Control de brazo robótico mediante IMUs

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Jose Manuel Illera Rodríguez

TUTORIZADO POR:
Moisés Díaz Cabrera
Jose Juan Hernández Quintana

Fecha: 06/2024

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a mis tutores Moisés Díaz y Jose Juan Hernandez. Sin su guía, paciencia y confianza en mí, no hubiera tenido la posibilidad de realizar este Trabajo de Fin de Grado. Su apoyo y sus enseñanzas han sido fundamentales para alcanzar esta meta.

A mi familia, especialmente a mis padres, quiero agradecerles profundamente por haber estado siempre a mi lado, tanto en los momentos buenos como en los malos. Su amor y su apoyo constante me han dado la fuerza necesaria para superar todos los obstáculos y seguir adelante.

También quiero dedicar un especial recuerdo a aquellos familiares que, aunque hoy no me acompañan, siempre creyeron en mi capacidad para lograr grandes cosas en la vida. Sus palabras y su confianza en mí han sido una fuente de inspiración constante.

Finalmente, a mis amigos más cercanos, gracias por estar siempre ahí. Sin lugar a dudas, sin su amistad, su apoyo y sus ánimos, no estaría donde estoy hoy. Su compañía ha hecho de este viaje algo mucho más llevadero y gratificante.

A todos ustedes, gracias de corazón.

Resumen

El ser humano, por naturaleza, busca aprender, mejorar y explorar tanto en entornos conocidos como desconocidos. Sin embargo, esto conlleva peligros y dificultades de acceso, lo que ha llevado al desarrollo de la teleoperación, un sistema en el que se utiliza un robot remoto controlado por un operador humano. En este proyecto, se ha diseñado un sistema teleoperado compuesto por sensores de captura de movimiento desarrollados por la empresa "Noitom" (Neuron Mocap) y un brazo robótico UR5e fabricado por la empresa "Universal Robots".

Para ello, se ha llevado a cabo un estudio detallado sobre el formato y la recopilación de los datos de los sensores, así como sobre el marco de trabajo que permite la operación de los seis ejes del brazo robótico. Los datos de Neuron Mocap se procesan y se envían al UR5e mediante ROS2, integrando en ambos sistemas para que el brazo robótico pueda replicar los movimientos capturados por los sensores. ROS2 (Robotic Operating System) es un framework de desarrollo de software para robots que proporciona la funcionalidad de un sistema operativo en un clúster heterogéneo.

El robot ha sido programado para imitar, en la medida de lo posible, los movimientos de la persona que utiliza los sensores, teniendo en cuenta las limitaciones de localización y la estructura del propio robot. Para este objetivo se han usado Matrices de Transformación Homogénea para asegurar la precisión y eficiencia en la imitación de los movimientos.

Finalmente, se ha llevado a cabo una evaluación de los resultados obtenidos y se han propuesto posibles mejoras para el futuro.

Palabras Claves: Teleoperación, sensores, UR5e, ROS2, Matrices de Transformación Homogéneas, ur_control, Neuron Mocap.

Abstract

Human beings, by nature, seek to learn, improve, and explore both known and unknown environments. However, this entails dangers and difficulties in access, which has led to the development of teleoperation, a system in which a remote robot is controlled by a human operator. In this project, a teleoperated system has been designed consisting of motion capture sensors developed by the company "Noitom" (Neuron Mocap) and a UR5e robotic arm manufactured by the company "Universal Robots."

To this end, a detailed study has been conducted on the format and collection of sensor data, as well as on the framework that allows the operation of the six axes of the robotic arm. Neuron Mocap data is processed and sent to the UR5e via ROS2, integrating both systems so that the robotic arm can replicate the movements captured by the sensors. ROS2 (Robotic Operating System) is a software development framework for robots that provides the functionality of an operating system in a heterogeneous cluster.

The robot has been programmed to mimic, as closely as possible, the movements of the person using the sensors, taking into account the localization limitations and the structure of the robot itself. For this purpose, Homogeneous Transformation Matrices have been used to ensure accuracy and efficiency in imitating the movements.

Finally, an evaluation of the obtained results has been carried out and possible improvements for the future have been proposed.

Keywords: Teleoperation, sensors, UR5e, ROS2, Homogeneous Transformation Matrices, ur_control, Neuron Mocap.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Objetivos

El objetivo principal de este proyecto consiste en diseñar y desarrollar un sistema que posibilite el control de un brazo robótico mediante el uso de Unidades de Medición Inercial (IMUs).

Para alcanzar este propósito, se han establecido los siguientes objetivos específicos:

1. **Analizar y estudiar en profundidad las tecnologías y herramientas a implementar**: Asegurar una base sólida para el desarrollo del sistema mediante un análisis exhaustivo de las tecnologías y herramientas necesarias.

2. **Desarrollar e integrar un sistema de comunicación robusto y confiable**: Facilitar la interacción efectiva entre las IMUs y el brazo robótico mediante un sistema de comunicación bien diseñado.

3. **Programar y configurar el controlador del brazo robótico**: Garantizar una respuesta precisa y eficiente a las instrucciones recibidas a través de las IMUs mediante la correcta programación y configuración del controlador del brazo robótico.

4. **Realizar pruebas exhaustivas**: Asegurar el correcto funcionamiento del sistema mediante pruebas exhaustivas y elaborar una documentación

detallada que sirva de guía y referencia para futuros proyectos similares.

5. **Desarrollar la capacidad del brazo robótico para imitar los movimientos del brazo humano**: Una vez establecida una comunicación efectiva entre las IMUs y el brazo robótico, se añadió este objetivo. Este objetivo presenta desafíos significativos debido a la precisión y adaptabilidad necesarias. No obstante, se busca lograr que el brazo robótico pueda replicar con precisión los movimientos del brazo humano.

El conjunto de estos objetivos se dirige a la consecución de un sistema de control robótico que cumpla con las expectativas de funcionamiento y precisión, y que contribuya al avance en el campo de la robótica. Este proyecto se configura como una oportunidad para innovar y aportar en el campo de la ingeniería robótica.

1.2. Motivación

Este proyecto de control para un brazo robótico se inspira en el deseo de replicar la complejidad del movimiento humano en máquinas. Su objetivo es avanzar en la robótica, enfocándose especialmente en el desarrollo de nuevas técnicas de control. El Trabajo de Fin de Grado (TFG) abre posibilidades prácticas en áreas como la medicina, la asistencia doméstica, el arte y el entretenimiento, y además promueve la integración de la robótica en la educación.

Las principales motivaciones para este TFG incluyen:

- Mejorar la comprensión y aplicación de nuevas técnicas de control robótico.
- Establecer una base para futuras investigaciones y desarrollos en el campo de la robótica.
- Ofrecer una herramienta educativa que fomente un aprendizaje interactivo y práctico de la robótica.

Por lo tanto, este proyecto actúa como un puente entre la teoría robótica y su aplicación práctica, motivando a futuras generaciones de ingenieros y diseñadores de robótica.

1.3. Estructura de la Memoria

El presente documento se organiza en siete capítulos, distribuidos de la siguiente manera:

1. **Introducción**: Se presenta la motivación y necesidad del proyecto, se establecen los objetivos generales y se proporciona una visión global del desarrollo y estructura del trabajo.

2. **Marco Teórico**: Se exponen los conceptos fundamentales en robótica, teleoperación y se realiza una revisión de la literatura y proyectos previos relacionados.

3. **Arquitectura del Hardware y Software**: Se describe el diseño y la estructura del hardware y software desarrollado para el control del brazo robótico, incluyendo las plataformas y herramientas utilizadas.

4. **Análisis Cinemático**: Se examina la cinemática del brazo humano y del brazo robótico, incluyendo modelos matemáticos que describen posiciones y movimientos del UR5e basados en ángulos.

5. **Desarrollo**: Se explica el proceso de implementación del sistema, desde la configuración inicial hasta las pruebas finales. Se incluyen los métodos de programación y la integración del software.

6. **Evaluación y Discusión**: Se expone las bases del experimento y los resultados obtenidos. Se discuten las fortalezas y debilidades del sistema desarrollado, comparando la experiencia de distintos sujetos. Asimismo, se identifican las áreas que requieren mejoras y se proponen posibles soluciones.

7. **Conclusiones**: Se resume el trabajo realizado, destacando los logros y resultados más relevantes del proyecto y se proponen recomendaciones para futuras investigaciones.

- **Apéndices**: Se proporcionan detalles técnicos adicionales que complementan la información del proyecto.

- **Referencias**: Se listan todas las fuentes bibliográficas y recursos consultados para el desarrollo del TFG.

Cada capítulo está diseñado para proporcionar una comprensión clara y completa del proyecto, facilitando el acceso a información específica y detallada para el lector.

1.4. Competencias adquiridas

A lo largo del desarrollo de este Trabajo de Fin de Grado, se han empleado diversas competencias específicas del Grado en Ingeniería Informática. A continuación, se enumeran y justifican las competencias que han encontrado aplicación directa en el proyecto.

Tabla de Competencias Aplicadas

Código	Descripción
CI1	Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.
CI6	Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.
CI9	Capacidad de conocer, comprender y evaluar la estructura y arquitectura de los computadores, así como los componentes básicos que los conforman.
TI7	Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos.

Tabla 1.1: Competencias específicas aplicadas en el TFG

Justificación de Competencias

La relevancia de estas competencias para el TFG se justifica de la siguiente manera:

- **CI1:** La creación de un sistema de control para un brazo robótico con IMUs implica una profunda comprensión de los requisitos de software y hardware, así como de los estándares éticos y normativos que garantizan la seguridad y efectividad del producto.

- **CI6:** La resolución de problemas de control de movimiento del brazo robótico es central en el TFG. Esto implica la aplicación de algoritmos básicos de tecnologías informáticas y la evaluación de su adecuación y complejidad, una competencia clave para el diseño de soluciones efectivas.

- **CI9:** El desarrollo del proyecto requirió una comprensión de la arquitectura de computadores y la integración de componentes hardware para el control del brazo robótico.

- **TI7:** Comprender y aplicar los principios de garantía y seguridad de los sistemas informáticos es crucial, especialmente si el brazo robótico se utiliza en entornos donde la seguridad es primordial, como en aplicaciones médicas o de investigación.

1.5. Metodología de desarrollo

La metodología de desarrollo empleada en este Trabajo de Fin de Grado se basó en un enfoque de cascada con opción a retroceso, caracterizado por su naturaleza secuencial y sistemática. El modelo en cascada facilitó una gestión estructurada y organizada del proyecto, donde cada fase debía completarse antes de avanzar a la siguiente. Este enfoque tradicional se seleccionó por su simplicidad y eficacia en proyectos con requisitos bien definidos y establecidos desde el inicio. Aunque se tenía un objetivo claro a alcanzar en cada fase, se revisaba lo hecho anteriormente para asegurar la compatibilidad con las siguientes partes.

La planificación del trabajo se detalla en el siguiente diagrama de Gantt1.1.

El proyecto comenzó con un análisis preliminar exhaustivo del estado del arte, investigando diferentes sistemas de teleoperación documentados en la literatura científica para destacar ventajas y desventajas de cada sistema. Simultáneamente, se evaluaron las capacidades del sistema de captura de movimiento Perception Neuron para verificar su compatibilidad con los requisitos específicos de la aplicación.

Posteriormente, se desarrolló la arquitectura de software utilizando el entorno de ROS. Dada la importancia de ROS para la implementación, se invirtió un tiempo significativo en la capacitación y el aprendizaje profundo del entorno. Desde mediados de junio hasta principios de septiembre, el enfoque principal fue el desarrollo del algoritmo de decodificación que constituye la base del sistema de control de teleoperación de la mano robótica. Luego, el trabajo se amplió para incluir el control de teleoperación del brazo robótico, subdividiendo esta tarea en varias subtareas más específicas, reflejadas en el diagrama de Gantt.

La etapa final del proyecto se centró en la evaluación del rendimiento de la plataforma de teleoperación mano-brazo desarrollada, realizando pruebas

para asegurar que se cumplían los objetivos propuestos.

El modelo en cascada con opción a retroceso permitió revisar y ajustar fases previas del proyecto en caso de encontrar problemas o requerir cambios.

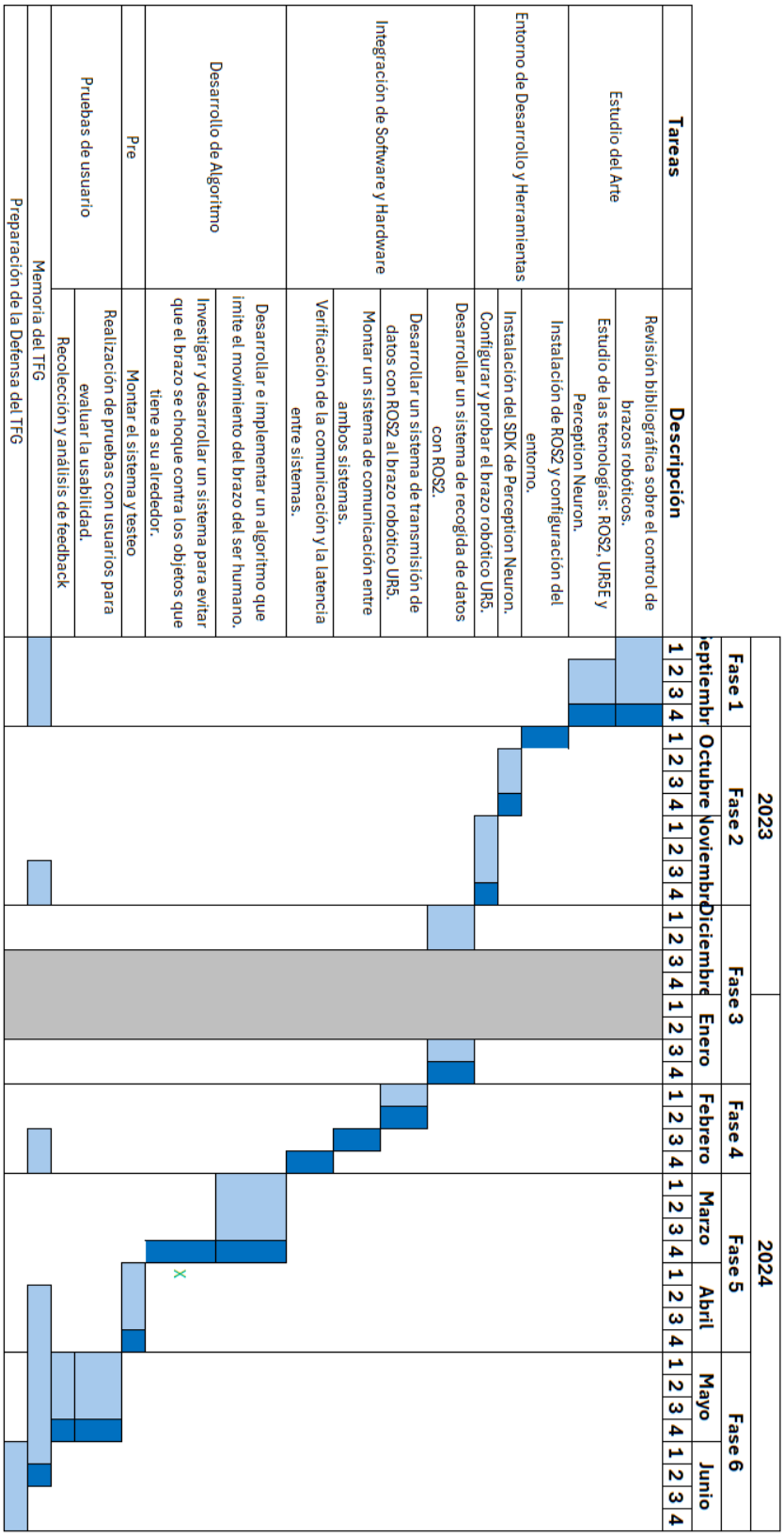

Figura 1.1: Planificación de actividades del TFG según la metodología en cascada.

Capítulo 2

Marco teórico

Desde 1960, la robótica ha experimentado una significativa evolución debido al interés continuo en desarrollar máquinas que imiten funciones y movimientos de seres vivos. A principios del 2000, los robots comenzaron a integrarse en entornos cotidianos como hospitales, supermercados e industrias, alcanzando un nivel operativo cada vez más avanzado y aumentando su demanda. Un informe de la Federación Internacional de Robótica (IFR) titulado "Why service robots are booming worldwide" resalta esta tendencia, indicando que "los robots están claramente en aumento, tanto en la fabricación como en entornos cotidianos"[1].

Además, se anticipa que los robots personales jugarán un papel importante en la transformación de la vida cotidiana, similar a la evolución observada en la industria de los computadores, como lo destaca Bill Gates en su artículo "A Robot in Every Home"[2] .

La presencia de robots en la vida diaria también impulsa investigaciones en la interacción humano-robot (HRI), con el objetivo de comprender, diseñar y evaluar sistemas robóticos que colaboren con humanos. Las investigaciones en HRI involucran múltiples disciplinas como ingeniería mecánica, eléctrica, ciencias de la computación, y control e inteligencia artificial, y se centran en áreas clave[3]:

1. Supervisión humana de robots para tareas rutinarias.

2. Control remoto de vehículos en entornos inaccesibles o peligrosos.

3. Vehículos automatizados donde los humanos son pasajeros.

4. Interacción social entre humanos y robots para asistencia y entretenimiento.

En resumen, la robótica actual está marcada por una rápida evolución tecnológica que amplía constantemente los límites de lo posible, transformando industrias y la vida cotidiana en todo el mundo.

