Descripción de la solución

Modelo cinemático directo del robot

Se creó el modelo cinemático del robot usando la toolbox de Peter Corke. Inicialmente, se midieron los eslabones utilizando un calibrador pie de rey. Las imágenes de las mediciones se pueden observar en las figuras 1 y 2.

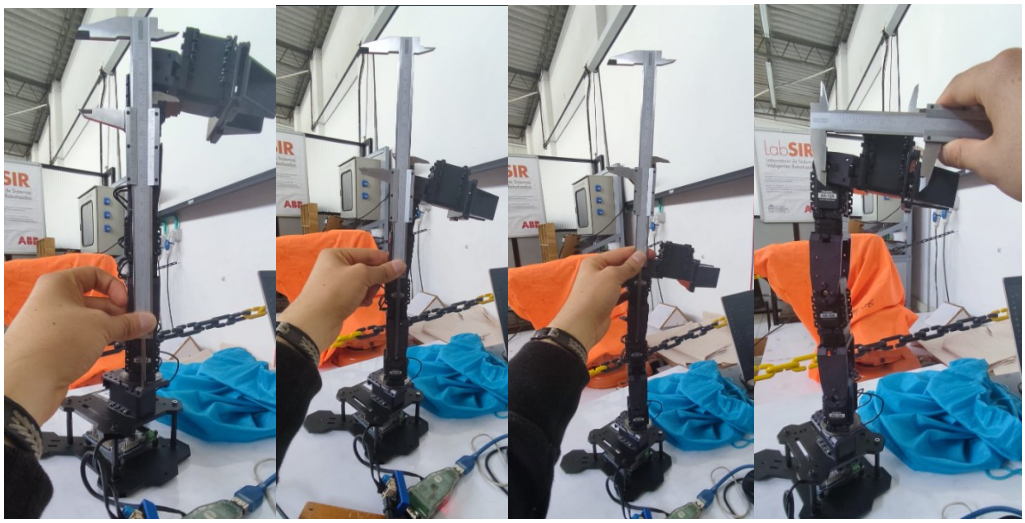

Figura 1. Mediciones tomadas para el robot.

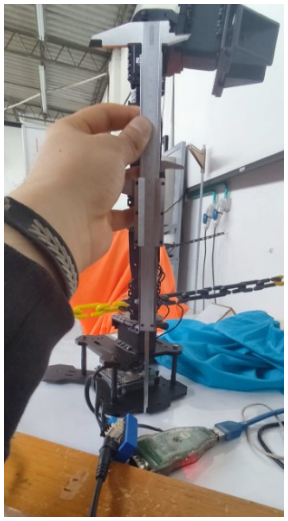

Figura 2. Medición de la base.

Los resultados fueron L1=45mm, L2= L3=105mm, L4 = 75mm. Cabe aclarar que L4 va hasta el centro la “base” del gripper, la distancia desde esta base al centro del gripper es de 20mm. Ahora se realiza el esquema del robot para establecer los sistemas coordenados y así los parámetros DH. Este diagrama se observa en la figura 3.

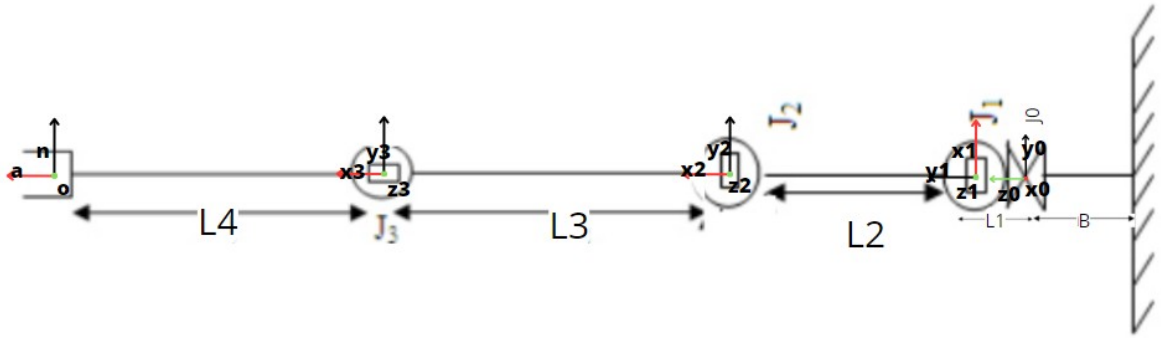

Figura 3. Diagrama del robot. Por comodidad en el documento se poner horizontal.

