

7. Ejercicios prácticos



7.1. Explora las articulaciones del robot	2
● Movimientos:	2
7.2. Práctica el modo de aprendizaje del robot	3
● Modo aprendizaje:	3
7.3. Programación básica de movimientos con Blockly:	3
● Programación con bloques:	3
7.4. Programación del Niryo One con Python	4
● Programación con python:	4
7.5. Resolución del problema cinemático directo para el Niryo One	4
● Problema cinemático directo:	4
● Código de cinemática directa	4
Links:	7

7.1. Explora las articulaciones del robot

- Movimientos:

En primer lugar calibraremos los ejes del robot para evitar posibles problemas a la hora de mover demasiado los ejes si estos están descalibrados. A continuación procedemos a mover cada uno de los ejes por separado para comprobar el correcto funcionamiento de los mismos.

Finalizamos con un movimiento total de todas las articulaciones en conjunto.

Continuamos moviendo las articulaciones a las posiciones que nos indica en la hoja de práctica.

$$\theta = [1.6, 0.016, -0.1, -0.026, -0.005, 0]$$

Una vez llegado a la posición asignada en radianes, podemos observar la posición en coordenadas haciendo clic en set to current y guardar esta nueva posición.

A continuación movemos el robot a una posición aleatoria e insertamos los valores pedidos por el ejercicio:

$$x = 0.08; y = -0.04; z = 0.55$$

Esto moverá nuestra herramienta directamente a las coordenadas especificadas.

Realizaremos los mismos pasos anteriores para asegurarnos de que el robot está en las coordenadas asignadas y guardaremos la posición nuevamente.

Ahora añadiremos la posición cero a nuestro robot, para esto movemos las articulaciones con MOVE JOINTS a su punto 0 y salvaremos la posición de la misma forma pero esta vez con el nombre “Posición 0”

Para finalizar en la pestaña JOINTS seleccionamos la posición guardada anteriormente y seleccionaremos MOVE JOINTS .

7.2. Práctica el modo de aprendizaje del robot

- Modo aprendizaje:

Activamos el LEARNING MODE del robot y movemos las articulaciones a una posición cualquiera. Observamos que al hacer esto el robot pierde la capacidad de soportar su propio peso pero sigue reconociendo la posición en la que se encuentra en todo momento, enviando los datos a través de la aplicación.

Realizado estos movimientos pinchamos en UPDATE VALUES y desactivamos el LEARNING MODE.

A continuación realizaremos el movimiento a la posición guardada anteriormente haciendo clic en MOVE AXES

Realizaremos el mismo proceso que en el paso anterior pero a diferencia de su posición, esta vez debe ser totalmente vertical. También realizamos el posicionamiento de la misma forma que antes MOVE AXES

7.3. Programación básica de movimientos con Blockly:

- Programación con bloques:

En este apartado no nos centraremos mucho ya que lo que nos interesa es la programación directa en Python que veremos más tarde.



Para este punto solamente realizaremos un pequeño código donde se mueva a dos posiciones de forma repetida.

Manuel Fernández Uceira
Denís Gómez Solla

7.4. Programación del Niryo One con Python

- Programación con python:

En este apartado copiamos el código ya predefinido para probar como se carga el programa en el robot niryo

7.5. Resolución del problema cinemático directo para el Niryo One

- Problema cinemático directo:

Este apartado es el más importante de la práctica, ya que ponemos en funcionamiento los conocimientos adquiridos durante gran parte del curso para poder mover el robot a las posiciones que deseamos.

Utilizadno como guía el código proporcionado por el profesor creamos nuestro propio codigo para resolver La matriz homogenea, los ejes de las articulaciones y las matrices exponenciales para determinar la posición final de la herramienta.

