Resumen ejecutivo

Natalia Cely, Oscar Alvarado, Omar Pérez, Julian Pulido
ncelyc@unal.edu.co, *oalvaradob@unal.edu.co*, *operezo@unal.edu.co*, *jdpulidoca@unal.edu.co*
Universidad Nacional de Colombia

I. Logros

A lo largo del desarrollo del proyecto, se alcanzaron varios hitos importantes que demostraron la viabilidad y funcionalidad del sistema implementado. A continuación, se describen los principales logros obtenidos:

- **Control del robot mediante el mando:** Se logró utilizar los valores del mando (joystick) como entrada para controlar los movimientos y la lógica del robot. Este control se implementó desde otro PC conectado en la misma red, lo que permitió una interacción intuitiva y eficiente con el sistema de manera remota.
- **Sistema de visión remota:** Se implementó un sistema de visión remota que, en teoría, utiliza pocos recursos, no depende en gran medida del hardware utilizado y es de fácil acceso. Este sistema facilitó la supervisión y el control del robot desde ubicaciones remotas.
- **Desarrollo en ROS 2:** Se crearon paquetes y nodos en ROS 2, estableciendo una base sólida para la comunicación y el control del sistema. Además, se logró la conexión con los servos Dynamixel, permitiendo la escritura y lectura de datos en los registros de los servos.
- **Control articulado del robot:** Se implementó la capacidad de mover el robot de manera articulada, controlando cada una de sus juntas de forma individual. Esto permitió una mayor precisión en los movimientos y la ejecución de tareas específicas.
- **Implementación de la cinemática inversa:** Se desarrolló y aplicó la cinemática inversa del robot, permitiendo calcular las posiciones articulares necesarias para alcanzar una posición deseada en el espacio cartesiano. Esto se logró utilizando el método geométrico, teniendo en cuenta las longitudes de los eslabones del robot.
- **Simulación de una rutina preprogramada:** En un *livescript* de MATLAB, se implementó la simulación de una rutina preprogramada de tipo *pick and place*. Para ello, se desarrollaron dos funciones clave:
  - Una función para calcular la trayectoria mediante interpolación esférica, tomando como entrada un array de puntos en términos de coordenadas (x, y, z, ϕ).
  - Una función para calcular la cinemática inversa del robot en cualquier punto, utilizando las longitudes de los eslabones y el método geométrico para determinar los valores de las articulaciones en cada posición deseada.

  Además, se creó un nodo y un publicador en ROS 2 dentro del mismo *livescript*, permitiendo transmitir los valores articulares del Phantom Pincher a otro PC conectado en la misma LAN. Este mensaje contenía los valores articulares calculados, lo que facilitó la integración y comunicación entre sistemas.
- **Comunicación en red:** Se estableció un sistema de comunicación en red que permite enviar los valores articulares del robot desde MATLAB a otro dispositivo en la misma LAN. Esto se logró mediante la creación de un mensaje personalizado en ROS 2 y la configuración de un publicador dentro del *livescript*, demostrando la interoperabilidad entre MATLAB y ROS 2.

II. Pendientes

- **Interfaz unificada para dispositivos y modos de funcionamiento:** No se logró crear una interfaz gráfica o de usuario que integrara y combinara los diferentes dispositivos (joystick, cámaras, servos Dynamixel) y modos de funcionamiento (manual, automático) que se implementaron de manera independiente. Una interfaz unificada facilitaría la operación y supervisión del sistema de manera más intuitiva y eficiente.
- **Creación de rutinas personalizadas:** No se logró implementar la capacidad de crear y ejecutar rutinas diferentes de manera dinámica. Para ello, sería necesario definir una entrada que permita al usuario especificar los puntos de interés o seleccionar entre rutinas predefinidas (por ejemplo, dejar un objeto en diferentes ubicaciones). Con más tiempo, esta funcionalidad podría integrarse en la programación completa del robot, permitiendo un mayor avance en los objetivos principales del proyecto.