2.1. Teleoperación

La teleoperación se refiere al control remoto de máquinas o robots 2.2 en entornos inaccesibles o peligrosos para los humanos[4], como áreas con alta radiactividad, fondos submarinos, minas subterráneas y el espacio exterior. Esta tecnología permite a los operadores realizar operaciones complejas desde un lugar seguro, usando interfaces como joysticks, pantallas y sistemas de retroalimentación sensorial para simular una presencia física en el lugar del robot[5].

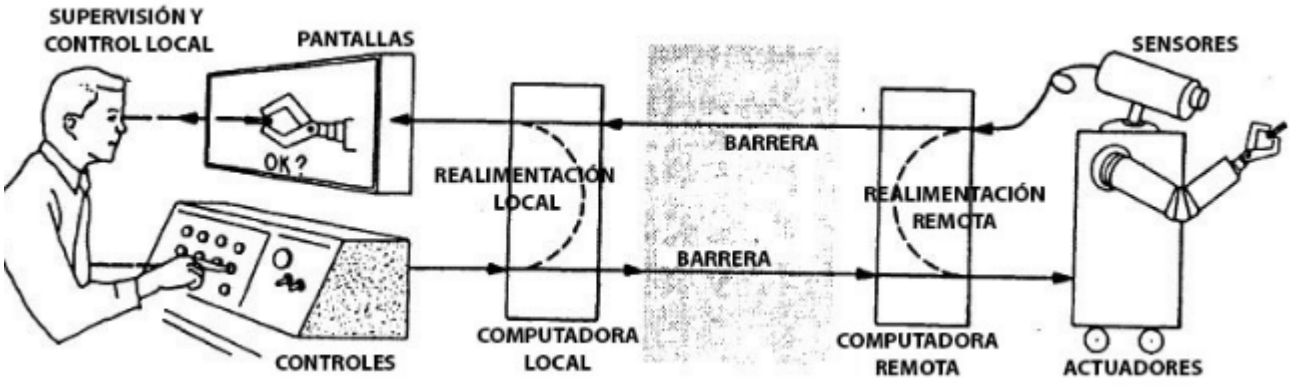

Figura 2.1: Sistema de Teleoperación.

Un sistema teleoperado se compone de dos partes principales: el entorno del operador y el entorno remoto. En el entorno del operador, una persona supervisa y controla la operación a distancia. La intervención del operador puede variar desde control constante hasta monitoreo ocasional, con responsabilidades que incluyen la señalización de objetivos y la planificación en intervalos específicos. En el entorno remoto, el dispositivo teleoperado (manipulador, robot, vehículo u otro dispositivo) realiza tareas específicas bajo la dirección del operador[6].

Las estrategias de teleoperación se dividen en tres categorías principales[6]:

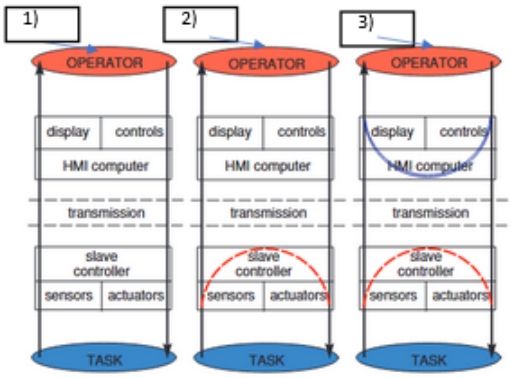

Figura 2.2: Estrategia de la Teleoperación

1. **Control en circuito cerrado:** El operador maneja los actuadores directamente y recibe retroalimentación en tiempo real, viable cuando los retardos son mínimos. Un ejemplo es un automóvil controlado por radio.

2. **Teleoperación coordinada:** El operador controla los actuadores con un circuito de control interno en el teleoperador, sin autonomía en el lado remoto. Los bucles remotos compensan los retardos. Un ejemplo es un teleoperador con control de velocidad remoto.

3. **Control supervisor:** La mayor parte del control está en el teleoperador, que realiza tareas parcialmente autónomas. El operador supervisa y da comandos de alto nivel. Este tipo de control también se conoce como "teleoperación basada en tareas".

2.2. Áreas de aplicaciones de la teleoperación

La teleoperación ha encontrado aplicaciones significativas en diversos sectores. Su capacidad para controlar remotamente máquinas en entornos inaccesibles o peligrosos ha revolucionado múltiples campos. A continuación, se describen las áreas de aplicación más relevantes[7].

2.2.1. Aplicación en el espacio

Las aplicaciones en el espacio presentan varias razones para utilizar la teleoperación como técnica de manipulación remota. La presencia física de un ser humano implica altos costos y riesgos debido al ambiente hostil y a la duración de las misiones, que a menudo se extienden por muchos años y se realizan sin tripulación humana.

Entre las principales aplicaciones espaciales se encuentran la experimenta-

ción y exploración planetaria, normalmente con un vehículo tipo Rover, así como el mantenimiento y operación de satélites, y la construcción y mantenimiento de estaciones espaciales, como los brazos robóticos en el espacio exterior.

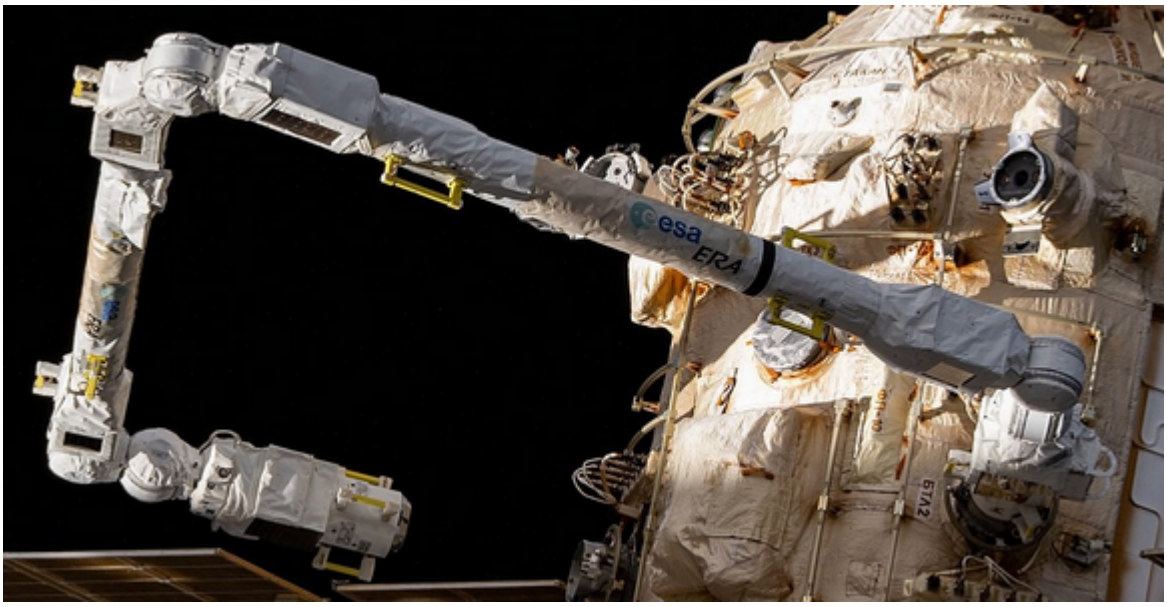

Figura 2.3: El brazo robótico europeo con control de ERA

2.2.2. Aplicaciones submarinas

En este caso, la mayoría de los manipuladores están montados sobre vehículos submarinos denominados ROV (Remote Operated Vehicle), que también son teleoperados. La utilidad de estos sistemas radica en su capacidad para acceder a zonas y profundidades donde es imposible o peligroso para un submarinista. Entre las principales aplicaciones se encuentran la inspección, mantenimiento y construcción de instalaciones submarinas, la minería submarina y la inspección del suelo marino.

Uno de los ejemplos claros de esta aplicación es VICTOR, un sistema francés de exploración submarina.

2.2.3. Aplicaciones en cirugía

Recientemente, se ha observado un notable avance en la aplicación de tecnologías de teleoperación en el ámbito médico. Desde los primeros desarrollos de prótesis o dispositivos de asistencia para discapacitados hasta las más innovadoras prácticas de telecirugía[8].

En este ámbito, la primera operación teleoperada significativa utilizando el sistema quirúrgico robótico ZEUS se realizó el 7 de septiembre de 2001. El

Figura 2.4: Sistema de exploración submarina VICTOR

sistema robótico ZEUS fue manejado por un cirujano en Manhattan, Nueva York, mientras que la parte esclava o del paciente se encontraba en Satena, Francia[9].

Figura 2.5: Zeus y Da Vinci, aplicación robótica para la operación teleoperada.

2.2.4. Otras aplicaciones

Entre otras aplicaciones también se encuentran:

- Militares
- Asistencia
- Educación
- Minería

La expansión de la teleoperación a diversos sectores ha demostrado su versatilidad y capacidad para mejorar la eficiencia y seguridad en ambientes que presentan riesgos significativos para la intervención humana. Esta tecnología continúa evolucionando, impulsando no solo el progreso técnico sino también ofreciendo nuevas posibilidades. Con cada avance tecnológico, la teleoperación se establece cada vez más como una herramienta indispensable en el panorama moderno de la automatización y la robótica.

2.3. Panorama Histórico

En 1948, R. C. Goertz, del Argonne National Laboratory, desarrolló el primer sistema de telemanipulación con el objetivo de manejar elementos radiactivos sin riesgo para el operador[10]. Este sistema consistía en un dispositivo mecánico maestro-esclavo. Años más tarde, en 1954, Goertz aplicó tecnología electrónica y servo control, reemplazando la transmisión mecánica por una eléctrica y desarrollando el primer sistema de telemanipulación con servo control bilateral.

En 1954, C. W. Kenward solicitó la primera patente de un dispositivo robótico, emitida en 1957. Sin embargo, George C. Devol sentó las bases del robot industrial moderno al idear un dispositivo de transferencia de artículos en 1954, patentado en 1961. Junto con Joseph F. Engelberger, Devol llevó estas máquinas a la aplicación industrial, estableciendo la Consolidated Controls Corporation, posteriormente conocida como Unimation[11]. En 1961, implementaron la primera máquina Unimate en la fábrica de General Motors en Trenton, Nueva Jersey, para aplicaciones de fundición por inyección.

Para fines comerciales, estas máquinas comenzaron a denominarse robots, destinados a transferir piezas de manera versátil o universal, impulsando su

Figura 2.6: Sistema de telemanipulación bilateral

Figura 2.7: Uno de los primeros Unimate

aceptación industrial y proyectando una imagen de modernismo y avance tecnológico.

A finales de los años 60 y principios de los 70, se establecieron los cimientos de la investigación robótica en las universidades. Surgieron los primeros robots móviles con un cierto grado de autonomía, como Shakey del Stanford Research Institute y el Stanford Cart de la Universidad de Stanford.

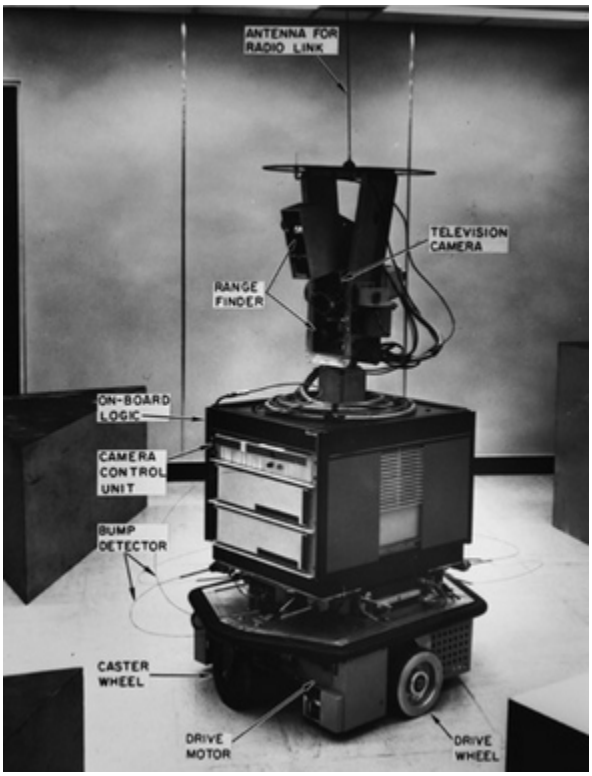

Figura 2.8: Robot Shakey

Víctor Sheinman desarrolló el brazo de Stanford, el primer manipulador controlado por computadora y con accionamiento eléctrico, que más tarde dio origen al famoso robot industrial PUMA.

Figura 2.9: Robot PUMA

En 1973, la firma sueca ASEA construyó el primer robot con accionamiento totalmente eléctrico, el robot IRB6.

Figura 2.10: Robot IRB6 de la firma sueca ASEA

En 1980, se fundó la Federación Internacional de Robótica en Estocolmo, Suecia. En 1982, el profesor Makino de la Universidad Yamanashi en Japón introdujo el concepto del robot SCARA (Selective Compliance Assembly Robot Arm), diseñado para tener un número reducido de grados de libertad, un coste limitado y una configuración orientada al ensamblado de piezas.

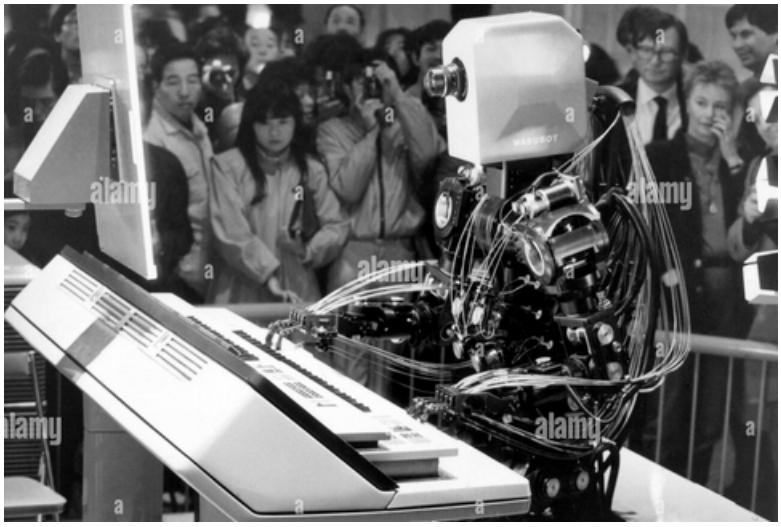

Figura 2.11: Wasubot, robot de la Universidad de Waseda

En 1981, la Universidad Carnegie Mellon desarrolló el accionamiento di-

recto, conectando directamente los motores a las articulaciones, eliminando la necesidad de reductores y permitiendo movimientos más rápidos y precisos.

Además de los entornos de fabricación e investigación, en la década de 1970, los robots comenzaron a ser utilizados de manera práctica en otros ámbitos como el espacial y el submarino, utilizando principalmente tecnología de teleoperación.

Tabla 2.1: Algunos hitos de la robótica.

Año	**Descripción**
1985	El Profesor Ichiro Kato de la Universidad de Waseda creó el Wasubot, robot humanoide capaz de tocar partituras con un piano.
1999	Sony presentó su robot mascota con forma de perro denominado Aibo.
1970 / 2009	Llega a la Luna el primer rover Lunojod 1. Primer robot que aterriza en un cuerpo celeste. Existen dos astromóviles en activo, Spirit y Opportunity.
2005	Boston Dynamics presentó su primer robot BigDog, un vehículo terrestre no tripulado o robot andador, cuadrúpedo y dinámicamente estable, para uso militar.
2013 / Actualidad	Atlas, un robot humanoide bípedo desarrollado por Boston Dynamics, diseñado para tareas de búsqueda y rescate. En 2016 se presentó su nueva versión, especializada en manipulación móvil y en caminar sobre una amplia gama de terrenos, incluida la nieve.
2014	Presentación de Pepper, un robot semihumanoide fabricado por SoftBank Robotics (antes Aldebaran Robotics), diseñado con la capacidad de leer emociones.
2015	Presentación de Sophia, desarrollada por Hanson Robotics, con sede en Hong Kong. Sophia está diseñada para aprender, adaptarse al comportamiento humano y trabajar satisfactoriamente con personas.
2018	Creación de Nao Robot, un robot humanoide programable y autónomo, desarrollado por Aldebaran Robotics.
2020 / 2021	Perseverance, un vehículo Mars rover fabricado por el Laboratorio de Propulsión a Reacción, parte de la misión espacial Mars 2020 del Programa de Exploración de Marte de la NASA.

2.4. Robot

La evolución de los sistemas teleoperados hacia el uso de programas informáticos para controlar manipuladores marcó la transición hacia los robots. En el campo de la robótica, la definición de “robot” se actualiza con frecuencia.

Aplicaciones tecnológicas modernas, como casas automatizadas y automóviles con capacidades avanzadas de interacción humana, han pasado de ser cien-

cia ficción a realidades dentro de la robótica[10].

Tabla 2.2: Definición de robot

Enciclopedia Británica	Robot, cualquier máquina operada automáticamente que reemplaza el esfuerzo humano, aunque no se parezca a los seres humanos en apariencia ni realice funciones de manera humana[12].
Diccionario Merriam Webster	Robot, una máquina que se parece a un ser vivo porque es capaz de moverse de forma independiente (como caminar o rodar sobre ruedas) y realizar acciones complejas (como agarrar y mover objetos)[13].
Diccionario de la Real Academia Española	Robot, máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones[14].

