Universidad Nacional de Colombia - Sede Bogotá
Facultad de Ingeniería
Departamento de Ingeniería Mecánica y Mecatrónica
Asignatura: Robótica
Estudiantes: Jonathan Andres Jimenez Trujillo
Yeira Liseth Rodriguez Rodriguez
Daniel Felipe Valbuena Reyes
Daniel Mauricio Rivero Lozada

Resumen Ejecutivo

Objetivos Alcanzados:

Para el desarrollo del proyecto se lograron los siguientes objetivos:

- Se logró emplear el desarrollo de la cinemática directa a partir de la obtención de los parámetros DH que se obtuvieron mediante las mediciones sobre la longitud de los eslabones y las restricciones geométricas y articulares que permitieron construir un modelo gráfico para posteriormente realizar el análisis cinemático.
- Se logró crear un nodo para el Jacobiano y su inverso para el cálculo que realiza la transformación de los movimientos desde el joystick como un Δx a los movimientos de articulaciones del pincher en un resultado de Δq.
- Cómo uno de los objetivos iniciales, fue posible llevar a cabo el proceso de establecer una conexión con el joystick y los demás botones del mando, de manera que desde ROS2, se establece esta combinación que permite tomar los datos de estos elementos y publicarlos dentro de un tópico.
- Por otro lado, la creación de paquetes dentro de un workspace desarrollado fue llevado de manera exitosa en cuanto a su construcción e instalación a partir de los archivos desarrollados en Python que definen el funcionamiento de cada paquete. Por medio de estos paquetes, es posible realizar un desarrollo segmentado de la solución, de manera que se logra establecer de manera clara y óptima la creación de los nodos correspondientes a la solución planteada.
- Para temas de simulación, se logró emplear en MATLAB el desarrollo de una rutina pick and place con ayuda de la librería de Robotic Tools de Peter Corke. Para ello, a partir de puntos en coordenadas homogéneas, se obtiene la cinemática inversa del robot, es decir, su configuración articular para cada uno de estos puntos con ayuda de la función ikine(). Posteriormente, teniendo los valores de la configuración del robot en cada uno de los puntos, se realizan trayectorias de interpolación que permiten representar y obtener los valores de la configuración del robot a lo largo de cada punto que se recorre en la trayectoria.

Pendientes:

En primer lugar, aún no se ha implementado la interfaz gráfica en la zona local. Es necesario desarrollar una interfaz que incluya botones para iniciar y detener el proceso, indicadores del estado del sistema (operación/parada y modo manual/automático), un selector de trayectoria para el modo automático y, opcionalmente, una vista remota mediante una cámara virtual para la supervisión en tiempo real.

También falta la integración completa de la arquitectura distribuida. Aunque hemos trabajado en la conexión del joystick y la comunicación entre nodos dentro de ROS2, aún debemos verificar y asegurar la conexión en red (LAN) entre la computadora local (operador) y la remota (donde se encuentra el robot físico o la simulación en CoppeliaSim/Rviz).

Respecto a la simulación, si bien se ha desarrollado una rutina pick and place en MATLAB, se debe complementar la solución mediante la implementación en Rviz o CoppeliaSim. Esto permitirá una representación más fiel del entorno y una integración directa con la simulación del robot, incluyendo la dinámica y la interacción con objetos.

Dificultades:

1. Problemas en la creación y configuración de paquetes en ROS2

Durante la primera fase del desarrollo, se encontraron diversas dificultades en la creación y gestión de paquetes y nodos en ROS2. Algunos de los problemas más recurrentes fueron:

- Fallas en la activación del entorno en ciertos computadores.
- Paquetes que, a pesar de ser creados, no aparecían en la lista de ‘ros2 pkg list’.
- Inconsistencias en la nomenclatura de los paquetes, como el uso de mayúsculas, que generaban errores de compatibilidad.
- Problemas generales en la implementación de ROS2, lo que requirió una depuración constante para garantizar un funcionamiento adecuado del entorno de trabajo.

2. Dificultades en el cálculo de la matriz Jacobiana

Otro desafío importante surgió en la validación del cálculo de la matriz Jacobiana del robot. Se encontraron diferencias entre los resultados obtenidos manualmente y aquellos generados mediante la función ‘jacob0’ del Toolbox de MATLAB. La confusión inicial se debió a la omisión de un vector excedente ‘o5’ en la matriz que relaciona la base con el TCP. Esto nos generó dudas, porque pensamos que al añadirse este nuevo vector, también se necesitaba añadir el vector z5, cambiando la matriz jacobiana a un tamaño de 6x5, cuando los resultados esperados y obtenidos por el toolbox era de 6x4. Una vez aclarada la duda, de que no se generaba un vector z5 y la matriz seguía siendo 6x4, al realizar los cálculos se pudo

evidenciar que finalmente los resultados obtenidos de manera manual y por la función ‘jacob0’ fueron iguales.

3. Problemas en la expresión simbólica de la matriz Jacobiana

Al representar la matriz Jacobiana en símbolos matemáticos dentro de MATLAB, se encontraron dificultades en la generación de ecuaciones de las posiciones de la matriz. MATLAB devolvía fracciones extremadamente grandes y expresiones numéricas con un alto número de decimales, lo que complicaba su implementación en el nodo de Python. A pesar de utilizar las funciones simplify y vpa, las expresiones resultantes seguían siendo poco manejables. Como solución, se optó por reescribir manualmente la matriz 6x4 en el código de Python con expresiones simplificadas, permitiendo una implementación más eficiente, pero más demandante en términos de tiempo al escribir todas las posiciones manualmente, y corroborar una por una cuando daba algún valor diferente.

4. Inconsistencias en la comunicación entre Nodos

Finalmente, se encontraron problemas en la comunicación entre nodos, debido a diferencias en los tipos de datos utilizados en los tópicos de ROS2. Un error recurrente surgió cuando el nodo encargado del cálculo del Jacobiano operaba con vectores de tipo float64, mientras que los datos provenientes del nodo de control del joystick utilizaban un formato diferente. Esto provocó fallos en los cálculos y errores de compatibilidad. La solución implementada consistió en estandarizar todos los datos a float64, asegurando que cada nodo realizara la conversión necesaria antes de enviar o recibir información, garantizando así una comunicación consistente y libre de errores.

5. Implementación del nodo del Joystick

La integración del joystick al sistema presentó múltiples desafíos técnicos. Inicialmente, las librerías utilizadas y las pruebas realizadas en el primer computador dejaron de funcionar al trasladar la implementación al equipo final donde se trabajaría el control. Esto requirió un cambio de librerías y una reestructuración en la forma en que se abordaba el control del dispositivo.

Posteriormente, se encontraron problemas con el reconocimiento del joystick en los puertos del computador, lo que impedía su detección y uso dentro de ROS2. Una vez solucionado este inconveniente, surgieron dificultades al identificar correctamente qué datos correspondían a cada botón y determinar los rangos en los que se movían los controles. Esto era fundamental para la construcción del vector 6x1, el cual se publicaría como tópico para el cálculo del Jacobiano a partir del Δx recibido desde el joystick.

Otro problema crítico fue que si un joystick se movía antes que el otro, la recolección de datos se desordenaba, generando inconsistencias en la lectura de los valores. Se determinó que para garantizar la correcta inicialización del sistema, siempre se debía comenzar moviendo el

joystick izquierdo. Esto aseguraba que la lectura de cada uno de los botones y palancas fuera consistente con la configuración establecida desde el principio.