalizado y se interpolan las coordenadas cartesianas y la orientación. Después, se aplica la cinemática inversa planar para transformar ese punto intermedio en ángulos articulares. Esto hace muy evidente la conexión entre las ecuaciones de cinemática que se ven en clase y el movimiento real simulado en ROS2.

El código del Dofbot, en cambio, se acerca más a una tarea de manipulación típica, donde no basta con mover el brazo, sino que también hay que coordinar el gripper.Un gripper es, en pocas palabras, la "mano" del robot. Es el dispositivo que va al final del brazo robótico y que se encarga de agarrar, sujetar y soltar objetos durante una tarea (por ejemplo, tomar una pieza de una mesa y dejarla en otra posición). En este caso se define una secuencia de etapas controladas por una variable de estado; en cada etapa el robot abre o cierra la pinza y se desplaza a una postura específica en el espacio tridimensional. La función de cinemática inversa es más compleja, porque considera la altura y la orientación del efector, además de las longitudes de los eslabones. Comparar ambos programas ayuda a entender la diferencia entre un manipulador planar y uno espacial, y también muestra que la arquitectura del software cambia cuando se quiere ejecutar una trayectoria continua frente a una secuencia de acciones discretas. Como estudiante, esta comparación permite valorar la importancia de diseñar bien la interfaz entre la teoría (modelos cinemáticos) y la implementación (nodos, tópicos y temporizadores en ROS2).

2. Códigos

Listing 1: Control de robot SCARA con trayectoria lineal

```
1 #!/usr/bin/env python3
2 import rclpy
```

```
from rclpy.node import Node
from trajectory_msgs.msg import JointTrajectory,
    JointTrajectoryPoint
from builtin_interfaces.msg import Duration
import time
from math import cos, sin, acos, atan2, sqrt

class ScaraTrayLineNode(Node):
    def __init__(self):
        super().__init__("scara_tray_line_node")
        # Nombre del topic para publicar la trayectoria del robot
            SCARA
        topic_name = "/scara_trajectory_controller/joint_trajectory
            "
        # Nombres de las articulaciones a controlar
        self.joints_ = ['link_1_joint', 'link_2_joint', '
            link_3_joint']
        # Variable para controlar el tiempo/avance de la
            trayectoria
        self.lamda_ = 0
        # Tiempo total de ejecuci n de la trayectoria en segundos
        self.Tiempo_ejec_ = 10
        # Creador del publicador para publicar mensajes de tipo
            JointTrajectory
        self.scara_tray_pub_ = self.create_publisher(
            JointTrajectory, topic_name, 10)
        # Timer que ejecuta la funci n trayectory_cbck cada 1
            segundo
        self.tray_timer_ = self.create_timer(1, self.
            trayectory_cbck)
        # Log para indicar que el nodo est  activo y el tipo de
            trayectoria
        self.get_logger().info('Scara activo, trayectoria linea
            recta')

    def trayectory_cbck(self):
        # Crea el mensaje de trayectoria para enviar las posiciones
             articulares
        trayectory_msg = JointTrajectory()
        trayectory_msg.joint_names = self.joints_
        point = JointTrajectoryPoint()
```

```
32
33          if self.lamda_ <= self.Tiempo_ejec_:
34              # Definici n del punto inicial (x_1,y_1,theta_1) y
                    final (x_2,y_2,theta_2) del movimiento lineal
35              x_1 = 0.1
36              y_1 = 0.6
37              theta_1 = 0
38              x_2 = 0.3
39              y_2 = -0.6
40              theta_2 = 1.57
41
42              # Invoca la cinem tica inversa interpolando entre
                    punto inicial y final seg n lamda_
43              solucion = invk_sol(self.lamda_, x_1, y_1, theta_1, x_2
                    , y_2, theta_2)
44              point.positions = solucion
45              # Define el tiempo desde el inicio para esta posici n
46              point.time_from_start = Duration(sec=1)
47              trayectory_msg.points.append(point)
48              # Publica la trayectoria calculada
49              self.scara_tray_pub_.publish(trayectory_msg)
50              self.get_logger().info("Postura actual {}".format(
                    solucion))
51              # Pausa para permitir movimiento
52              time.sleep(2)
53              self.lamda_ += 1
54
55          elif self.lamda_ > 10:
56              # Cuando termina la trayectoria, se devuelve la
                    posici n de reposo en 0
57              link_1_joint = 0
58              link_2_joint = 0
59              link_3_joint = 0
60              return [float(link_1_joint), float(link_2_joint), float
                    (link_3_joint)]
61
62  def invk_sol(param, x_in, y_in, theta_in, x_fin, y_fin, theta_fin):
63      # Par metros y longitudes de los brazos
64      Tiempo_ejec_ = 10
65      L_1 = 0.5
66      L_2 = 0.5
```

```
67      L_3 = 0.3
68
69      # Interpolaci n lineal del punto final (posici n y
            orientaci n)
70      x_P = x_in + (param/Tiempo_ejec_)*(x_fin - x_in)
71      y_P = y_in + (param/Tiempo_ejec_)*(y_fin - y_in)
72      theta_P = theta_in + (param/Tiempo_ejec_)*(theta_fin - theta_in
            )
73
74      # Cinem tica inversa para calcular posiciones articulares
75      x_3 = x_P - L_3*cos(theta_P)
76      y_3 = y_P - L_3*sin(theta_P)
77      # Angulo de la segunda articulacion basado en la ley del coseno
78      theta_2 = acos((pow(x_3, 2)+pow(y_3,2)-pow(L_1, 2)-pow(L_2, 2))
            /(2*L_1*L_2))
79      beta = atan2(y_3, x_3)
80      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))))
81      #  ngulo  de la primera articulaci n
82      theta_1 = beta - psi
83      #  ngulo  de la tercer articulaci n para orientaci n final
            deseada
84      theta_3 = theta_P - theta_1 - theta_2
85
86      return [float(theta_1), float(theta_2), float(theta_3)]
87
88  def main(args=None):
89      rclpy.init(args=args)
90      node = ScaraTrayLineNode()
91      rclpy.spin(node)
92      rclpy.shutdown()
93
94  if __name__ == "__main__":
95      main()
```