- Código de cinemática directa

```
#!/usr/bin/env python3
"""
Programa de cinemática directa
Uso: python3 CinematicaDirectaNiryo.py -j "90 90 90 90 90 90" -c config.yaml
"""

# Importamos las librerias necesarias para el programa
import numpy as np
import argparse
import yaml
import sys

# Funciones para los vectores
# Funcion para calcular la matriz antisimetrica a partir de un vector
def matriz_antisimetrica_3(omega):
    return np.array([
        [0, -omega[2], omega[1]],
        [omega[0], 0, -omega[2]],
        [-omega[1], omega[2], 0]])
Manuel Fernández Uceira
Denís Gómez Solla
```

```

[omega[2], 0, -omega[0]],
[-omega[1], omega[0], 0]
])

# Funcion para devolver un vector de 3 componentes a partir de una matriz antisimetrica
def vector_de_antiS(matriz):
    return np.array([matriz[2, 1], matriz[0, 2], matriz[1, 0]])

# Funciones para las matrices exponentiales
# Funcion para calcular la matriz exponencial (transformacion homogenea) a partir de un
vector de giro
def matriz_exponencial_3(vec):
    return np.r_[np.c_[matriz_antisimetrica_3(vec[0:3]), vec[3:6]], np.zeros((1, 4))]

# convierte matriz se3 en una matriz de transformacion homogenea a traves de la
exponencial
def matriz_exponencial_6(matriz):
    matriz = np.array(matriz)
    vec = matriz[0:3, 3]
    omgmattheta = matriz[0:3, 0:3]
    omgtheta = vector_de_antiS(omgmattheta)

    if np.linalg.norm(omgtheta) < 1.e-6:
        return np.r_[np.c_[np.eye(3), vec], [[0, 0, 0, 1]]]
    else:
        theta = np.linalg.norm(omgtheta)
        omgmat = omgmattheta / theta
        G_theta = (
            np.eye(3) * theta +
            (1 - np.cos(theta)) * omgmat +
            (theta - np.sin(theta)) * np.dot(omgmat, omgmat)
        )
        R = (
            np.eye(3) +
            np.sin(theta) * omgmat +
            (1 - np.cos(theta)) * np.dot(omgmat, omgmat)
        )
        return np.r_[np.c_[R, np.dot(G_theta, vec) / theta], [[0, 0, 0, 1]]]

# devuelve los ángulos de Euler a partir de una matriz rotacion que es una matriz 3x3
def formula_Euler(R):
    sy = np.sqrt(R[0, 0]**2 + R[1, 0]**2)
    singular = sy < 1.e-6
    if not singular:
        x = np.arctan2(R[2, 1], R[2, 2])
        y = np.arctan2(-R[2, 0], sy)

```

Manuel Fernández Uceira
 Denís Gómez Solla

```

z = np.arctan2(R[1, 0], R[0, 0])
else:
    x = np.arctan2(-R[1, 2], R[1, 1])
    y = np.arctan2(-R[2, 0], sy)
    z = 0
return np.array([x, y, z])

# Funcion para cargar la configuracion del robot desde un archivo yaml
def cargar_configuracion(ruta_yaml):
    try:
        with open(ruta_yaml, 'r') as file:
            data = yaml.safe_load(file)
        return data['robot']
    except Exception as e:
        print(f"Error al leer el archivo YAML: {e}")
        sys.exit(1)

# Funcion principal del programa
def main():
    parser = argparse.ArgumentParser(description = "Cinemática directa de robot desde configuración YAML")
    parser.add_argument('-j', '--joints', type = str, default = "0 0 0 0 0 0", help = 'Ángulos de las articulaciones en grados')
    parser.add_argument('-c', '--config', type = str, default = 'configuracion.yaml', help = 'Archivo de configuración YAML')
    args = parser.parse_args()

    robot = cargar_configuracion(args.config)
    enlaces = robot['enlaces']

    # Cargar ejes y posiciones desde YAML
    ws = [np.array(link['eje_articulacion']) for link in enlaces]
    qs = [np.array(link['coordenadas_articulacion']) for link in enlaces]

    # Validación de ángulos
    t_grados = [float(a) for a in args.joints.split()]
    if len(t_grados) != len(ws):
        print(f"Error: se esperaban {len(ws)} ángulos, pero se proporcionaron {len(t_grados)}.")
        sys.exit(1)
    t = np.deg2rad(t_grados)
    print("Coordenadas en radianes.", np.round(t, 5))

    # Cálculo de vectores de tornillo (Si = [w, v])
    vs = [np.cross(-w, q) for w, q in zip(ws, qs)]
    Si = [np.r_[w, v] for w, v in zip(ws, vs)]
  
```

```
# Supongamos una matriz M identidad si no está en el YAML
M = np.eye(4)

# Transformación homogénea final
T = np.eye(4)
for i in range(len(Si)):
    T = np.dot(T, matriz_exponencial_6(matriz_exponencial_3(Si[i] * t[i])))
T = np.dot(T, M)

# Resultados
print("\nMatriz de transformación homogénea:", np.round(T, 3))
print("\nCoordenadas (x, y, z) del TCP:", np.round(T[0:3, 3], 3))
R = np.round(T[0:3, 0:3], 3)
euler_rad = formula_Euler(R)
print("\nÁngulos de Euler en radianes:", np.round(euler_rad, 5))
print("\nÁngulos de Euler en grados:", np.round(np.rad2deg(euler_rad), 3))

# Llamada de la función principal
if __name__ == "__main__":
    main()
```

Con este código podemos observar como realiza los movimientos esperados dentro de un margen de error, tanto por las holguras de la cinemática del robot como por las aproximaciones a 3 decimales de los cálculos de el código de cinemática directa.

Este mismo también tiene una repetitividad relativamente precisa, no llegando a variar mas de 0.8mm.

Links:

Carpeta con videos de práctica.

https://drive.google.com/drive/folders/1Z2OA7S_xajWofHBsCEvBRY23d2-FtuOt?usp=drive_link