Con el avance tecnológico en robótica, incluso estas definiciones pueden ser insuficientes para describir completamente los sistemas modernos considerados como robots. Por esta razón, es común especificar el tipo de robot, como manipuladores, humanoides, domésticos, aéreos, submarinos, caminantes o tele-robots.

Hasta finales de los años 80, los sistemas robóticos eran limitados, siendo los utilizados en manufactura de productos en talleres y líneas de producción los únicos ejemplos de robots reales fuera de los laboratorios, excluyendo los elementos de ciencia ficción. Estos robots industriales se especializaban en manipular piezas o herramientas en entornos industriales.

Capítulo 3

Arquitectura del Hardware y Software

3.1. Arquitectura del Hardware

En este capítulo se detalla la arquitectura de hardware empleada para el proyecto, destacando cómo cada componente interactúa y se integra dentro del sistema global. Se describirán los dispositivos específicos utilizados y su configuración.

La Figura muestra el esquema de la arquitectura de hardware, ilustrando la interconexión y la función de cada componente dentro del sistema global de teleoperación. Esta representación esquemática facilita la comprensión de cómo se integran los diversos elementos hardware para lograr la captura de movimiento y el control robótico.

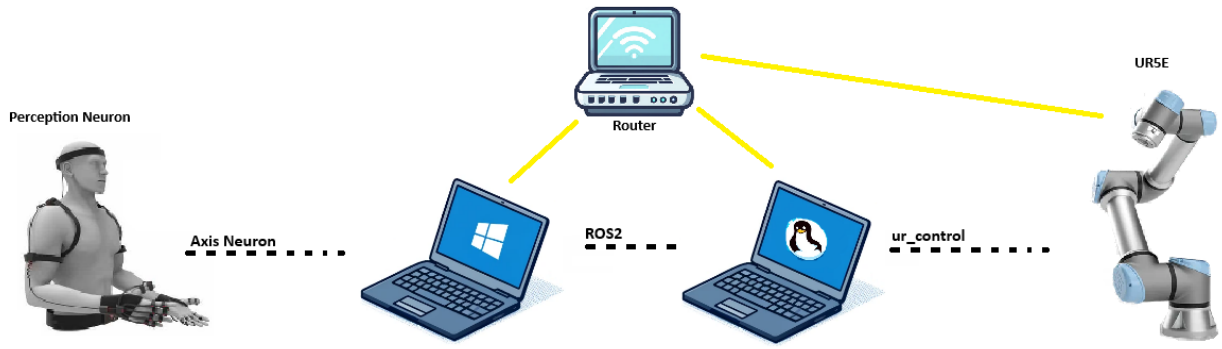

Figura 3.1: Esquema de la arquitectura de hardware para el sistema de teleoperación.

- **Perception Neuron:** Este sistema avanzado de captura de movimiento transmite los datos capturados al portátil Windows, utilizando la red

interna provista por el router.

- **Portátil Windows:** Equipado con ROS2, recibe y procesa los datos recibidos de Perception Neuron y los envía al portátil Linux para su gestión, a través de la misma red interna.

- **Portátil Linux:** Equipado con ROS2, recibe y procesa los datos del portátil Windows. Utiliza esta información para calcular los parámetros de control, que luego envía al brazo robótico UR5e.

- **UR5e:** El brazo robótico recibe comandos de control desde el portátil Linux y ejecuta acciones en el entorno físico.

3.1.1. Sistema Teleoperación

Sistema Perception Neuron Mo-CAP

Las mediciones cualitativas o cuantitativas son indispensables para cualquier procedimiento relacionado con la captura de movimiento humano (Mo-Cap). Mientras que el análisis cualitativo se refiere a la descripción y análisis de características que no se pueden expresar en términos numéricos, el cuantitativo exige evaluar variables biomecánicas, como ángulos posturales, distribución de presión, momentos y fuerzas generadas por el cuerpo humano [15].

Figura 3.2: Distintas técnicas de captura de movimiento

Existen diversas técnicas de captura que se adaptan a diferentes propósitos y contextos[16]:

1. Óptica (pasiva): Con esta técnica, se adhieren marcadores retrorreflectantes a los cuerpos u objetos, y reflejan la luz generada cerca del objetivo

de la cámara. Una vez reflejada, la luz se utiliza para calcular la posición de los marcadores dentro de un espacio tridimensional, y se graba.

2. Óptica (activa): Esta técnica es exactamente igual, pero los marcadores emiten luz en lugar de reflejarla. Por lo tanto, los marcadores necesitan una fuente de energía.

3. Sin marcadores: esta técnica no requiere marcadores de ningún tipo. Se basa en cámaras sensibles a la profundidad y en un software especializado para seguir y grabar a personas y objetos en movimiento.

4. Inercial: Esta técnica no necesita necesariamente cámaras para funcionar. Registra el movimiento a través de las IMU (unidades de medición inercial), que contienen sensores para medir y capturar datos.

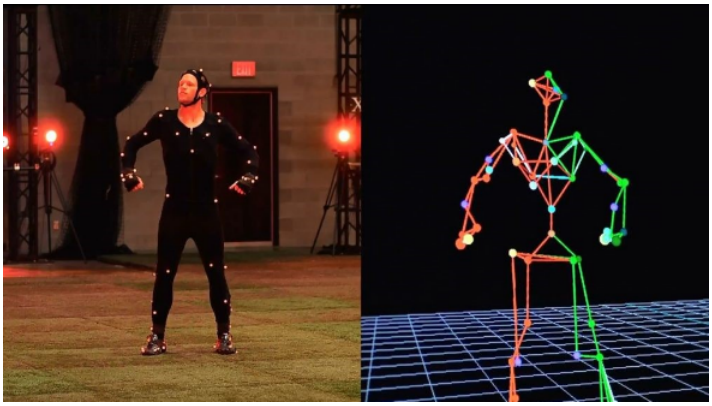

Figura 3.3: Actor equipado con un traje de captura de movimiento óptico (por infrarrojos) y el esqueleto digital generado

Para este proyecto se escogió el sistema Perception Neuron[17] , un sistema inercial, fue seleccionado como la plataforma de captura de movimiento debido a su eficacia y flexibilidad. A diferencia de las técnicas ópticas, tanto activas como pasivas, que requieren un entorno controlado con iluminación y cámara específicas, Perception Neuron ofrece la ventaja de la portabilidad y la independencia de un entorno de estudio específico.

El sistema incluye varios componentes clave que trabajan en conjunto para capturar con precisión el movimiento humano. Cada componente cumple una función específica dentro del sistema de captura de movimiento:

El **Neuron Sensor** es una Unidad de Medición Inercial (IMU), que está compuesta por un giroscopio, un acelerómetro y un magnetómetro. Estos sensores capturan datos tridimensionales sobre la orientación, la aceleración

y el campo magnético alrededor del sensor, lo que es crucial para el registro preciso del movimiento.

Figura 3.4: Inertial Measurement Unit (UMI) compuesto por un giroscopio, un acelerómetro y un magnetómetro

El **Hub** es el dispositivo central que recoge los datos de todos los Neuron Sensors conectados. Este dispositivo procesa la información recibida y la envía al computador a través de una conexión USB o por conectividad inalámbrica, facilitando así un flujo de datos constante y en tiempo real para el análisis y la visualización del movimiento.

Figura 3.5: HUB, aparato que hace de puente entre los IMU y el Axis Neuron

Los **Straps** son correas ajustables que aseguran los Neuron Sensors al cuerpo del usuario. Cada correa está claramente etiquetada en la parte trasera para indicar su posición correcta sobre el cuerpo, asegurando así la colocación adecuada de los sensores para una captura de movimiento óptima.

La transferencia de datos se efectúa a través de un “Cable USB”, que conecta los dispositivos al sistema de captura de movimiento o por conexión Wi-Fi.

Portátil Windows

El sistema Perception Neuron transfiere los datos capturados a un portátil Windows que desempeña un papel fundamental en el procesamiento inicial antes de transmitir la información al sistema Linux para el control del brazo robótico. A continuación, se detallan las especificaciones técnicas del portátil Windows utilizado en el proyecto:

- **Nombre del dispositivo:** El portátil es identificado como DESKTOP-P83E7EQ, una nomenclatura que puede ser útil para la administración de red y el soporte técnico.

- **Procesador:** Equipado con un Intel(R) Core(TM) i5-5200U CPU a 2.20GHz, este procesador dual-core ofrece un rendimiento adecuado para tareas de procesamiento de datos en tiempo real.

- **RAM instalada:** 8.00 GB, proporcionando suficiente memoria para manejar las aplicaciones de captura de movimiento y procesamiento sin retardos significativos.

- **Sistema operativo:** Windows 10 Pro, lo que garantiza compatibilidad con una amplia gama de software especializado y proporciona características de seguridad avanzadas necesarias para proteger los datos sensibles.

- **Tipo de sistema:** Sistema operativo de 64 bits, procesador basado en x64, ofreciendo una buena gestión de la memoria y eficiencia en el procesamiento.

Este portátil es clave en el esquema de teleoperación debido a su capacidad para manejar tareas intensivas de procesamiento, facilitando una transición fluida y efectiva de los datos hacia el siguiente eslabón en la cadena de procesamiento del sistema de teleoperación.

3.1.2. Sistema Teleoperado

Universal Robot UR5e

El UR5e es un robot colaborativo industrial, destacado por su versatilidad y ligereza, diseñado para adaptarse a una amplia variedad de tareas de nivel medio con notable flexibilidad. Su diseño permite una integración fluida en múltiples aplicaciones, proporcionando una solución adaptable y eficiente para un extenso rango de requerimientos industriales.

3.1.2.1. Características Técnicas del UR5e

Teniendo en cuenta Anexo A: Ficha técnica UR5e, el consumo máximo de energía del UR5e es de 570 W, con un consumo típico de 200 W durante operaciones moderadas. Esto subraya su eficiencia energética en entornos industriales. Además, el robot cumple con las normativas de seguridad EN ISO 13849-1, PLd Categoría 3, y EN ISO 12100.

Figura 3.6: Portatil utilizado para el sistema Windows.

Figura 3.7: UR5e, brazo robótico.

Tiene una precisión de repetibilidad de ± 0.03 mm, lo que garantiza movimientos exactos y consistentes. El rango de movimiento de sus seis ejes es de 360°.

La velocidad máxima del punto de control del robot (TCP) es de 1 m/s,

lo que facilita operaciones rápidas y eficientes.

El UR5e opera con un nivel de ruido inferior a 65 dB(A), permitiendo su uso en entornos donde el ruido debe mantenerse al mínimo. Su interfaz cuenta con varias entradas y salidas digitales y analógicas, lo que le permite interactuar eficazmente con otros componentes del sistema automatizado.

- Entradas digitales: 16
- Salidas digitales: 16
- Entradas analógicas: 2
- Salidas analógicas: 2

La huella del UR5e es de Ø 149 mm, con materiales que incluyen aluminio, plástico y acero, asegurando durabilidad y robustez. Además, el robot puede operar en temperaturas de 0-50°C y tiene un rango de humedad operativa del 98 % RH sin condensación.

Componentes y Motores del UR5e

El UR5e, un robot colaborativo de seis ejes, está compuesto por varias partes cruciales que trabajan en conjunto. Cada uno de los seis ejes está impulsado por su propio motor, que permite movimientos precisos y controlados. A continuación, se detalla cada componente y motor del UR5e:

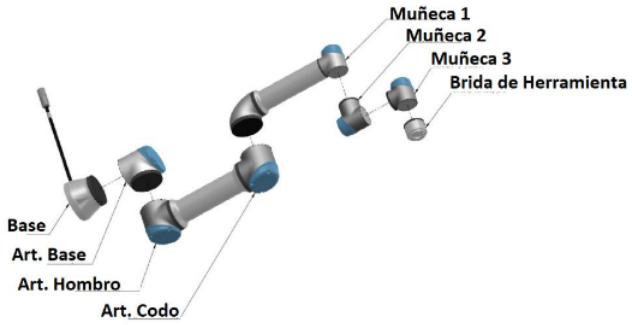

Figura 3.8: UR5e, segmentos del robot industrial de 6 ejes.

Interfaz y Programación

Equipado con la interfaz de programación PolyScope, el UR5e permite una programación accesible incluso para usuarios sin experiencia previa en robótica. PolyScope proporciona una programación intuitiva a través de una interfaz gráfica que guía al usuario en la configuración y operación del robot.

Configuración de Red y Control Remoto

Integrar el UR5e en una red privada requiere ajustes adecuados de los parámetros de red. Esto incluye configurar una dirección IP estática o utilizar DHCP, según las necesidades del entorno de red, además de ajustar la máscara de subred y la puerta de enlace predeterminada para asegurar la conectividad.

La activación del control remoto, necesaria para la comunicación desde el portátil Linux, en PolyScope permite operar el robot desde ubicaciones remotas, facilitando la supervisión y el control sin la necesidad de presencia física. Esto se logra habilitando la función de control remoto en los ajustes de PolyScope y seleccionando el perfil de control remoto para permitir la operación a distancia.

1. En el encabezado de PolyScope, seleccionar el menú Hamburgesa y acceder a *Ajustes*.

2. Dentro de *Ajustes*, seleccionar *Control Remoto*.

3. Activar la función para habilitar el control remoto. Esta opción permite que el robot sea accesible y controlable desde un tercer dispositivo.

4. Configurar los parámetros de red adecuados, incluyendo la dirección IP estática o DHCP, para asegurar la conectividad efectiva con el dispositivo remoto.

Sistema de control propio del UR5e: PolyScope

PolyScope[18] se presenta como la interfaz gráfica de usuario (IGU) diseñada para facilitar la manipulación del brazo robótico y la caja de control, permitiendo la ejecución de programas existentes y la creación de nuevos de manera sencilla.

Características

- **Interfaz Gráfica de Usuario:** PolyScope ofrece una interfaz gráfica basada en un sistema de menús intuitivos y pantallas táctiles, lo que simplifica la configuración inicial del robot y la programación de tareas.

- **Programación basada en bloques:** Los usuarios pueden arrastrar y soltar bloques de funciones para construir secuencias de operaciones,

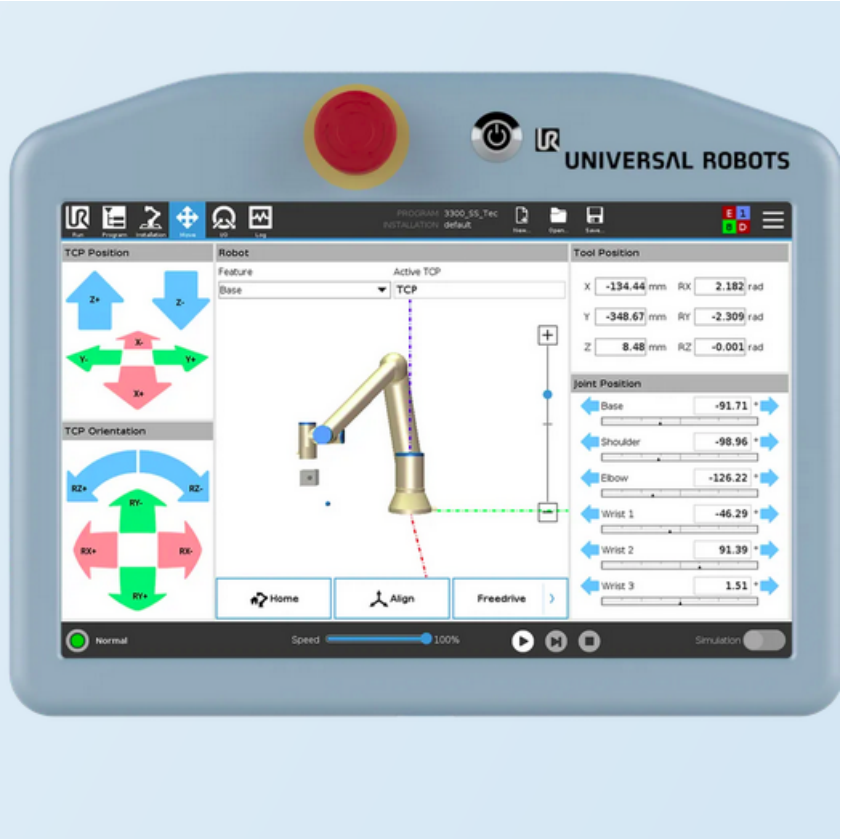

Figura 3.9: PolyScope, sistema de control propio del UR5e

reduciendo la complejidad de la programación.

- **Compatibilidad con accesorios:** PolyScope soporta la integración de numerosos accesorios y periféricos, permitiendo a los usuarios ampliar las capacidades del robot para tareas específicas.

- **Simulación y prueba:** La interfaz proporciona herramientas para simular y probar los programas antes de su ejecución real, aumentando la seguridad y reduciendo los errores operativos.

Portátil Linux

El portátil Linux juega un papel fundamental en el proceso de teleoperación, actuando como el intermediario entre la captura inicial de datos del sistema Perception Neuron, que se procesa en un portátil Windows, y la ejecución final de movimientos por el brazo robótico UR5e. Este sistema no solo recibe datos del portátil Windows sino que también calcula los ángulos necesarios y transmite comandos precisos al brazo robótico. A continuación, se presentan las especificaciones clave del portátil Linux que facilitan estas operaciones críticas:

- **Nombre del dispositivo:** Identificado como ‘eva-211x55‘, facilitando su administración y configuración en la red.