3. Código del robot Dofbot

Listing 2: Control secuencial del Dofbot con gripper

```
1  #!/usr/bin/env python3
2  import rclpy
```

```
from rclpy.node import Node
from trajectory_msgs.msg import JointTrajectory,
    JointTrajectoryPoint
from builtin_interfaces.msg import Duration
import time
from math import cos, sin, acos, atan2, sqrt

class DofbotControlNode(Node):
    def __init__(self):
        super().__init__("dofbot_tray_control_node")
        # Variables y t picos para publicar comandos de
            trayectoria y para control del gripper
        self.lamda_ = 0
        topic_dofbot_ = "/dofbot_trajectory_controller/
            joint_trajectory"
        topic_gripper_ = "/dofbot_gripper_controller/
            joint_trajectory"
        self.dofbot_publisher_ = self.create_publisher(
            JointTrajectory, topic_dofbot_, 10)
        # Lista de nombres de las articulaciones del brazo
        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)
        # Lista de articulaciones del gripper para abrir/cerrar
        self.gripper_joints_ = ['grip_joint', 'rfinger_joint_01', '
            rfinger_joint_02', 'lfinger_grip_joint_01', '
            lfinger_grip_joint_02', 'lfinger_grip_joint_03']
        # Timer que llama peri dicamente a timer_callback cada 0.5
             segundos
        self.timer_ = self.create_timer(0.5, self.timer_callback)
        self.get_logger().info('Nodo de control del dofbot en
            funcionamiento')

    def timer_callback(self):
        # Mensajes para trayectoria del brazo y gripper
        dofbot_msg = JointTrajectory()
        dofbot_msg.joint_names = self.dofbot_joints_
        dofbot_point = JointTrajectoryPoint()
        gripper_msg = JointTrajectory()
        gripper_msg.joint_names = self.gripper_joints_
```

```
33          gripper_point = JointTrajectoryPoint()
34
35          # Control de secuencia con variable lamda_ para seguir
               pasos definidos
36          if self.lamda_ == 0:
37              # Abrir gripper
38              gstate = 1.57
39              gripper_st = gripper_state(gstate)
40              gripper_point.positions = gripper_st
41              gripper_point._time_from_start = Duration(sec=1)
42              gripper_msg.points.append(gripper_point)
43              self.gripper_publisher_.publish(gripper_msg)
44              self.get_logger().info('Gripper open')
45              self.get_logger().info('poture {}'.format(gripper_st))
46              time.sleep(5)
47              self.lamda_ += 1
48
49          elif self.lamda_ == 1:
50              # Cerrar gripper
51              gstate_2 = 0
52              gripper_st = gripper_state(gstate_2)
53              gripper_point.positions = gripper_st
54              gripper_point._time_from_start = Duration(sec=1)
55              gripper_msg.points.append(gripper_point)
56              self.gripper_publisher_.publish(gripper_msg)
57              self.get_logger().info('Gripper close')
58              self.get_logger().info('poture {}'.format(gripper_st))
59              time.sleep(5)
60              self.lamda_ += 1
61
62          elif self.lamda_ == 2:
63              # Abrir gripper (otra vez)
64              gstate = 1.57
65              gripper_st = gripper_state(gstate)
66              gripper_point.positions = gripper_st
67              gripper_point._time_from_start = Duration(sec=1)
68              gripper_msg.points.append(gripper_point)
69              self.gripper_publisher_.publish(gripper_msg)
70              self.get_logger().info('Gripper open')