A estos sistemas coordenados le corresponden los parámetros mostrados en la figura 4.

```
Phantom =

Phantom:: 4 axis, RRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|         45|          0|     1.5708|          0|
|  2|         q2|          0|        105|          0|     1.5708|
|  3|         q3|          0|        105|          0|          0|
|  4|         q4|          0|         75|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
base:    t = (90, 0, 0), RPY/xyz = (0, 0, 0) deg
tool:    t = (20, 0, 0), RPY/xyz = (0, 90, 90) deg
```

Figura 4. Parámetros DH.

Luego, se crearon las matrices de transformación homogénea para cada eslabón, se realizó el modelo cinemático directo desde la base hasta el centro del gripper multiplicando las matrices de cada eslabón. Dada la extensión de estas matrices, se dejan en el código de Matlab anexo a este repositorio en https://github.com/Ggio0/Ros_lab4/tree/main/Entregables/Matlab .

Este modelo es simulado usando la toolbox de Peter Corke y se simulan las 4 poses dadas por la guía para compararlas con la pose del robot en la implementación física. Los resultados se muestran en las figuras 5, 6, 7, 8, 9.

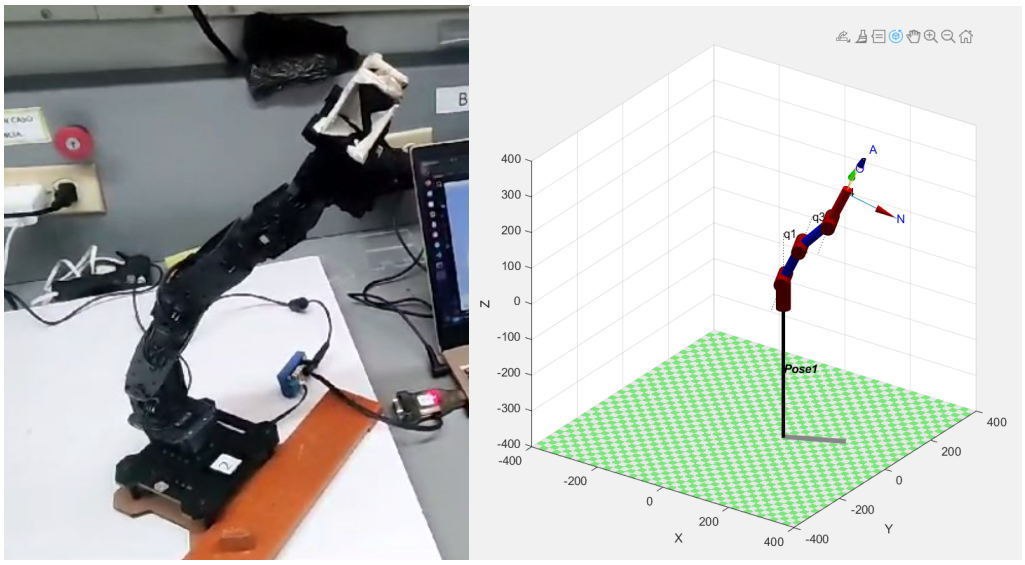

Figura 5. Comparación de la simulación con la implementación de la pose 1.