- **Procesador:** Equipado con un Intel(R) Core(TM) i5-9300H CPU a 2.40GHz, capaz de manejar múltiples tareas y cálculos intensivos sin degradar el rendimiento.

- **RAM instalada:** 8 GB, proporcionando suficiente capacidad para el procesamiento de datos y cálculos complejos en tiempo real.

- **Sistema operativo:** Ubuntu 20.04 LTS, una plataforma estable y confiable para aplicaciones de desarrollo y producción, especialmente en entornos de robótica y automatización.

- **Tipo de sistema:** Sistema operativo de 64 bits, lo que permite una gestión eficiente de la memoria y compatibilidad con software de última generación.

Este portátil es esencial para calcular y ajustar los ángulos de movimiento requeridos basados en los datos de movimiento humano capturados, asegurando que el UR5e ejecute movimientos que sean precisos y sincronizados. Este portátil junto con el de sistema windows procesan rápidamente y transmiten datos a través del framework ROS2.

3.1.3. Descripción del Router TP-Link TL-WA801ND

El router TP-Link TL-WA801ND se utiliza como un nodo central en el sistema de teleoperación, facilitando la interconexión de los diversos dispositivos dentro de una red local cerrada. A continuación se describe la función específica del router en esta configuración:

- **Velocidad Inalámbrica:** Capaz de alcanzar velocidades de hasta 300Mbps en la banda de 2.4GHz, lo que asegura una transmisión de datos eficiente y rápida entre los dispositivos conectados.

- **Antenas:** Equipado con dos antenas omnidireccionales desmontables de 5dBi, que mejoran la extensión de la cobertura y la potencia de la señal, permitiendo una comunicación inalámbrica más estable y un alcance más amplio.

- **Puerto Ethernet:** Incluye un puerto Ethernet 10/100Mbps que soporta

Figura 3.10: Especificaciones técnicas del portátil Linux utilizado en el sistema de teleoperación.

PoE pasivo, lo que facilita la instalación del dispositivo en lugares donde no se dispone de una toma de corriente, ya que permite la transmisión de energía eléctrica junto con los datos a través del cable Ethernet.

- **Seguridad:** Proporciona múltiples medidas de seguridad, incluyendo encriptación WPA/WPA2, control de acceso basado en direcciones MAC, y aislamiento de SSID para proteger la red de accesos no autorizados.

- **Configuración y Gestión:** Soporta configuración mediante la interfaz web y permite la administración de la red a través de la aplicación TP-Link Tether, facilitando el monitoreo y la gestión remota.

3.2. Arquitectura del Software - Sistema Windows

Basandonos en 3.1 hablaremos de la arquitectura del software que controla los dispositivos dentro del sistema de teleoperación.

Figura 3.11: Router TP-Link TL-WA801ND, interconexión de dispositivos.

3.2.1. El software Axis Neuron

El software Axis Neuron [17], desarrollado por Perception Neuron y exclusivamente compatible con el sistema operativo Windows, constituye el núcleo del sistema de captura de movimiento. Aunque representa una versión anterior y no es compatible con los modelos más recientes como PN Pro, PN3 o PN Studio, su funcionalidad principal sigue siendo indispensable. Este software se encarga de la captura, grabación y transmisión en tiempo real de los datos de movimiento recogidos por las unidades de medición inercial (IMU). En nuestro proyecto, este software desempeña un papel crucial ya que es desde esta plataforma en Windows desde donde nuestro software de ROS, ejecutado en el mismo sistema, lee los datos para después comunicarlos al sistema Linux, facilitando así la interacción y el control del brazo robótico UR5e a través de una integración eficaz entre los diferentes componentes y sistemas operativos.

Axis Neuron presenta una interfaz de usuario diseñada para optimizar la captura y transmisión de datos. Facilita la configuración de diversos parámetros a través de una barra de menú intuitiva, que incluye opciones para la gestión de archivos, ajustes de ventana, herramientas y ayuda.

En este software se proporcionan dos tipos de datos *Biovision Hierarchy* (BVH) y *Calculation Data*. El formato BVH es ampliamente utilizado para la

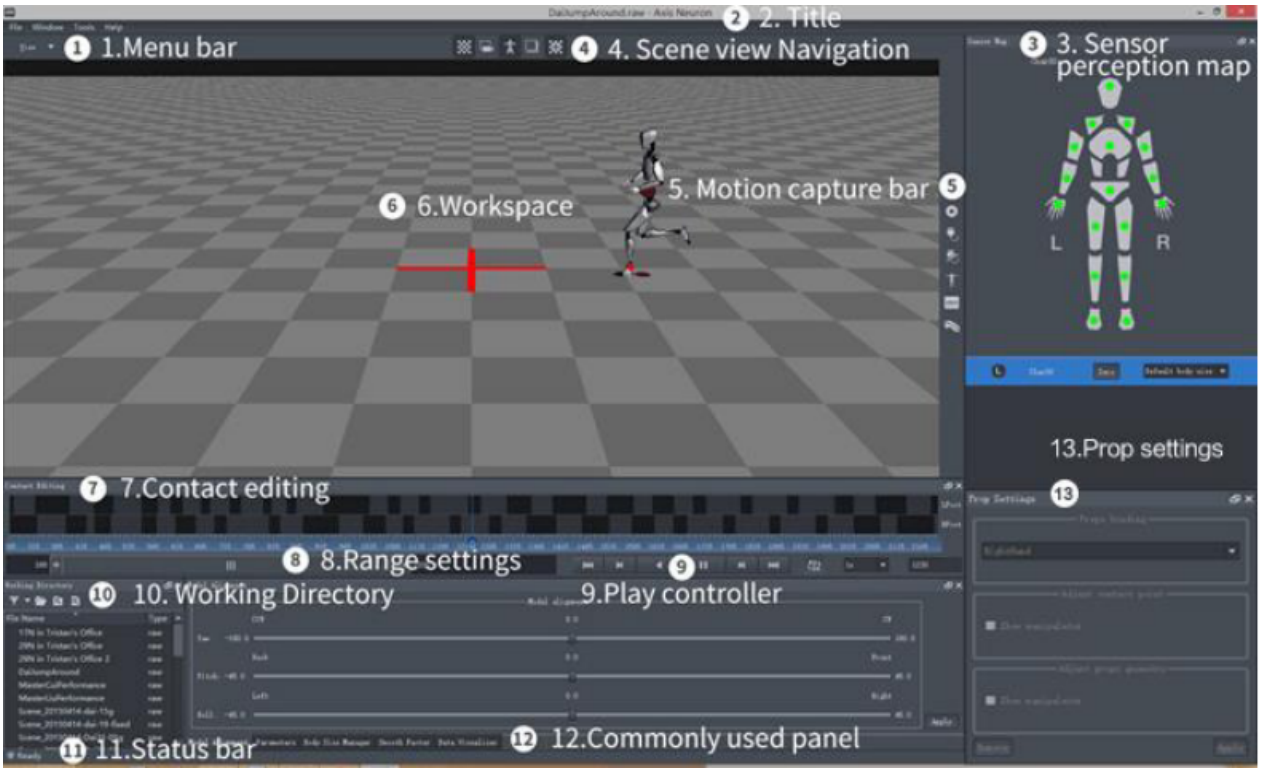

Figura 3.12: Interfaz de usuario de Axis Neuron mostrando la barra de menú y opciones de configuración.

animación de esqueletos virtuales, proporcionando datos de desplazamiento y rotación de cada articulación. Por otro lado, el formato de *Calculation Data* incluye información detallada sobre la posición, velocidad, cuaterniones (que describen la orientación), aceleración y datos giroscópicos de cada hueso, lo que es esencial para aplicaciones avanzadas de análisis de movimiento.

Además, el software Axis Neuron facilita la configuración del broadcasting de datos a través de la red. En este proyecto, es el software de ROS ejecutado en el sistema operativo Windows es el que recibe estos datos y posteriormente los envía al sistema operativo Linux. Dentro de esta configuración, se pueden especificar detalles como el protocolo (TCP o UDP), la dirección IP del host y el puerto para la transmisión de los datos. Esta funcionalidad es crucial para la integración efectiva de Axis Neuron en la arquitectura de hardware del proyecto, permitiendo la lectura y el acceso en tiempo real a los datos de movimiento por parte de los otros componentes del sistema.

El modo de conexión de los sensores hacia Axis Neuron puede operar a través de conexiones directas mediante USB o de manera inalámbrica utilizando Wi-Fi como puede ser en nuetro caso. Esto permite flexibilidad dependiendo de las restricciones del entorno y los requerimientos específicos de

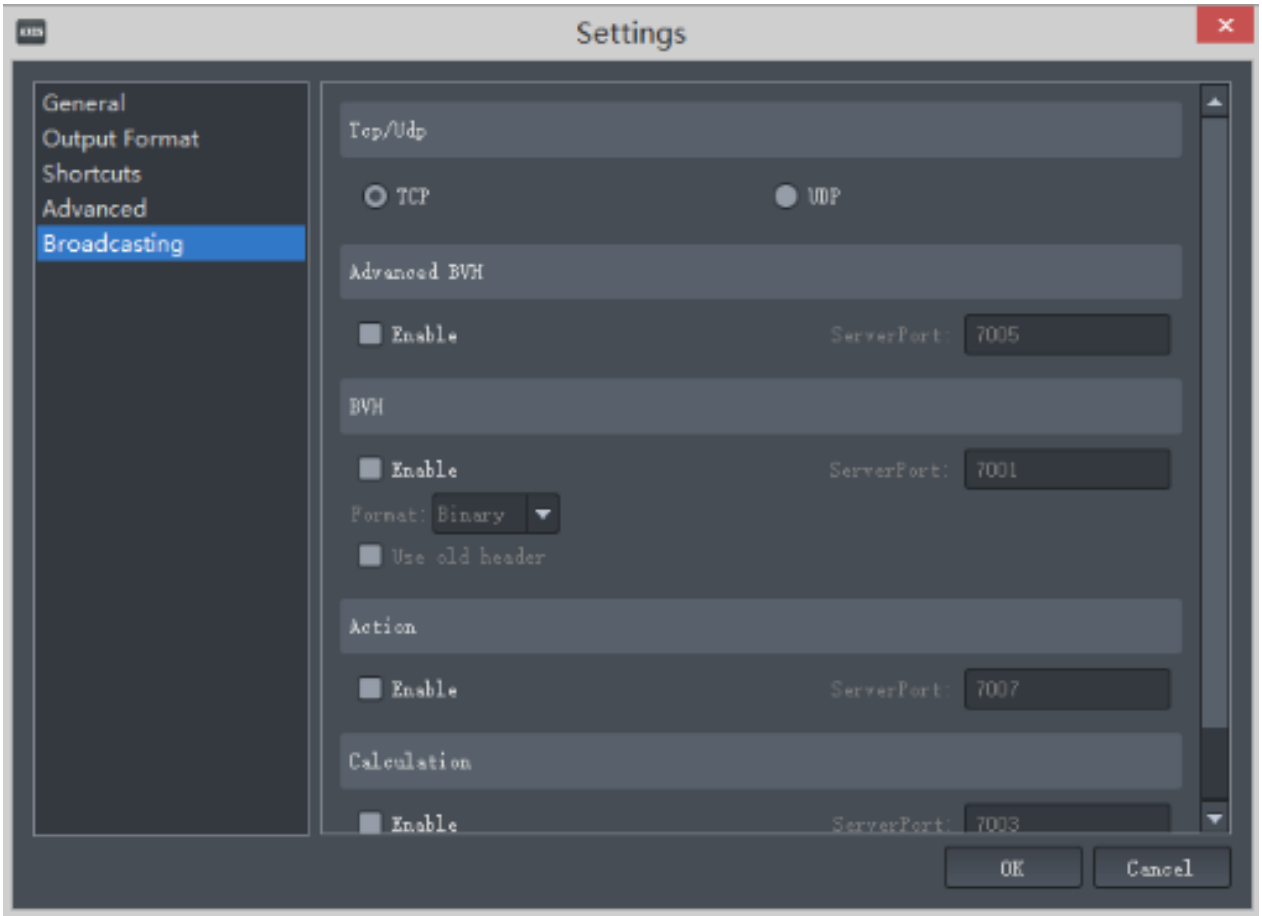

Figura 3.13: Configuración de Axis Neuron Broadcasting.

movilidad del usuario. Tambien proporciona un método intuitivo de calibración basado en movimientos corporales 3.18, simplificando notablemente el proceso de configuración del sistema.

Figura 3.14: Selección de Calibración de Postura en Software de Captura de Movimiento

El sistema Perception Neuron ofrece diversos modos de combinación de sensores que se adaptan a las necesidades específicas de captura de movimiento. Estos modos permiten una configuración personalizada según las partes del cuerpo que necesitan ser monitorizadas con precisión.

Modo Cuerpo Completo Utiliza sensores en ambos brazos, piernas, cadera y columna, ideal para capturas que requieren un análisis completo del movimiento corporal.

Modo Brazo Único En el proyecto actual, se ha seleccionado el modo Brazo Único, que se centra exclusivamente en capturar los movimientos de un brazo.

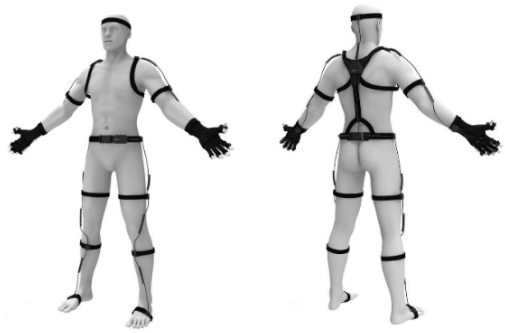

Figura 3.15: Modo Cuerpo Completo

Este modo es particularmente útil para aplicaciones donde se requiere precisión en los movimientos de una extremidad, como en nuestro caso, donde se analizan y replican estos movimientos para controlar un brazo robótico.

Single arm	3-Neurons Hub USB cable Dual pogo-pin cable/Body straps Gloves	

Figura 3.16: Modo 'Single Arm'

Modo Cuerpo Superior Captura movimientos del torso y ambos brazos, aplicable en escenarios donde los movimientos de la parte superior del cuerpo son críticos.

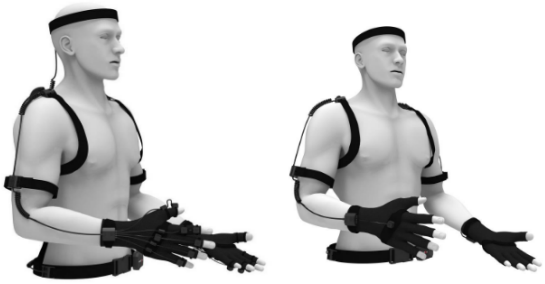

Figura 3.17: Modo Cuerpo Completo

Modo Cuerpo Inferior Diseñado para capturar movimientos de la cadera y las piernas, ideal para análisis de caminata o actividades deportivas.

El modo Brazo Único se ha seleccionado como la configuración más adecuada para el proyecto, dado que se ajusta precisamente a la recreación del

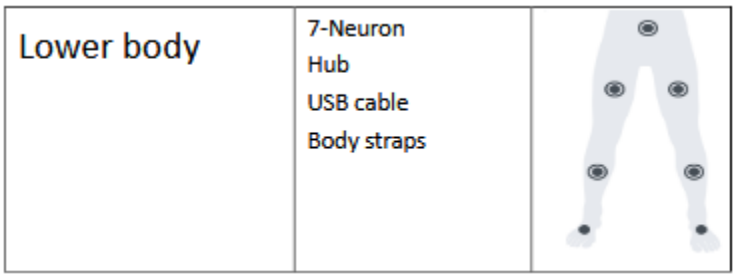

Figura 3.18: Modo 'Lower body'

movimiento del brazo humano que posteriormente será imitado por el brazo robótico UR5e. Esta configuración optimiza el uso de recursos al concentrarse exclusivamente en los movimientos del brazo.

En conclusión, el software Axis Neuron es un componente esencial en la arquitectura del proyecto, permitiendo una captura y transmisión eficiente de los datos de movimiento. Esta integración asegura que los datos recogidos sean procesados adecuadamente, permitiendo que el brazo robótico UR5e replique con exactitud los movimientos humanos. La elección del modo Brazo Único se alinea perfectamente con estas metas, concentrando los esfuerzos en capturar y analizar los movimientos específicos del brazo que son críticos para la operativa del sistema.

3.2.2. NeuronDataReader

NeuronDataReader [19], desarrollado por Noitom Technology Co., Ltd., es una biblioteca diseñada para facilitar la recepción y procesado de datos de esqueleto en tiempo real obtenidos a través de los sistemas de captura de movimiento de Perception Neuron. Esta librería nos permite integrar datos de movimiento en la aplicacion en tiempo real, proporcionando una interfaz robusta y flexible para el manejo de datos.

Formatos de Datos NeuronDataReader maneja dos formatos principales de datos que son los formatos habilitados en el AxisNeuron:

- **BVH (Biovision Hierarchy)**: Estos datos contienen información detallada sobre la posición y la orientación de cada uno de los huesos en un esqueleto animado. Cada frame en el formato BVH incluye datos de posición (X, Y, Z) y rotación (rotY, rotX, rotZ) para cada hueso, lo que permite una reconstrucción detallada del movimiento capturado.

- **Calculation Data**: Incluyen datos detallados sobre la posición, velocidad y orientación (cuaterniones) de cada hueso, proporcionando un

conjunto de datos exhaustivo para análisis avanzados.