71              self.get_logger().info('poture {}'.format(gripper_st))
72              time.sleep(5)
```

```
73              self.lamda_ += 1
74
75          elif self.lamda_ == 3:
76              # Primera postura, calculada con cinem tica inversa
                    para posicionar el brazo
77              x_1 = 0.2
78              y_1 = 0.0
79              z_1 = 0.05
80              theta_p_1 = 3.1416*(3/4)
81              theta_g_1 = 0.0
82              solution_pos = dofbot_ink(x_1, y_1, z_1, theta_p_1,
                    theta_g_1)
83              dofbot_point.positions = solution_pos
84              dofbot_point.time_from_start = Duration(sec=2)
85              dofbot_msg.points.append(dofbot_point)
86              self.dofbot_publisher_.publish(dofbot_msg)
87              self.get_logger().info('poture {}'.format(solution_pos)
                    )
88              time.sleep(15)
89              self.lamda_ += 1
90
91          elif self.lamda_ == 4:
92              # Cerrar gripper para sujetar objeto
93              gstate_2 = 0
94              gripper_st = gripper_state(gstate_2)
95              gripper_point.positions = gripper_st
96              gripper_point._time_from_start = Duration(sec=2)
97              gripper_msg.points.append(gripper_point)
98              self.gripper_publisher_.publish(gripper_msg)
99              self.get_logger().info('Gripper close')
100             self.get_logger().info('poture {}'.format(gripper_st))
101             time.sleep(10)
102             self.lamda_ += 1
103
104         # ... Siguientes pasos para varias posturas y movimientos
                en secuencia ...
105
106         elif self.lamda_ == 9:
107             # Regresar a postura inicial (todo en ceros)
108             solution_pos = [float(0.0)] * 5
109             dofbot_point.positions = solution_pos
```

```
110             dofbot_point.time_from_start = Duration(sec=2)
111             dofbot_msg.points.append(dofbot_point)
112             self.dofbot_publisher_.publish(dofbot_msg)
113             self.get_logger().info('poture {}'.format(solution_pos)
                    )
114             time.sleep(10)
115
116 def dofbot_ink(x_P, y_P, z_P, theta_1_P, theta_g):
117     # Par metros de longitud y offsets del brazo
118     z_0_1 = 0.105
119     L_1 = 0.084
120     L_2 = 0.084
121     L_3 = 0.115
122     # C lculos de  ngulos  para cinem tica inversa en 3D
123     theta_1 = atan2(y_P, x_P)
124     aux_x = sqrt(pow(x_P, 2) + pow(y_P, 2)) - L_3*sin(theta_1_P)
125     aux_z = z_P - z_0_1 - L_3*cos(theta_1_P)
126     norm_4_P = sqrt(pow(aux_z, 2) + pow(aux_x, 2))
127     epsilon = acos(aux_z / norm_4_P)
128     alpha = acos((pow(L_1, 2) + pow(norm_4_P, 2) - pow(L_2, 2)) /
            (2 * L_1 * norm_4_P))
129     theta_2 = epsilon - alpha
130     theta_3 = 3.1416 - asin((sin(alpha) * sqrt(pow(aux_x, 2) + pow(
            aux_z, 2))) / L_2)
131     theta_4 = theta_1_P - theta_2 - theta_3
132     theta_5 = theta_g
133     return [float(theta_1), float(-theta_2), float(-theta_3), float
            (-theta_4), float(theta_5)]
134
135 def gripper_state(theta):
136     # Genera la posici n para los distintos dedos del gripper con
            signo alternado
137     return [float(-theta), float(theta), float(-theta), float(theta
            ), float(-theta), float(theta)]
138
139 def main(args=None):
140     rclpy.init(args=args)
141     node = DofbotControlNode()
142     rclpy.spin(node)
143     rclpy.shutdown()
144
```

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