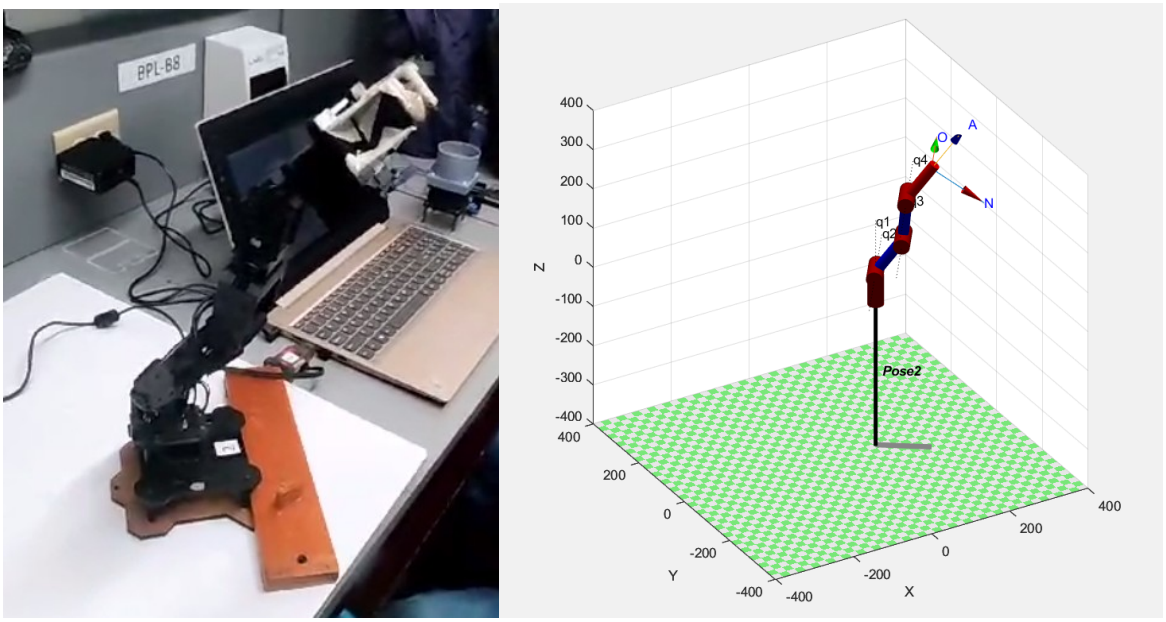

Figura 6. Comparación de la simulación con la implementación de la pose 2.

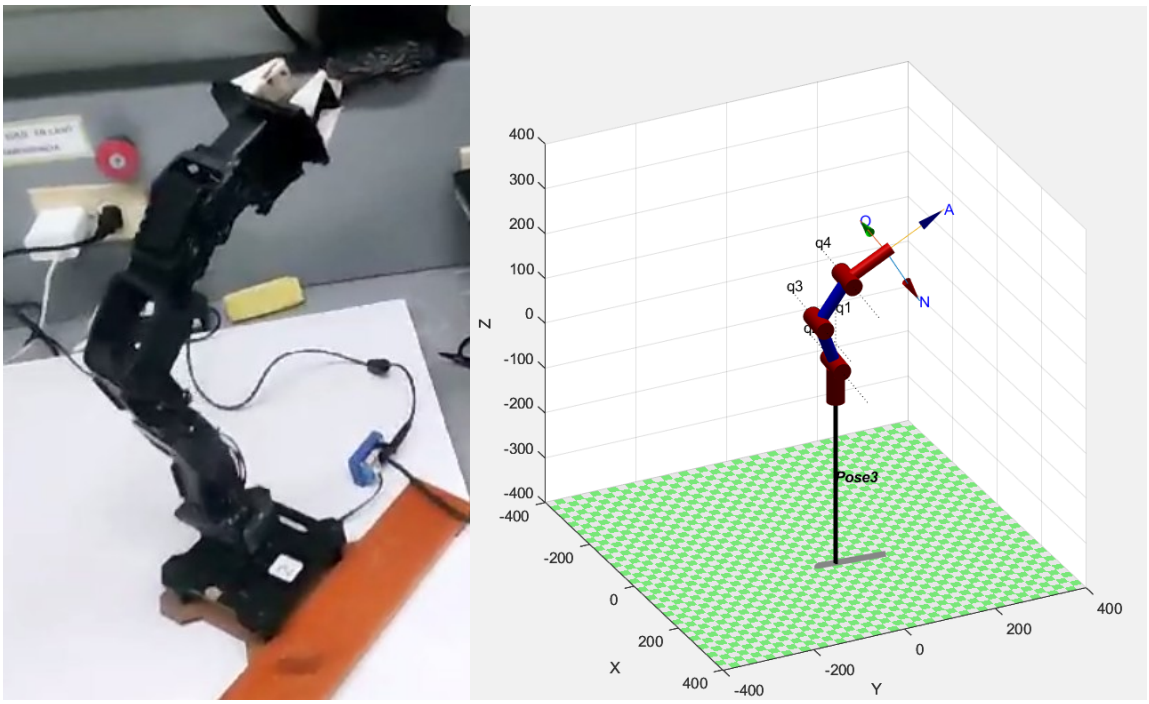

Figura 7. Comparación de la simulación con la implementación de la pose 3.