Integración y Personalización NeuronDataReader puede recibir datos por los protocolos TCP/IP o UDP especificado en AxisNeuron tanto en formato BVH como de Calculation Data. Esto permite una configuración flexible del flujo de datos, adecuándose a las diversas necesidades de los proyectos.

Definiciones de Tipos de Datos La librería especifica varios tipos de datos y estructuras de los cuales los utilizados en el proyecto son:

- **Estado de Conexión de Socket**:
  - `Connected`: Indica que la conexión con el servidor está establecida.
  - `Connecting`: Estado transitorio mientras se establece la conexión.
  - `Disconnected`: La conexión se ha cerrado o no se ha podido establecer.
- **Encabezados de Datos de Flujo**:
  - `BvhDataHeader`: Incluye información sobre el flujo de datos BVH, como la presencia de desplazamientos y la secuencia de rotación de huesos.
  - `CalcDataHeader`: Utilizado para datos de Calculation Data, detallando la estructura de los datos transmitidos.
- **Órdenes de Rotación en BVH**: Define la secuencia de rotación aplicada a los datos de BVH.

Callbacks y Registro de Callbacks La biblioteca utiliza métodos de callback para entregar los datos recibidos. Estos callbacks deben ser registrados adecuadamente para recibir datos de esqueleto, datos de cálculo o cambios en el estado del socket:

- **Callback de Datos de Esqueleto** (`FrameDataReceived`): Recibe datos de esqueleto en formato BVH.

- **Callback de Datos de Cálculo** (`CalculationDataReceived`): Maneja los datos de cálculo que incluyen posición, velocidad, cuaterniones, entre otros.

- **Callback de Estado de Socket** (`SocketStatusChanged`): Informa sobre los cambios en el estado de la conexión de red.

Estos fragmentos de código son fundamentales para configurar adecuadamente la recepción de datos y asegurar que la aplicación cliente pueda procesar la información de movimiento en tiempo real de manera efectiva. La implementación correcta de estos callbacks permite una integración robusta y eficiente del sistema de captura de movimiento dentro del software desarrollado.

3.2.3. Biovision Hierarchy (BVH)

Los datos de BVH se consideran fundamentales para la aplicación actual. BVH (Biovision Hierarchy) es un formato de archivo que almacena información jerárquica sobre un esqueleto y sus movimientos. Cada fotograma transmitido por NeuronDataReader crea un esqueleto basado en los datos de BVH, reproduciendo los movimientos detallados en estos datos.

Formato de Datos de BVH Los datos de acción en formato BVH se generan por callback. La estructura incluye un encabezado de BVH y cada fotograma con los datos de movimiento de los huesos en formato flotante. A través de NeuronDataReader, estos datos se reciben desde Axis Neuron, conteniendo toda la información de movimiento de 59 huesos, o 6*59 datos. La secuencia de datos de los huesos en formato flotantes se muestra en el Anexo B: Secuencia de datos esqueleto en matriz.

Datos de BVH con y sin desplazamiento En el contexto del proyecto [19], los datos de BVH con desplazamiento incluyen seis valores flotantes por hueso, tres para desplazamientos (d_x, d_y, d_z) y tres para rotaciones (R_y, R_x, R_z). Los índices para calcular estos valores se especifican de la siguiente manera:

$$\begin{aligned}
\mathrm{d}_x &= \text{índice del hueso} \times 6 + 0 \\
\mathrm{d}_y &= \text{índice del hueso} \times 6 + 1 \\
\mathrm{d}_z &= \text{índice del hueso} \times 6 + 2 \\
\mathrm{R}_y &= \text{índice del hueso} \times 6 + 3 \\
\mathrm{R}_x &= \text{índice del hueso} \times 6 + 4 \\
\mathrm{R}_z &= \text{índice del hueso} \times 6 + 5
\end{aligned}$$

Para los datos de BVH sin desplazamiento, excepto el nodo raíz (cadera), que incluye datos de desplazamiento, los demás huesos únicamente contienen

datos de rotación.

Sistema de coordenadas de BVH Para representar la orientación de los huesos en el formato BVH (Biovision Hierarchy), es fundamental respetar el orden de los datos de rotación. El orden específico de rotación es: primero R_y, seguido por R_x y finalmente R_z. Esto es debido a que las rotaciones se aplican en un orden jerárquico, donde la primera rotación se realiza en torno al eje correspondiente del sistema de coordenadas local del hueso y las siguientes en función de los sistemas de coordenadas que se van generando.

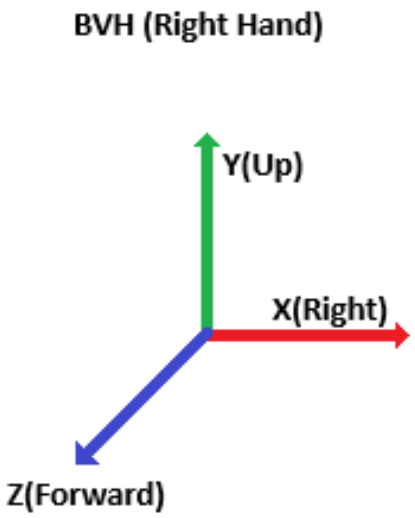

Figura 3.19: Sistema de coordenadas BVH utilizado en la transmisión de datos.

Jerarquía en el formato BVH El formato BVH establece una jerarquía clara donde la cadera actúa como nodo raíz y para nuestro caso la mano derecha como nodo terminal. Esta estructura jerárquica asigna a cada nodo una relación padre-hijo con el nodo anterior. Por ejemplo, los movimientos del hombro afectan directamente al brazo, y así sucesivamente hasta la mano. Este enfoque jerárquico asegura que los cambios en la posición o la orientación de un hueso superior afecten a todos los huesos subsecuentes en la cadena, permitiendo un mapeo preciso y coordinado de los movimientos del brazo derecho. Si nos fijamos en la siguiente imagen los índices que nos interesan son 12, 13, 14, 15.

3.3. Arquitectura del Software - Sistema Linux

Basandonos en 3.1 hablaremos de la arquitectura del software que controla los dispositivos dentro del sistema de teleoperado. Tambien hay que tener en cuenta que aunque se explique en esta sección ROS2, este framework tambien se uso en la sección anterior 3.2, en donde se explica el sistema windows.

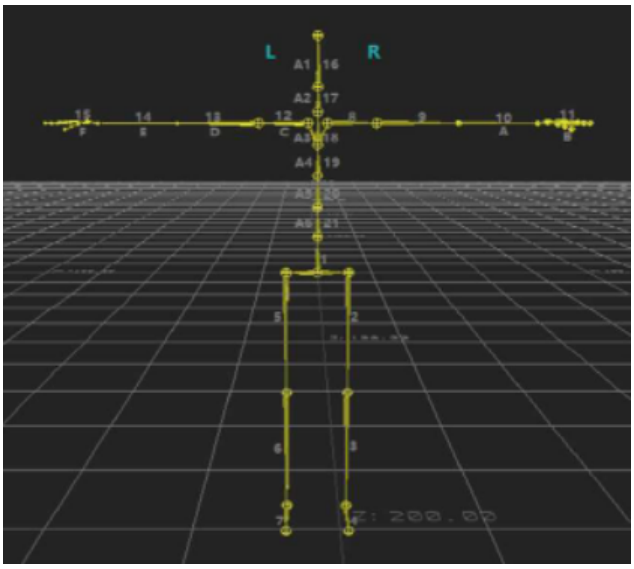

Figura 3.20: Estructura jerárquica de los huesos en formato BVH, en el Axis Neuron.

3.3.1. Sistema Operativo de Robots (ROS)

ROS es un framework ampliamente utilizado para el desarrollo eficiente de software robótico. No es un lenguaje de programación per se, sino un conjunto de herramientas y librerías escritas en varios lenguajes, incluidos Python y C++. Aunque no es un sistema operativo convencional, ROS proporciona servicios similares, como abstracción de hardware, control de dispositivos de bajo nivel, funcionalidades comunes, comunicación entre procesos y gestión de paquetes. Predominantemente se ejecuta en plataformas Unix como Ubuntu o Mac OS X, aunque también es compatible con Windows, donde su estabilidad aún está en desarrollo. Su estructura flexible permite la interconexión entre robots y computadoras de diferentes sistemas y arquitecturas.

Figura 3.21: ROS (Robot Operating System), sus versiones 1 y 2

La elección de ROS2 se justifica por ser una evolución de su versión predecesora, manteniendo los principios fundamentales pero extendiendo sus capacidades con funcionalidades avanzadas[20].

Tabla 3.1: Comparativa entre ROS y ROS2.

ROS	ROS2
Comunicación centralizada a través de un Master, con potenciales puntos de fallo únicos.	Comunicación descentralizada utilizando DDS, permitiendo un descubrimiento distribuido de nodos.
Sin soporte multi-robot eficaz debido a un único Master.	Soporte multi-robot mejorado a través de la comunicación descentralizada de DDS.
Limitado soporte multiplataforma, mayormente en Ubuntu.	Soporte amplio multiplataforma, incluyendo Ubuntu, Windows y Mac OS X.

Conceptos de ROS

En ROS, los elementos denominados *nodos* ejecutan tareas específicas dentro de un sistema robótico. Estos nodos pueden comunicarse entre sí ya sea dentro del mismo proceso, entre procesos distintos o incluso entre diferentes máquinas. Los nodos tienen la capacidad de publicar en *topics* o suscribirse a ellos para intercambiar datos, y pueden actuar tanto como clientes de servicios, solicitando cálculos a otros nodos, como servidores de servicios, proporcionando funcionalidades. Generalmente, los nodos combinan funciones de publicadores, suscriptores, servidores y clientes de servicios, así como de acciones.

La variable `ROS_DOMAIN_ID` es crucial para definir el ámbito en el que los nodos pueden descubrirse y comunicarse. Esta configuración permite que nodos dentro del mismo dominio establezcan comunicaciones utilizando el servicio de descubrimiento que proporciona DDS. Al asignar el mismo `ROS_DOMAIN_ID` a varios nodos, se asegura que formen parte del mismo entorno de comunicación, facilitando así el intercambio eficiente de mensajes y la coordinación de acciones dentro de la misma red.

Los nodos en ROS2 pueden comunicarse mediante *topics*, *services* o *actions*, utilizando un lenguaje de descripción de interfaz simplificado (IDL) para definir las estructuras de datos intercambiadas. Cada tipo de comunicación tiene una extensión de archivo asociada: `.msg` para topics, `.srv` para services y `.action` para actions.

3.4. Universal Robots RTDE C++ Interface

La librería URcontrol [21], utilizada en el sistema Linux, es una herramienta de Python diseñada para facilitar la comunicación y control de robots UR (Universal Robots) mediante la interfaz RTDE (Real-Time Data Exchange). Esta librería ofrece varias interfaces que permiten tanto el control del robot como la recepción de datos y la manipulación de entradas y salidas digitale-

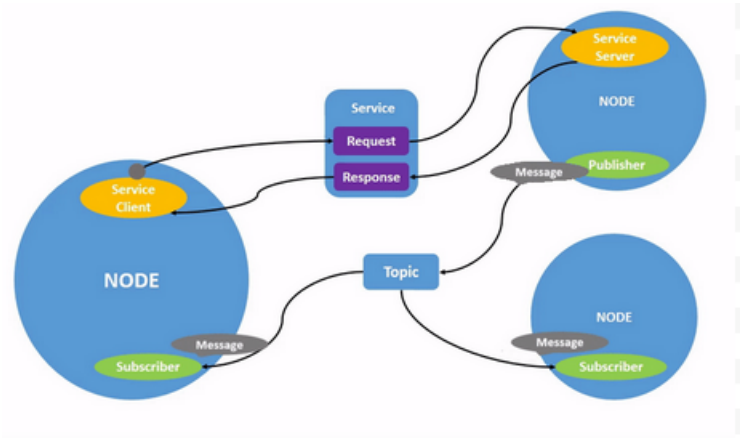

Figura 3.22: Configuración de la Red ROS

s/analógicas.

La librería URcontrol proporciona las siguientes interfaces:

- **RTDE Control Interface:** Utilizada principalmente para mover el robot y ejecutar funciones utilitarias. Requiere que un script de control esté corriendo en el robot, el cual es cargado automáticamente.

- **RTDE Receive Interface:** Permite recibir datos del robot, facilitando la monitorización de su estado y comportamiento.

- **RTDE IO Interface:** Usada para configurar entradas/salidas digitales y analógicas y ajustar el deslizador de velocidad del robot. Esta interfaz permite el control de IOs sin interferir con los movimientos del robot.

Instalación y Compatibilidad

URcontrol es compatible con varias versiones de Ubuntu (16.04, 18.04, 20.04), macOS (10.14 Mojave) y Windows 10 Pro x64. Para instalar la librería en un sistema Ubuntu, se puede utilizar pip con el siguiente comando:

```
pip install --user ur_rtde
```

Para una instalación más detallada y opciones avanzadas, se puede consultar la documentación oficial [**?**].

Ejemplos de Uso

Ejemplo de Control Asíncrono

Este ejemplo demuestra cómo realizar movimientos asíncronos utilizando las funciones 'moveJ' y 'moveL', y detener estos movimientos antes de alcanzar sus objetivos:

```
from rtde_control import RTDEControlInterface as RTDEControl
from rtde_receive import RTDEReceiveInterface as RTDEReceive

rtde_c = RTDEControl("127.0.0.1")
rtde_r = RTDEReceive("127.0.0.1")

# Movimientos as ncronos
rtde_c.moveJ([0, -1.57, 0, -1.57, 0, 0], async=True)
rtde_c.stopJ()
rtde_c.moveL([0.3, 0.3, 0.2, 0, 3.14, 0], async=True)
rtde_c.stopL()
```

Funciones de Control y Recepción

- El comando `moveL` es útil para aplicaciones que requieren movimientos rectilíneos precisos.

- El comando `moveJ` es adecuado para movimientos donde se requiere que el robot siga una trayectoria a velocidad constante en cada una de las articulaciones durante su movimiento. Esto no asegura que la trayectoria del extremo del robot sea lineal.

- **getActualTCPPose():** Retorna la pose actual del punto de control de la herramienta (TCP).

Real-Time Capabilities

URcontrol permite definir prioridades en tiempo real al instanciar las interfaces 'RTDEControl' y 'RTDEReceive'.

El comando `moveJ` utiliza un vector de posiciones articulares `joint` para mover el brazo robótico a una configuración específica. Cada elemento del vector `joint` corresponde a una articulación del brazo robótico. A continuación se detalla a qué parte del brazo robótico pertenece cada dato:

- **joint[0]**: Ángulo de la base

  - **Función**: Rota la base del robot.

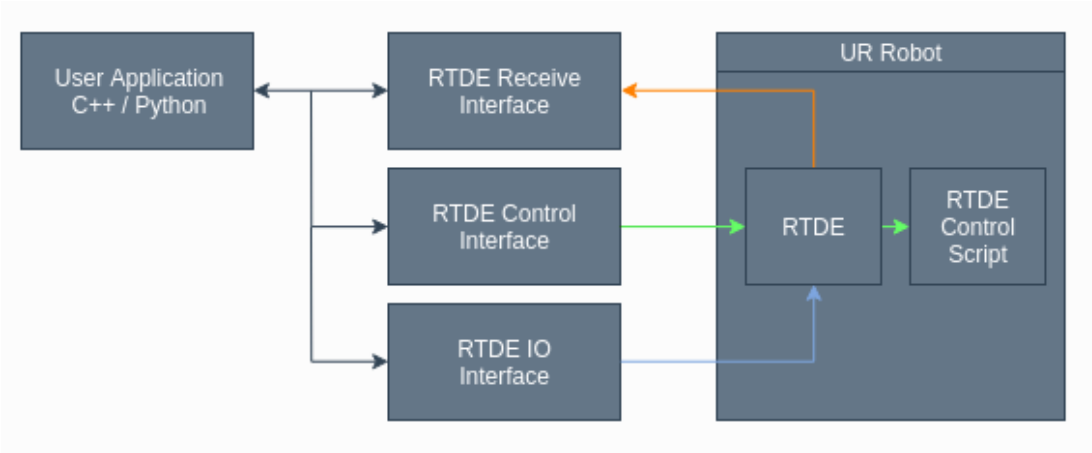

Figura 3.23: Diagrama de la arquitectura de la librería URcontrol.

- **joint[1]**: Ángulo del hombro
  - **Función**: Mueve el primer brazo del robot, permitiendo que suba o baje.
- **joint[2]**: Ángulo del codo
  - **Función**: Controla el movimiento del segundo segmento del brazo, permitiendo que se flexione o extienda.
- **joint[3]**: Ángulo de la muñeca 1
  - **Función**: Mueve el tercer segmento del brazo, proporcionando un giro adicional.
- **joint[4]**: Ángulo de la muñeca 2
  - **Función**: Controla otro grado de rotación en la muñeca.
- **joint[5]**: Ángulo de la muñeca 3
  - **Función**: Proporciona la última rotación en la muñeca, completando la capacidad de orientación del extremo del brazo.

Capítulo 4

Análisis Cinemático

La manipulación robótica de objetos requiere precisión en el movimiento espacial, lo que implica un profundo conocimiento de la posición y orientación del objeto respecto a la base del robot. Este desafío se aborda mediante herramientas matemáticas avanzadas que facilitan la especificación exacta de la posición y orientación en el espacio. Este apartado detalla la técnica matemática empleada para imitar los movimientos humanos al brazo robótico UR5e utilizando datos BVH (Biovision Hierarchy) 6.2.