III. Dificultades

Durante el desarrollo del proyecto, se presentaron varias dificultades técnicas y de implementación que afectaron el funcionamiento óptimo del sistema. A continuación, se describen los principales problemas encontrados y las soluciones parciales o alternativas implementadas:

1. **Limitaciones en las interfaces de comunicación:** No fue posible implementar interfaces personalizadas para enviar estados a las articulaciones del

Phantom Pincher. Como solución, se utilizaron tipos de datos estándar disponibles en ROS, como arreglos, para la transmisión de información.

2. **Errores en el controlador del Phantom:** El controlador proporcionado por el repositorio de LabSir presentó errores de configuración y programación que dificultaron su uso directo. Esto llevó al equipo a implementar un controlador desde cero para el Phantom Pincher, lo que requirió un esfuerzo adicional en el desarrollo.
3. **Falta de control de velocidad:** No se logró implementar un control de velocidad efectivo, lo que causó oscilaciones en las juntas del robot. Para mitigar este problema, se aumentó el número de puntos en la trayectoria, reduciendo así los cambios bruscos en los valores articulares.
4. **Latencia en la red:** Se observó un retraso significativo entre el envío de comandos y la ejecución de los mismos, debido a la latencia en la red LAN. Esto dificultó el funcionamiento en modo manual, ya que los comandos no se ejecutaban de manera inmediata.
5. **Configuración manual de motores:** Fue necesario configurar manualmente la temperatura y el torque de los motores, lo que implicó forzar los motores para realizar ciertos movimientos. Esto aumentó el riesgo de desgaste prematuro de los componentes.
6. **Problemas con entornos de ROS:** Durante la instalación y configuración de los entornos de ROS, se presentaron problemas de corrupción en los builds de `colcon` para CoppeliaSim y RViz. Esto dificultó la simulación del robot utilizando estas herramientas.
7. **Incremento discreto de posición y velocidad:** Al analizar el comportamiento del incremento de posición de forma discreta, se observó que la velocidad aumenta de manera casi cuadrática con cada nuevo incremento. En el modo manual, el programa envía incrementos o decrementos de 2 unidades en cada paquete de articulaciones. Para cambios pequeños, esto no representa un problema, pero en cambios grandes se aprecia una limitante significativa.
   Dado que se estableció un límite de velocidad, los mensajes con incrementos de posición no corresponden con las velocidades requeridas para alcanzar la configuración deseada. Esto provoca que, al soltar el joystick, el robot se quede en una posición articular muy atrás de la última calculada, ignorando las configuraciones intermedias. Como resultado, el robot realiza un incremento brusco de velocidad para alcanzar la última posición calculada en el menor tiempo posible. Este problema se debe a que el tiempo de ejecución de una posición es más lento que el tiempo de lectura del mensaje (posición deseada), lo cual está relacionado con las limitantes físicas de los motores y el incremento constante y discreto de la posición.
8. **Problema similar en modo rutina automática:** En el modo de rutina automática, se presentó un problema similar al del modo manual. El tiempo de ejecución de una posición es más lento que el tiempo de lectura del mensaje (posición deseada). Para mitigar este problema, se implementaron pausas manuales entre la lectura de mensajes, permitiendo que el robot alcance primero la posición actual antes de leer el siguiente mensaje.
9. **Desafíos con el nodo del joystick:** Respecto al nodo del joystick, el análisis de la cantidad de datos generados, su filtrado e interpretación representó un desafío significativo. Fue necesario implementar técnicas de procesamiento de datos para manejar eficientemente la información recibida.
10. **Problemas con las cámaras:** Respecto a las cámaras, el tipo de dato que maneja el publicador requiere la dirección IP y la cantidad de fotogramas para operar correctamente. Además, la librería de visualización de Python utilizada para mostrar el video (`view`) consume mucho tiempo de procesamiento, lo que reduce la optimización del programa y afecta su rendimiento general.