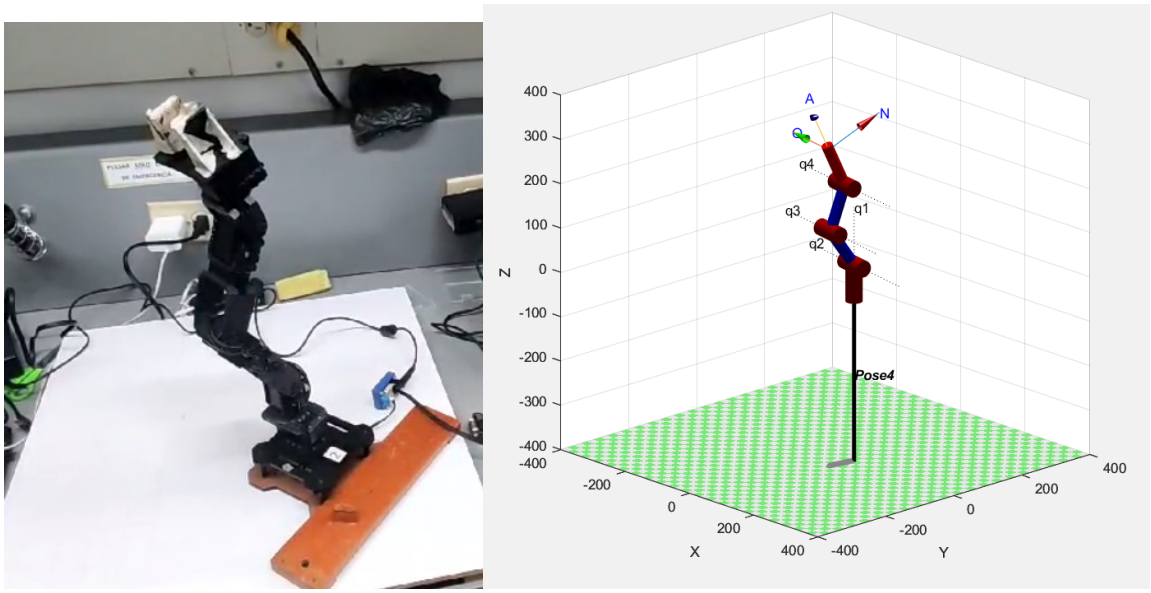

Figura 8. Comparación de la simulación con la implementación de la pose 4.