Entre los métodos explorados, las matrices de transformación homogénea [10] y principios trigonométricos destacan por su capacidad para ofrecer una representación conjunta de la posición y orientación. Estas matrices no solo son pertinentes en robótica sino también en otros campos como la creación de gráficos por computadora, demostrando su versatilidad y eficacia.

El cálculo de las partes del brazo robótico se realiza mediante transformaciones y rotaciones en el espacio tridimensional. La base se determina aplicando transformaciones sucesivas para obtener el ángulo de rotación necesario. El hombro se evalúa considerando las posiciones y orientaciones relativas entre el brazo y el antebrazo. El ángulo del codo se calcula a partir de los vectores de posición relativos entre el antebrazo y la mano, lo que permite deducir el ángulo de flexión. La orientación de la muñeca 1 se ajusta considerando la relación angular entre el antebrazo y la mano. Finalmente, la orientación de la muñeca 2 se obtiene evaluando las rotaciones relativas entre el brazo, el antebrazo y la mano, determinando así la posición final de la muñeca.

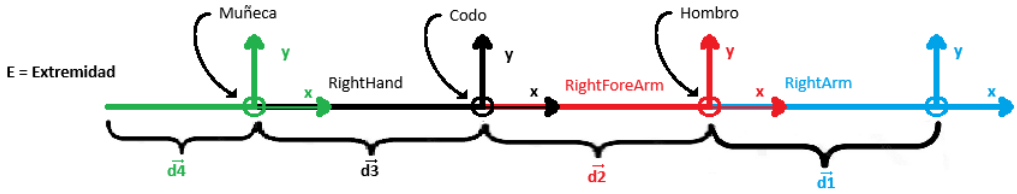

Figura 4.1: Cada color representa un hueso y su sistema de referencia asociado, datos del BVH (posición y rotación).

4.1. Base y Hombro

Coordenadas y Matrices Homogéneas

La representación [10] de la localización de sólidos en espacios de dimensión n se efectúa a través de coordenadas en un espacio $(n+1)$-dimensional. Esto implica que un punto (x, y, z) en un espacio tridimensional se representa como (wx, wy, wz, w) en coordenadas homogéneas, donde w es un factor de escala arbitrario. Un vector en el sistema de referencia xyz se expresa en **coordenadas homogéneas** mediante el vector columna:

$$p = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} aw \\ bw \\ cw \\ w \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix}$$

donde a, b, c son componentes del vector respecto a los ejes Ox, Oy, y Oz. Este método facilita transformaciones tales como traslación y rotación, permitiendo la manipulación eficiente de las coordenadas en proyectos de ingeniería y computación gráfica.

Composición de la "Matriz de Transformación Homogenea" para calcular los angulos de la base de un brazo tres ejes

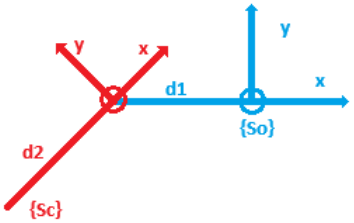

Figura 4.2: Sistemas de referencia del RightArm (índice 14) a RightForeArm (índice 15).

Una Matriz de transformación homogénea T es una matriz de dimensión 4 $\times$ 4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro. Esta matriz se compone de:

$$T = \begin{bmatrix} R_{3\times3} & P_{3\times1} \\ f_{1\times3} & w_{1\times1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix} \tag{4.1}$$

T permite el cálculo de los ángulos necesarios para configurar correctamente la base del brazo robótico (Arm $\rightarrow$ ForeArm). Al obtener la matriz de transformación desde el brazo hasta la clavícula 4.5 en el ser humano y replicar este proceso en el brazo robótico, se logra una correspondencia directa que permite igualar ambas matrices. Así, se garantiza que ambos sistemas —el humano y el robótico— operen bajo los mismos parámetros espaciales, facilitando la comparación y ajuste preciso de los ángulos necesarios para el funcionamiento del robot UR5e.

En nuestro proyecto, los datos que recibimos del RightArm (siendo 14 el indice del BVH) y teniendo en cuenta que los valores de p_{14y} y p_{14z} son cero:

$$\underbrace{p_{14x}}_{d_1}, p_{14y}, p_{14z}, \theta_{14y}, \theta_{14x}, \theta_{14z}$$

Figura 4.3: Datos que vienen del BVH de RightArm

Y los datos que recibimos del RightForeArm (siendo 15 el indice del BVH) y como en el caso anterior p_{15y} y p_{15z} son cero:

$$\underbrace{p_{15x}}_{d_2}, p_{15y}, p_{15z}, \theta_{15y}, \theta_{15x}, \theta_{15z}$$

Figura 4.4: Datos que vienen del BVH de RightForeArm

Utilizamos la idea de que una matriz de transformación homogénea puede ser descompuesta en múltiples transformaciones simples, como rotaciones y translaciones, que se aplican en secuencia. Cada translación o rotación se representa como una matriz homogénea, y la transformación completa se obtiene mediante la multiplicación de estas matrices en el orden correcto. El conjunto de estas matrices simples describen el comportamiento del sistema

de referencia de cada hueso e igualando el sistema al del robot podremos sacar los angulos que necesitamos para describir el movimiento del brazo.

$$^{o}T_c = \text{trasl}(\vec{d_1}) \cdot \text{R}_y(\theta_{14y}) \cdot \text{R}_x(\theta_{14x}) \cdot \text{R}_z(\theta_{14z}) \cdot \text{trasl}(\vec{d_2})$$

Esta matriz $^{o}T_c$ se compone de la siguiente manera:

- **Traslación** ($\text{trasl}(\vec{d_1})$):

$$\begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Esta operación desplaza el punto original unicamente a lo largo de los ejes, en este caso sólo a lo largo del eje x.

- **Rotación en y,x,z** ($\text{R}_y(\theta_{14y}) \cdot \text{R}_x(\theta_{14x}) \cdot \text{R}_z(\theta_{14z})$):

$$\begin{bmatrix} \cos(\theta_{14y}) & 0 & \sin(\theta_{14y}) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_{14y}) & 0 & \cos(\theta_{14y}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_{14x}) & -\sin(\theta_{14x}) & 0 \\ 0 & \sin(\theta_{14x}) & \cos(\theta_{14x}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta_{14z}) & -\sin(\theta_{14z}) & 0 & 0 \\ \sin(\theta_{14z}) & \cos(\theta_{14z}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Esta operación permiten modificar la orientación de cada eje en un espacio tridimensional.

- **Traslación** ($\text{trasl}(\vec{d_2})$):

$$\begin{bmatrix} 1 & 0 & 0 & d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Esta operación desplaza el punto extremo del arm unicamente a lo largo del eje x.

El resultado final sería el siguiente:

$$ {}^{o}T_c = \begin{bmatrix} * & * & * & -d_1 - d_2(\cos(\theta_{14y})\cos(\theta_{14z}) + \sin(\theta_{14y})\sin(\theta_{14x})\sin(\theta_{14z})) \\ * & * & * & -d_2\cos(\theta_{14x})\sin(\theta_{14z}) \\ * & * & * & -d_2(-\sin(\theta_{14y})\cos(\theta_{14z}) + \cos(\theta_{14y})\sin(\theta_{14x})\sin(\theta_{14z})) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2} $$

La parte de la matriz indicada con asteriscos al final del documento no es relevante para el análisis específico. Esto se debe a que solo se requieren los componentes de translación que se ven afectados directamente por la orientación del sistema de referencia de cada hueso.

Composición de la "Matriz de Transformación Homogénea" para Calcular los Ángulos de la Base de un Brazo Robótico de dos Ejes

La base y el hombro del brazo robótico realizan movimientos rotacionales alrededor de dos ejes principales, el eje y y el eje z, representados por los ángulos θ_{R1} y θ_{R2} respectivamente. Estas rotaciones son esenciales para la configuración precisa de los ángulos en la base del brazo robótico y son los que igualaremos a la matriz de tranformación homogenea del brazo humano 4.2.

En base a la información detallada en el apartado anterior 4.1, la matriz de transformación homogenea para el brazo robótico sería:

$$ {}^{o}T_c = \text{trasl}(d_1) \cdot \text{R}_y(\theta_{R1}) \cdot \text{R}_z(\theta_{R2}) \cdot \text{trasl}(d_2) $$

Donde ${}^{o}T_c$ se compone de la siguiente manera:

- **Traslación** ($\text{trasl}(d_1)$):

$$ \begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

  Esta operación desplaza el punto original unicamente a lo largo de los ejes.

- **Rotación en y, z** ($\mathrm{R}_y(\theta_{R1}) \cdot \mathrm{R}_z(\theta_{R2})$):

$$\begin{bmatrix} \cos(\theta_{R1}) & 0 & \sin(\theta_{R1}) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_{R1}) & 0 & \cos(\theta_{R1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta_{R2}) & -\sin(\theta_{R2}) & 0 & 0 \\ \sin(\theta_{R2}) & \cos(\theta_{R2}) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Esta operación permiten modificar la orientación de cada eje en un espacio tridimensional. rotándolos alrede

- **Traslación** ($\mathrm{trasl}(d_2)$):

$$\begin{bmatrix} 1 & 0 & 0 & d_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Esta operación desplaza el punto original unicamente a lo largo de los ejes.

El resultado final sería el siguiente:

$$^{o}T_c = \begin{bmatrix} * & * & * & -d_1 - d_2 \cdot \cos(\theta_{R1}) \cdot \cos(\theta_{R2}) \\ * & * & * & -d_2 \cdot \sin(\theta_{R2}) \\ * & * & * & -d_2 \cdot \sin(\theta_{R1}) \cdot \cos(\theta_{R2}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Y con ambas matrices en el mismo sistema solo nos queda igualar 4.2 y 4.3 para poder sacar los ángulos correctos.

$$\left.\begin{array}{c} d_1 + d_2(\cos(\theta_{14y}) \cdot \cos(\theta_{14z}) + \sin(\theta_{14y}) \cdot \sin(\theta_{14x}) \cdot \sin(\theta_{14z})) = d_1 + d_2 \cdot \cos(\theta_{R1}) \cdot \cos(\theta_{R2}) \\ d_2 \cdot \cos(\theta_{14x}) \cdot \sin(\theta_{14z}) = d_2 \cdot \sin(\theta_{R2}) \\ d_2(-\sin(\theta_{14y}) \cdot \cos(\theta_{14z}) + \cos(\theta_{14y}) \cdot \sin(\theta_{14x}) \cdot \sin(\theta_{14z})) = d_2 \cdot \sin(\theta_{R1}) \cdot \cos(\theta_{R2}) \end{array}\right\}$$

$$\left.\begin{array}{c} \cos(\theta_{14y}) \cdot \cos(\theta_{14z}) + \sin(\theta_{14y}) \cdot \sin(\theta_{14x}) \cdot \sin(\theta_{14z}) = \cos(\theta_{R1}) \cdot \cos(\theta_{R2}) \\ \cos(\theta_{14x}) \cdot \sin(\theta_{14z}) = \sin(\theta_{R2}) \\ \sin(\theta_{14y}) \cdot \cos(\theta_{14z}) - \cos(\theta_{14y}) \cdot \sin(\theta_{14x}) \cdot \sin(\theta_{14z})) = \sin(\theta_{R1}) \cdot \cos(\theta_{R2}) \end{array}\right\}$$

Despejando se obtienen los dos primeros ángulos del robot UR5e:

$$\theta_{R2} = \arcsin(\cos(\theta_{14x})\sin(\theta_{14z}))$$
$$\theta_{R1} = \arcsin\left(\frac{\sin(\theta_{14y})\cos(\theta_{14z}) - \cos(\theta_{14y})\sin(\theta_{14x})\sin(\theta_{14z})}{\cos(\theta_{R2})}\right) \tag{4.4}$$

4.2. Codo

El desafío de hacer que el brazo robótico UR5e imite los movimientos del brazo humano radica en las limitaciones mecánicas y de grados de libertad del robot. El brazo humano posee una complejidad biomecánica considerable, con múltiples ejes de rotación en cada articulación, permitiendo una amplia gama de movimientos precisos y coordinados. En contraste, el UR5e cuenta con dos motores para imitar los movimientos del hombro y uno para la flexión del codo.

Para el cálculo del ángulo del codo en el brazo robótico UR5e se realiza mediante el uso de matrices de transformación homogénea como en el apartado anterior.

La flexión del codo en el UR5e es controlada por un único motor, lo que significa que el movimiento se restringe a un solo plano de rotación. En el brazo humano, la flexión del codo es un movimiento coordinado que también puede involucrar ajustes en la pronación y supinación del antebrazo. La limitación a un solo motor implica que el robot no puede replicar estos ajustes adicionales, resultando en una imitación menos precisa y natural del movimiento del codo humano.

Composión de las "Matriz de Transformación Homogenea" para representar todos los huesos del brazo derecho.

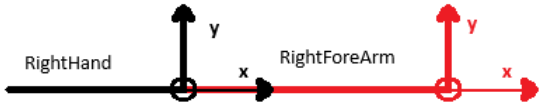

Figura 4.5: Sistemas de referencia del RightArm a RightForeArm

Primero, se define una matriz de transformación homogénea que representa la orientación y posición inicial del sistema de referencia de la espina dorsal (Sp).

$$^{o}T_{Sp} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Apartir de aqui se definen matrices de transformación para cada segmento del brazo con respecto al anterior: la base de la clavícula (Sh), el hombro (A), el codo (FA) y la muñeca (H). Cada matriz de transformación incluye traslaciones y rotaciones que alinean un segmento con el siguiente.

Base de la Clavícula (Sh)

El índice del BVH de donde sacar los datos como vemos en Anexo 8.2 es el 13.

$$\underbrace{p_{13x}}_{d_0}, p_{13y}, p_{13z}, \theta_{13y}, \theta_{13x}, \theta_{13z}$$

Se calcula la posición de la base de la clavícula (Sh) respecto al punto de enganche de la espina dorsal (Sp). Esto se realiza mediante la combinación de una traslación y rotaciones sucesivas:

$$^{Sp}T_{Sh} = \text{trasl}(\vec{d_0}) \cdot \text{R}_y(\theta_{13y}) \cdot \text{R}_x(\theta_{13x}) \cdot \text{R}_z(\theta_{13z})$$

Hombro (A)

El índice del BVH de donde sacar los datos como vemos en Anexo 8.2 es el 14.

$$\underbrace{p_{14x}}_{d_1}, p_{14y}, p_{14z}, \theta_{14y}, \theta_{14x}, \theta_{14z}$$

Se calcula la posición del hombro respecto a la clavícula. Este cálculo incluye una traslación a través de la clavícula y rotaciones para alinear el hueso del brazo con el eje x del nuevo sistema de referencia:

$$^{Sh}T_{A} = \text{trasl}(\vec{d_1}) \cdot \text{R}_y(\theta_{14y}) \cdot \text{R}_x(\theta_{14x}) \cdot \text{R}_z(\theta_{14z})$$

Codo (FA)

El índice del BVH de donde sacar los datos como vemos en Anexo 8.2 es el 15.

$$\underbrace{p_{15x}}_{d_2}, p_{15y}, p_{15z}, \theta_{15y}, \theta_{15x}, \theta_{15z}$$

Se calcula la posición del codo respecto al hombro. Este proceso incluye una traslación a través del brazo para llegar al codo y rotaciones para alinear el antebrazo con el eje x del sistema de referencia:

$$^{A}T_{FA} = \text{trasl}(\vec{d_2}) \cdot \text{R}_y(\theta_{15y}) \cdot \text{R}_x(\theta_{15x}) \cdot \text{R}_z(\theta_{15z})$$

Muñeca (H)

El índice del BVH de donde sacar los datos como vemos en Anexo 8.2 es el 16.

$$\underbrace{p_{16x}}_{d_3}, p_{16y}, p_{16z}, \theta_{16y}, \theta_{16x}, \theta_{16z}$$

Se calcula la posición de la muñeca respecto al codo mediante una traslación a través del antebrazo y rotaciones para alinear el hueso de la mano con el eje x del nuevo sistema de referencia:

$$^{FA}T_{H} = \text{trasl}(\vec{d_3}) \cdot \text{R}_y(\theta_{16y}) \cdot \text{R}_x(\theta_{16x}) \cdot \text{R}_z(\theta_{16z})$$

Utilizando las matrices de transformación, se calculan las posiciones de todas las partes del brazo respecto al origen. Esto se realiza multiplicando las matrices de transformación de cada segmento, comenzando desde la espina dorsal hasta llegar a la muñeca:

$$\begin{aligned} ^{0}T_{Sh} &= ^{0}T_{Sp} \cdot ^{Sp}T_{Sh} \\ ^{0}T_{A} &= ^{0}T_{Sh} \cdot ^{Sh}T_{A} \\ ^{0}T_{FA} &= ^{0}T_{A} \cdot ^{A}T_{FA} \\ ^{0}T_{H} &= ^{0}T_{FA} \cdot ^{FA}T_{H} \end{aligned}$$

Para determinar el ángulo del codo, se utilizan los vectores de posición relativos:$^{FA}P_A$ es el vector desde el codo hasta el hombro y $^{FA}P_H$ es el vector desde el codo hasta la muñeca.