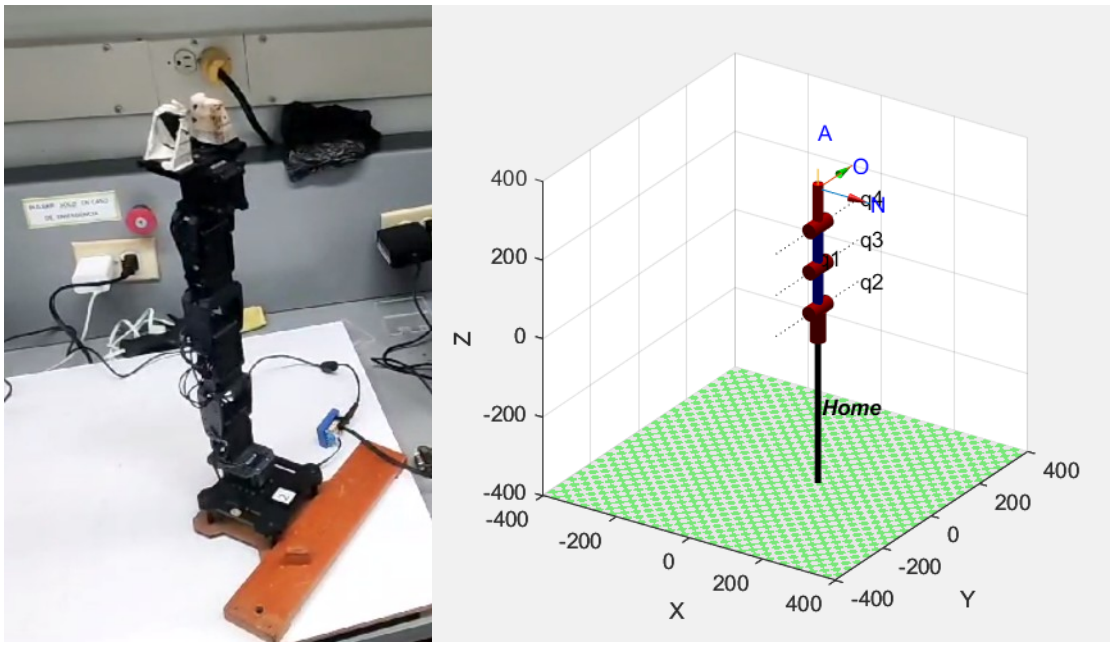

Figura 9. Comparación de la simulación con la implementación de la pose home.

Conexión al phantom

Para la conexión al phantom, se instalaron las librerías de dynamixel para su utilización en Ros, primero se realizó la verificación por la aplicación de Dynamixwl Wizard de las posiciones del robot, los limites articulares y la limitación de torque de cada motor.
Posteriormente se verifica la conexión del puerto y se ejecuta atreves de ROS el archivo pxcontroler.launch para realizar la conexión y nombrar las articulaciones por medio de archivo joints.yaml como se ve en la siguiente Figura 10.

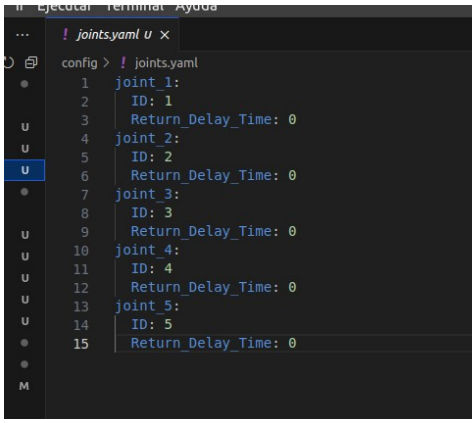

Figura 10. Código Joints.yml.

Código de Python y catkin

Se configura el archivo CmakeList.txt para agregar el script de Python donde esta nuestro código, además de archivos adicionales como las imágenes usadas para la interfaz gráfica, ver figura 11.

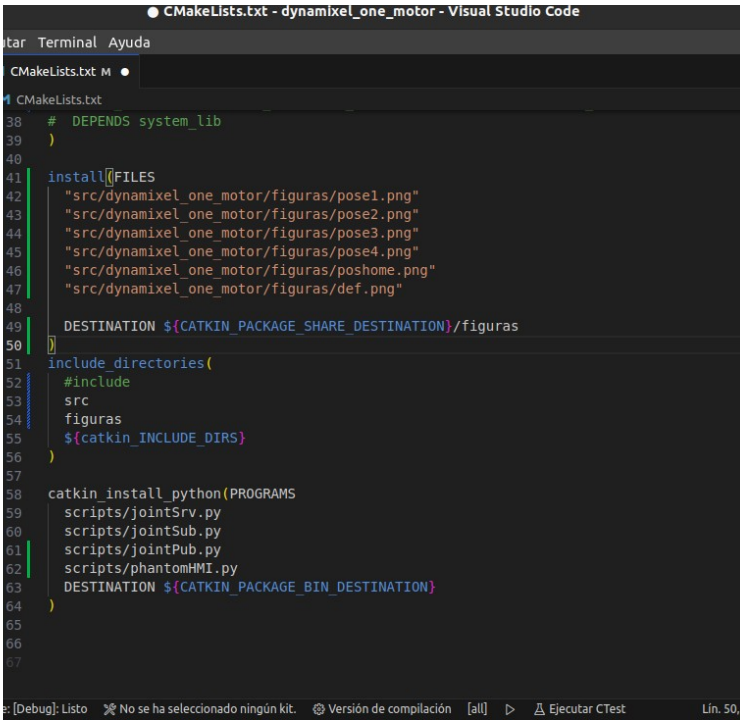

Figura 11 código CmakeList.txt.