El ángulo del codo (θ_{codo}) se calcula usando el producto escalar de estos vectores:

$$\cos(\theta_{\text{codo}}) = \frac{{}^{FA}p_A \cdot {}^{FA}p_H}{\|{}^{FA}p_A\|\|{}^{FA}p_H\|}$$

Luego, se utiliza la función arco coseno para obtener el ángulo:

$$\theta_{\text{codo}} = \arccos\left(\frac{{}^{FA}p_A \cdot {}^{FA}p_H}{\|{}^{FA}p_A\|\|{}^{FA}p_H\|}\right)$$

Finalmente, dado que en el robot UR5e la posición del codo estirado se considera como 0 grados, se ajusta el ángulo restándolo de 180 grados:

$$\theta_{\text{codo}} = 180^\circ - \theta_{\text{codo}}$$

4.3. Muñeca

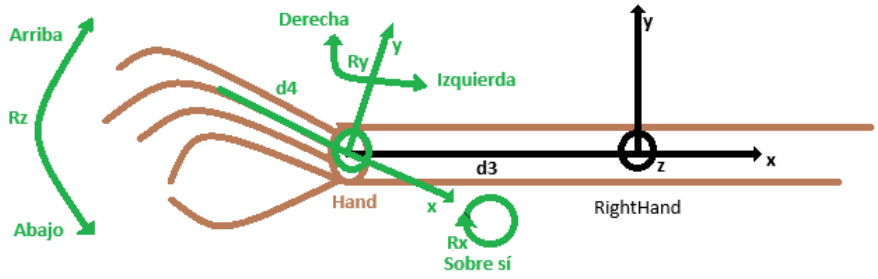

Figura 4.6: Sistemas de referencia del RightForeArm a RightHand

Ángulo de la Muñeca 1

Para los datos de este apartado el indice es el 16, Anexo 8.2.

$$\underbrace{p_{16x}}_{d_3}, p_{16y}, p_{16z}, \theta_{16y}, \theta_{16x}, \theta_{16z}$$

Para calcular el ángulo de la Muñeca 1, se utiliza la misma fórmula empleada para determinar el ángulo del hombro, ya que ambos movimientos se ven afectados por el mismo eje. La transformación homogénea y las rotaciones en los ejes y y z se aplican de manera similar, permitiendo determinar la orientación de la muñeca respecto al antebrazo. Así, el ángulo se calcula a partir de las componentes rotacionales obtenidas del BVH, siguiendo la misma metodología que en el cálculo del hombro 4.4.

$$T_H = \text{trasl}(\vec{d_3}) \cdot \text{R}_y(\theta_{16y}) \cdot \text{R}_x(\theta_{16x}) \cdot \text{R}_z(\theta_{16z})$$

Donde T_H representa la transformación de la muñeca con respecto al antebrazo.

Ángulo de la Muñeca 2

Los datos correspondientes para este apartado son 14, 15, 16 correspondiente al Anexo 8.2.

$$\begin{aligned} RightArm &= \theta_{14x} \\ RightForeArm &= \theta_{15x} \\ RightHand &= \theta_{16x} \end{aligned}$$

El ángulo de la Muñeca 2 describe la rotación en el eje x. Este ángulo se calcula como la diferencia entre las rotaciones de la mano derecha, el antebrazo derecho y el brazo derecho, dado que son rotaciones que se afectan consecutivamente según el formato BVH. La fórmula utilizada para este cálculo es:

$$\text{Ángulo Muñeca 2} = \theta_{14x} - \theta_{15x} - \theta_{16x}$$

Esta diferencia de ángulos se debe a que las rotaciones en el formato BVH se aplican en secuencia, afectando consecutivamente la orientación de cada segmento del brazo. El resultado es un ángulo que refleja la rotación acumulada a lo largo del brazo, desde el brazo hasta la mano.

Capítulo 5

Desarrollo

En este apartado se examina el proceso de desarrollo, implementación, integración y comunicación involucrado en el control robótico. El flujo de datos y control comienza con la captura de movimiento a través de **Perception Neuron**, seguido por su procesamiento inicial en un sistema operativo Windows. En este contexto, se destaca el papel fundamental del entorno ROS 2[22], el cual opera tanto en sistemas Windows como Linux, facilitando la comunicación y el manejo de datos. Finalmente, en un sistema Linux, se realiza el control del brazo robótico UR5e utilizando el controlador ur_control.

El proceso descrito a continuación hace referencia a este sistema (véase la Figura 3.1).

5.1. Instalación de ROS 2 en Windows (Versión Binaria)

Este apartado describe la instalación de ROS 2 Humble[23] en Windows utilizando un paquete binario precompilado.

Requisitos del Sistema

Windows 10 es el único sistema operativo soportado.

Instalación de Prerrequisitos

Instalar Chocolatey

Chocolatey es un gestor de paquetes para Windows. Las instrucciones de instalación se encuentran en su página oficial[24].

Instalar Python

Abre un Símbolo del sistema y ejecuta el siguiente comando:

```
choco install -y python --version 3.8.3
```

Python se instalará en `C:\Python38`.

Instalar Visual C++ Redistributables

Ejecuta el siguiente comando en un Símbolo del sistema:

```
choco install -y vcredist2013 vcredist140
```

Instalar OpenSSL

Descarga el instalador de OpenSSL v1.1.1n para Win64 y ejecútalo con los parámetros por defecto. Establece la variable de entorno:

```
setx /m OPENSSL_CONF "C:\Program Files\OpenSSL-Win64\bin\openssl.cfg"
```

Instalar Visual Studio 2019

Visual Studio 2019 se instala seleccionando el flujo de trabajo "Desarrollo de escritorio con C++". No se deben instalar las herramientas C++ CMake.

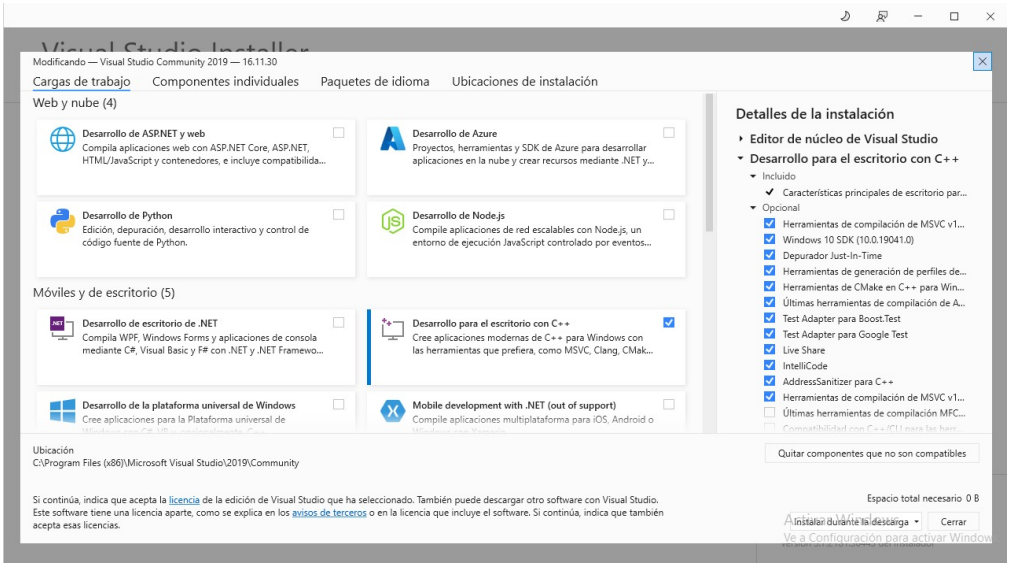

Figura 5.1: Instalación de Visual Studio 2019, propiedades a marcar.

Instalar OpenCV

OpenCV 3.4.6 se descarga precompilado y se establece la variable de entorno:

```
setx /m OpenCV_DIR C:\opencv
```

Instalar Dependencias

Para instalar CMake y las dependencias adicionales con Chocolatey:

```
choco install -y cmake
```

Se debe agregar `C:\Program Files\CMake\bin` a la variable `PATH`.

Los paquetes se descargan desde el repositorio de GitHub:

- asio.1.12.1.nupkg
- bullet.3.17.nupkg
- cunit.2.1.3.nupkg
- eigen-3.3.4.nupkg
- tinyxml-usestl.2.6.2.nupkg
- tinyxml2.6.0.0.nupkg

Una vez descargados, se ejecuta el siguiente comando en una shell administrativa, reemplazando `<RUTA\A\DESCARGAS>` con la ruta de los paquetes:

```
choco install -y -s <RUTA\A\DESCARGAS> asio cunit eigen tinyxml-usestl
    ↪ tinyxml2 bullet
```

Se actualizan pip y setuptools:

```
python -m pip install -U pip setuptools==59.6.0
```

Luego, se instalan las dependencias adicionales de Python:

```
python -m pip install -U catkin_pkg cryptography empy importlib-metadata lark
    ↪ ==1.1.1 lxml matplotlib netifaces numpy opencv-python
```

Instalar Qt5

Se descarga el instalador offline de Qt 5.12.X y se configuran las variables de entorno:

```
setx /m Qt5_DIR C:\Qt\Qt5.12.12\5.12.12\msvc2017_64
setx /m QT_QPA_PLATFORM_PLUGIN_PATH C:\Qt\Qt5.12.12\5.12.12\msvc2017_64\
    ↪ plugins\platforms
```

Descarga e Instalación de ROS 2

ROS 2 Humble se descarga desde la página de lanzamientos y se descomprime en `C:\dev\ros2_humble`.

Configuración del Entorno

El entorno de ROS 2 se configura ejecutando `setup.bat`:

```
call "C:\dev\ros2_humble\setup.bat"
```

Probar Algunos Ejemplos

Para verificar la instalación, se ejecutan ejemplos básicos de ROS 2: En

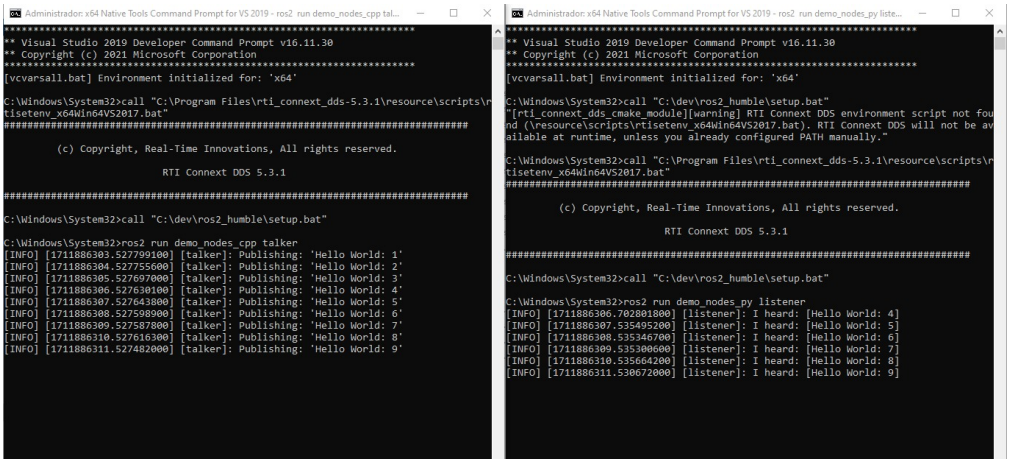

Figura 5.2: Ejecutar C++ talker y ejecutar Python listener.

dos terminales distintas se ejecutan los siguientes comandos:

```
ros2 run demo_nodes_cpp talker
ros2 run demo_nodes_py listener
```

Desinstalación

Para desinstalar ROS 2, se elimina el directorio donde se descomprimió:

```
rmdir /s /q \ros2_humble
```

5.2. Creación de un Paquete ROS2 con C++ en Windows

Para crear un paquete de ROS2 con C++ en Windows, se deben seguir los siguientes pasos:

Preparación del Entorno

Cree un espacio de trabajo para ROS2. Abra una terminal y ejecute:

```
mkdir -p c:\ros2_ws\src
cd c:\ros2_ws\src
```

Creación del Paquete

Utilice el comando `ros2 pkg create` para crear un nuevo paquete. En la terminal, navegue al directorio `src` de su espacio de trabajo y ejecute:

```
cd c:\ros2_ws\src
ros2 pkg create --build-type ament_cmake my_cpp_pkg
```

Esto creará un nuevo directorio `my_cpp_pkg` con los archivos y directorios básicos, incluyendo `package.xml` y `CMakeLists.txt`.

Construcción del Paquete

Cada vez que se actualice el archivo con modificaciones, para que ROS2 actualice el cambio, volvía al directorio del espacio de trabajo y construía el paquete usando `colcon build`:

```
cd c:\ros2_ws
colcon build --packages-select my_cpp_pkg
```

Ejecución del Nodo

Con todo correctamente construido y sin errores, preparas el entorno.

```
call install\setup.bat
```

Finalmente, ejecute el nodo usando el comando `ros2 run`(Ejemplo):

```
ros2 run my_cpp_pkg my_cpp_node
```

5.3. Instalación de ROS 2 en Ubuntu 20.04

Este apartado detalla los pasos necesarios para instalar ROS 2 Humble[25]para Ubuntu Focal.

Recursos Disponibles

Los paquetes de Debian para ROS 2 están disponibles para Ubuntu Focal en arquitecturas amd64 y arm64. La página de estado y los repositorios relevantes se pueden encontrar en el sitio web oficial de ROS 2.

Configuración del Sistema

Establecimiento del Locale

Es importante asegurar que el sistema soporte UTF-8. Para entornos mínimos, se recomienda la siguiente configuración:

```
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
```

Configuración de Fuentes de Paquetes

Para agregar el repositorio de ROS 2 al sistema, primero se debe asegurar que el repositorio Universe de Ubuntu esté habilitado. Luego, se agrega la clave GPG de ROS 2 y se configura el repositorio en la lista de fuentes:

```
sudo apt install software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
    ↪ -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros
    ↪ -archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-
    ↪ release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list
    ↪ .d/ros2.list > /dev/null
```

Instalación de Paquetes ROS 2

Después de configurar las fuentes de paquetes, se debe actualizar e instalar ROS 2 con:

```
sudo apt update
sudo apt upgrade
sudo apt install ros-foxy-desktop python3-argcomplete
```

Además, se recomienda instalar herramientas de desarrollo para compiladores y crear paquetes ROS:

```
sudo apt install ros-dev-tools
```

Configuración del Entorno

Para configurar el entorno, se deben incluir los archivos de configuración de ROS 2 Foxy en la shell:

```
source /opt/ros/foxy/setup.bash
```

Ejemplos de Prueba

Para verificar la instalación de ROS 2, se pueden ejecutar ejemplos básicos: En dos terminales distintas, se deben ejecutar los siguientes comandos:

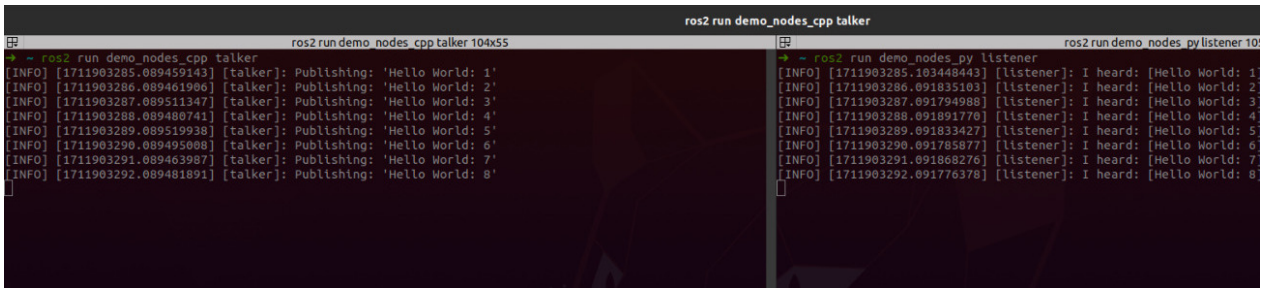

Figura 5.3: Ejecutar C++ talker y ejecutar Python listener.

```
ros2 run demo_nodes_cpp talker
ros2 run demo_nodes_py listener
```

Desinstalación

Si se necesita desinstalar ROS 2 Foxy o cambiar a una instalación basada en el código fuente, se debe ejecutar:

```
sudo apt remove ~nros-foxy-* && sudo apt autoremove
sudo rm /etc/apt/sources.list.d/ros2.list
sudo apt update
sudo apt autoremove
sudo apt upgrade
```

5.4. Creación de un Paquete ROS2 con Python en Ubuntu 20.04

Para crear un paquete de ROS2 con Python en Ubuntu 20.04, se deben seguir los siguientes pasos:

Preparación del Entorno

Cree un espacio de trabajo para ROS2. Abrir una terminal y ejecutar:

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
```

Creación del Paquete

Utilizar el comando `ros2 pkg create` para crear un nuevo paquete. En la terminal, navegar al directorio `src` del espacio de trabajo y ejecutar:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_python_pkg
```

Esto creará un nuevo directorio `my_python_pkg` con los archivos y directorios básicos, incluyendo `package.xml` y `setup.py`.

Construcción del Paquete

Cada vez que se actualice el archivo con modificaciones, para que ROS2 actualice el cambio, volver al directorio del espacio de trabajo y construir el paquete usando `colcon build`:

```
cd ~/ros2_ws
colcon build --packages-select my_python_pkg
```

Ejecución del Nodo

Con todo correctamente construido y sin errores, preparar el entorno:

```
source install/setup.bash
```

Finalmente, ejecutar el nodo usando el comando `ros2 run`(Ejemplo):

```
ros2 run my_python_pkg my_python_node
```

5.5. Comunicación entre dos ordenadores con ROS 2

Este apartado describe el proceso para establecer una comunicación efectiva entre dos ordenadores situados en la misma red utilizando ROS 2. Este proceso requiere una serie de pasos de configuración que garantizan que ambos sistemas pueden intercambiar mensajes de manera eficiente.