Una vez creadas las rutas procedemos con la realización del código, donde utilizamos 3 servicios, publicar, en la función join_publisher, donde tiene como parámetro un vector con la configuración de cada articulación en radianes; suscribirse, en la función listener, el cual se encarga de recibir las posiciones actuales del phantom, y por último el servicio de dynamixel, el cual no permite ajustar diversos parámetros de acuerdo al número id y su valor, en nuestra aplicación se usó para limitar el torque como se muestra en la figura 12.

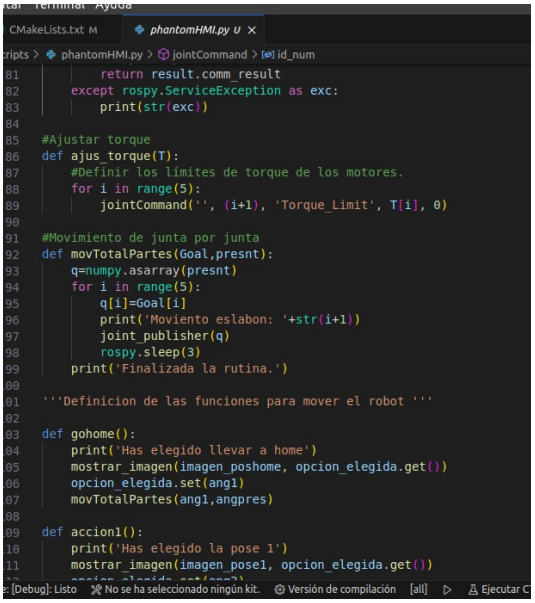

Figura 12 código en python.

Con estos servicios, se crea la función movTotalPartes, la cual recibe la posición objetivo y la posición presente y se encarga de mover articulación por articulación hasta llegar a la pose requerida.

Interfaz grafica

La interfaz gráfica se llevó a cabo con la librería tkinter, donde se agregaron los nombres de los integrantes del grupo, dos imágenes, en la izquierda la pose pasada del robot, y en la derecha la pose actual, posteriormente se imprimen los valores de la pose anterior y luego los de la pose actual, por último, se configuraron 5 botones, los cuales se encargarían de llevar el robot articulación por articulación hasta la pose objetivo, dicha interfaz se muestra en la siguiente figura 13.

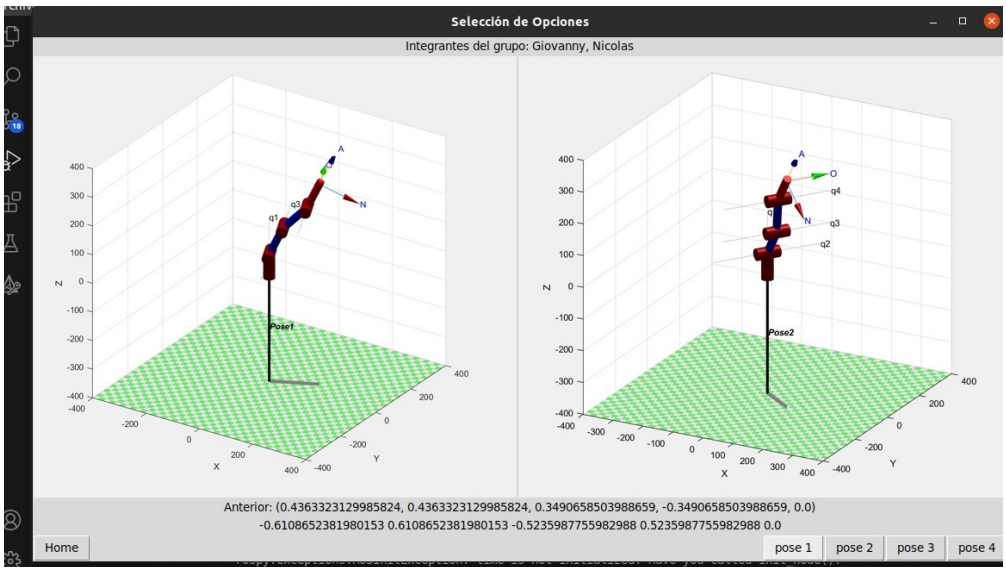

Figura 13 código Interfaz gráfica.