Inicialmente, se verifica que ambos ordenadores están en la misma subred y que no existen firewalls impidiendo la comunicación. Esto es esencial para asegurar que los mensajes de ROS 2 no sean bloqueados.

A continuación, se sincronizan ambos sistemas en el mismo dominio DDS ajustando la variable de entorno `ROS_DOMAIN_ID`. Es importante que este valor sea el mismo en ambos ordenadores para que operen en el mismo espacio de comunicaciones de ROS 2. El comando utilizado es el siguiente:

```
export ROS_DOMAIN_ID=42
```

donde ‘42‘ es un ejemplo de un número de dominio elegido.

Para asegurar que los ordenadores pueden comunicarse entre sí, se utiliza el comando `ping` para probar la conectividad de la red. Al confirmar que hay respuesta del ping, se sabe que los ordenadores pueden verse y comunicarse entre sí en la red.

El siguiente paso es iniciar un nodo ROS 2 en uno de los ordenadores. Se ejecuta un nodo `talker` en C++ que publica mensajes:

```
ros2 run demo_nodes_cpp talker
```

Simultáneamente, en el otro ordenador, se inicia un nodo `listener` en Python que se suscribe a los mensajes:

```
ros2 run demo_nodes_py listener
```

Esta configuración permite observar la comunicación correcta entre los nodos de distintos sistemas. A modo de ejemplo, se muestra la demo de ROS 2 para verificar que funciona (véase la Figura 5.4).

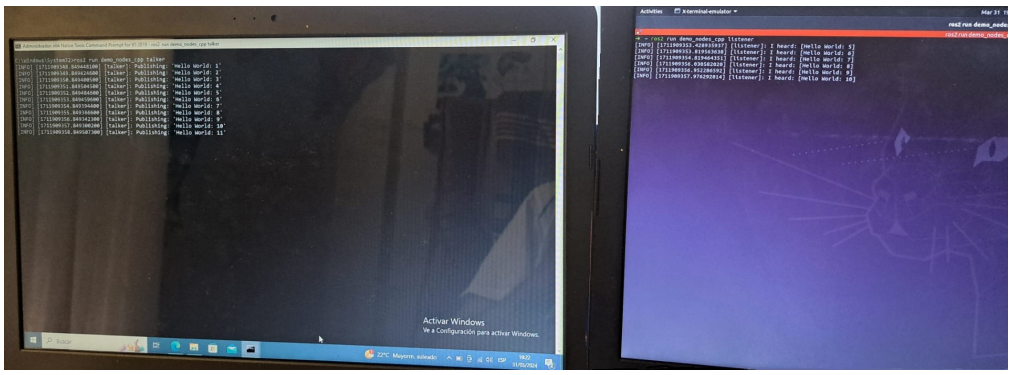

Figura 5.4: Comunicación entre dos ordenadores con ROS 2

5.6. Desarrollo en el Sistema Windows

Después de asegurar el correcto funcionamiento de ROS 2 y establecer el entorno de trabajo junto con el paquete necesario, se procede al desarrollo del software para la publicación de los datos. En esta sección se proporciona un análisis detallado del desarrollo, basado en el código y las configuraciones

descritas en los archivos del proyecto. El análisis abarca desde la configuración inicial del entorno de desarrollo hasta la ejecución del software para la transmisión de datos.

5.6.1. Configuración Inicial y Definición del Proyecto en Windows

El proceso inicia con la configuración del entorno CMake, necesario para la construcción del software en C++ de `readExport_BVH.cpp`, encargado de leer y transmitir los datos. Este entorno se define en el archivo `CMakeLists.txt`, donde se especifica la versión mínima de CMake requerida:

```
cmake_minimum_required(VERSION 3.8)
```

Se declara el nombre del proyecto:

```
project(my_cpp_pkg)
```

Se agregan opciones de compilación adicionales si el compilador es GNU C++ o Clang. Las opciones `-Wall`, `-Wextra` y `-Wpedantic` activan advertencias adicionales para mejorar la calidad del código:

```
if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()
```

Se buscan los paquetes de ROS 2 necesarios, asegurando que estén presentes para la compilación y el enlazado del proyecto. También se especifica `my_cpp_interfaces`, que contiene interfaces personalizadas de comunicación necesarias para el proyecto:

```
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(my_cpp_interfaces REQUIRED)
```

Se configuran una biblioteca importada estática y un ejecutable, definiendo sus dependencias, directorios de inclusión y bibliotecas enlazadas. Es en esta parte donde se especifica al archivo C++ de ROS 2 que se utilizará NeuronDataReader:

```
add_library(neuron STATIC IMPORTED)
set_target_properties(neuron PROPERTIES IMPORTED_LOCATION C:/dev/project/
    ↪ Illera1999Project_NDR_ROS2_UR05/src/my_cpp_pkg/neuron_reader/lib/
    ↪ NeuronDataReader.lib)

add_executable(bvh_read src/readExport_BVH.cpp)
ament_target_dependencies(bvh_read rclcpp my_cpp_interfaces)

target_include_directories(bvh_read PUBLIC neuron_reader/include)
target_link_libraries(bvh_read neuron)
```

Se configura la instalación del ejecutable dentro de la estructura del paquete, facilitando su uso y distribución:

```
install(TARGETS bvh_read DESTINATION lib/${PROJECT_NAME})
```

Esta función prepara el paquete para ser utilizado en un entorno ROS 2, asegurando que todas las configuraciones sean correctas y estén listas para la distribución y ejecución:

```
ament_package()
```

5.6.2. Recepción y Publicación de Datos de Movimiento de Axis Neuron en ROS 2

El código en C++ está diseñado para utilizar el SDK NeuronDataReader con el fin de procesar datos de captura de movimiento y publicarlos en un tópico mediante un nodo de ROS 2, permitiendo así que el sistema Linux reciba los datos. A continuación, se describe cada parte del código en detalle:

Se incluyen las bibliotecas necesarias para el funcionamiento del nodo en ROS 2. Se utilizan `rclcpp/rclcpp.hpp` para las funcionalidades básicas de ROS 2 y `my_cpp_interfaces/msg/data_right_bvh.hpp` para los mensajes personalizados que gestionan los datos de captura de movimiento:

```
#include "rclcpp/rclcpp.hpp"
#include "my_cpp_interfaces/msg/data_right_bvh.hpp"
#include "NeuronDataReader.h"
#include "DataType.h"
#include "iostream"
#include "windows.h"
#include "string"
#include "vector"
#include "mutex"
```

Se define un `mutex` para asegurar la exclusión mutua al acceder a los datos compartidos, y una estructura `BoneData` para almacenar los datos de los huesos del brazo derecho:

```
std::mutex myMutex;

struct BoneData
{
    std::string name;
    float rx, ry, rz;
    float dx, dy, dz;
};

std::array<BoneData, 4> rightArm;
std::string nameBone[] = { "RigthShoulder", "RightArm", "RightForeArm", "
    ↪ RightHand"};
```

El método `printCalcData` imprime los datos del brazo derecho, incluyendo nombres, rotaciones y desplazamiento de los huesos. Es importante destacar que `rightArm` no se refiere a un solo hueso en este caso, sino que se utiliza para imprimir los datos de los cuatro huesos empleados en el cálculo de los ángulos del brazo robótico.:

```
void printCalcData()
{
    std::cout << "Datos del brazo: " << std::endl;
    for(BoneData data: rightArm){
        std::cout << "Nombre: " << data.name << std::endl;
        std::cout << "Rotaciones: " << std::endl;
        std::cout << "{" << data.rx << ", " << data.ry << ", "<< data.rz << "}
            ↪ " << std::endl;
        std::cout << "Desplazamiento: " << std::endl;
        std::cout << "{" << data.dx << ", " << data.dy << ", "<< data.dz << "}
            ↪ " << std::endl;
        std::cout << "\n" << std::endl;
    }

    std::cout << "\n" << std::endl;
}
```

El *callback* `frameDataFromHand` procesa los datos del marco del sensor en formato BVH y actualiza la estructura `rightArm`:

```
static void frameDataFromHand(void* customedObj, SOCKET_REF sender,
    ↪ BvhDataHeader* header, float* data)
{
    int aux = 0;
    int sds = 13;

    myMutex.lock();

    for(BoneData& arm: rightArm)
    {
        arm.name = nameBone[aux];
        arm.dx = data[(sds * 6)];
        arm.dy = data[(sds * 6) + 1];
        arm.dz = data[(sds * 6) + 2];
        arm.ry = data[(sds * 6) + 3];
        arm.rx = data[(sds * 6) + 4];
        arm.rz = data[(sds * 6) + 5];
        sds = sds + 1;
        aux += 1;
    }
    myMutex.unlock();
}
```

El nodo `ExportDataAxisNeuron` se encarga de publicar los datos recibidos del sensor. El constructor del nodo inicializa el publicador y configura un temporizador para llamar al método `publishDataAxisNeuron` periódicamente:

```
class ExportDataAxisNeuron : public rclcpp::Node
{
public:
    ExportDataAxisNeuron() : Node("export_data_axisneuron")
```

```
    {
        pub_ = this->create_publisher<my_cpp_interfaces::msg::DataRightBVH>(
            "data_right_arm", 10);
        timer_ = this->create_wall_timer(
            std::chrono::seconds(1),
            std::bind(&ExportDataAxisNeuron::publishDataAxisNeuron, this));
        RCLCPP_INFO(this->get_logger(), " ReadExport publisher has been
            ↪ started");
    }

private:
    void publishDataAxisNeuron()
    {
        auto msg = my_cpp_interfaces::msg::DataRightBVH();
        int auxi = 0;
        int auxii = 0;
        int auxiii = 0;
        myMutex.lock();
        for (const BoneData& data: rightArm){
            msg.name[auxi] = data.name;
            msg.rotation[auxii] = data.rx;
            msg.rotation[auxii + 1] = data.ry;
            msg.rotation[auxii + 2] = data.rz;
            msg.position[auxii] = data.dx;
            msg.position[auxii + 1] = data.dy;
            msg.position[auxii + 2] = data.dz;
            auxi += 1;
            auxii += 3;
            auxiii += 4;
        }
        printCalcData();
        myMutex.unlock();
        pub_->publish(msg);
    }

    rclcpp::Publisher<my_cpp_interfaces::msg::DataRightBVH>::SharedPtr pub_;
    rclcpp::TimerBase::SharedPtr timer_;
};
```

La función principal configura los *callbacks* y se conecta al servidor de `AxisNeuron`. Luego inicia el nodo de ROS 2 para la publicación de datos:

```
int main(int argc, char **argv)
{
    BRRegisterFrameDataCallback(nullptr, frameDataFromHand);

    char serverIP[] = "127.0.0.1";
    int port_bvh = 7001;

    SOCKET_REF socketRef_bvh = BRConnectTo(serverIP, port_bvh);

    while(true)
    {
        if (socketRef_bvh != NULL)
        {
            SocketStatus ssStatus_bvh = BRGetSocketStatus(socketRef_bvh);
            switch (ssStatus_bvh)
            {
                case CS_Running:
                    std::cout << "Conectado BVH\n";
                    break;
```

```
                case CS_Starting:
                    std::cout << "Iniciando ... \n";
                    break;
                case CS_OffWork:
                    std::cout << "OffLine \n";
                    break;
            }
            rclcpp::init(argc, argv);
            auto node = std::make_shared<ExportDataAxisNeuron>();
            rclcpp::spin(node);
            rclcpp::shutdown();

        } else
        {
            std::cerr << "No se pudo establecer la conexi n." << std::endl;
        }
        Sleep(500);
    }
    BRCloseSocket(socketRef_bvh);
}
```

5.7. Desarrollo en el Sistema Linux

Se ha desarrollado un módulo que se importa al archivo principal para calcular los ángulos específicos necesarios para el funcionamiento del sistema de conexión con UR5e. A continuación, se explica su contenido.

5.7.1. Funciones de Transformación

Las funciones de transformación se utilizan para realizar cálculos de transformación y rotación en un sistema de referencia tridimensional. El código implementado hace uso de la biblioteca `numpy` y funciones matemáticas específicas del módulo `math`.

```
import numpy as np
from math import acos, atan2, radians, degrees, sin, cos, asin
```

Inicialmente, se importan las bibliotecas `numpy` y `math`, necesarias para llevar a cabo operaciones matemáticas y de álgebra lineal.

```
T0_Sp = np.array([
    [0, 0, 1, 0],
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 1]
])
```

El sistema de referencia de la espina dorsal (`Sp`) no está orientado de la misma forma que el sistema de referencia estándar. Por lo tanto, se define una matriz de transformación `T0_Sp` para convertir entre estos sistemas de referencia.

```
def trans1(d1):
    result = np.array([
        [1, 0, 0, d1[0]],
        [0, 1, 0, d1[1]],
        [0, 0, 1, d1[2]],
        [0, 0, 0, 1],
    ])
    return result
```

La función `trans1` toma un vector `d1` y devuelve una matriz de transformación de traslación. Esta matriz desplaza un punto en el espacio tridimensional por las cantidades especificadas en `d1`.

```
def trotx(beta):
    beta = radians(beta)
    result = np.array([
        [1, 0, 0, 0],
        [0, cos(beta), -sin(beta), 0],
        [0, sin(beta), cos(beta), 0],
        [0, 0, 0, 1]
    ])
    return result
```

La función `trotx` toma un ángulo `beta` en grados, lo convierte a radianes y devuelve una matriz de transformación que representa una rotación alrededor del eje `x`.

```
def troty(alpha):
    alpha = radians(alpha)
    result = np.array([
        [cos(alpha), 0, sin(alpha), 0],
        [0, 1, 0, 0],
        [-sin(alpha), 0, cos(alpha), 0],
        [0, 0, 0, 1]
    ])
    return result
```

La función `troty` toma un ángulo `alpha` en grados, lo convierte a radianes y devuelve una matriz de transformación que representa una rotación alrededor del eje `y`.

```
def trotz(gamma):
    gamma = radians(gamma)
    result = np.array([
        [cos(gamma), -sin(gamma), 0, 0],
        [sin(gamma), cos(gamma), 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    return result
```

La función `trotz` toma un ángulo `gamma` en grados, lo convierte a radianes y

devuelve una matriz de transformación que representa una rotación alrededor del eje `z`.

Las siguientes funciones se utilizan para calcular matrices de transformación compuestas que integran traslaciones y rotaciones en un sistema de referencia tridimensional.

```
def Th(Displacement1, Rotation1, Displacement2):
    d1 = Displacement1
    d2 = Displacement2
    beta = Rotation1[1]
    alpha = Rotation1[2]
    gamma = Rotation1[3]
    Td1 = trans1(d1)
    Ty = troty(alpha)
    Tx = trotx(beta)
    Tz = trotz(gamma)
    Td2 = trans1(d2)

    matrix_rounded = Td1 @ Ty @ Tx @ Tz @ Td2
    return matrix_rounded
```

La función `Th` calcula una matriz de transformación compuesta a partir de dos vectores de desplazamiento y un vector de rotación. Todo está explicado en el apartado 4.1.

```
def Thr(Displacement1, ArmRotation, Displacement2):

    Rotationy, Rotationz = rotation_matrix(ArmRotation[1], ArmRotation[2],
        ↪ ArmRotation[3])
    d1 = Displacement1
    d2 = Displacement2
    Td1 = trans1(d1)
    Ty = troty(Rotationy)
    Tz = trotz(Rotationz)
    Td2 = trans1(d2)

    matrix_rounded = Td1 @ Ty @ Tz @ Td2
    return matrix_rounded, Rotationy, Rotationz
```

La función `Thr` calcula una matriz de transformación compuesta similar a la función `Th`, pero utiliza una función adicional `rotation_matrix` para calcular los ángulos de rotación.

```
def rotation_matrix(rotacion1, rotacion2, rotacion3):
    titax = radians(rotacion1)
    titay = radians(rotacion2)
    titaz = radians(rotacion3)

    tita2z = degrees(asin(cos(titax) * sin(titaz)))
    tita2y = degrees(asin((cos(titaz) * sin(titay) - cos(titay) * sin(titax) *
        ↪  sin(titaz)) / cos(radians(tita2z))))
    return tita2y, tita2z
```

La función `rotation_matrix` calcula los ángulos de rotación necesarios para una matriz de transformación tridimensional.

```
def last_column_matches(matrix1, matrix2, tol=1e-4):
    last_col_matrix1 = matrix1[:, -1]
    last_col_matrix2 = matrix2[:, -1]

    return np.allclose(last_col_matrix1, last_col_matrix2, atol=tol)
```

La función `last_column_matches` compara las últimas columnas de dos matrices para verificar si son iguales dentro de una tolerancia especificada. Esto permite verificar que el desplazamiento en ambas matrices de transformación homogénea es el mismo y deducir que se pueden igualar.

```
def angle_arm(armDisplacement, armRotation, foreArmDisplacement):
    Tcb = Th(armDisplacement, armRotation, foreArmDisplacement)
    Tcb1, arm_b, arm_h = Thr(armDisplacement, armRotation, foreArmDisplacement
        ↪ )
    if last_column_matches(Tcb, Tcb1):
        return arm_b, arm_h
    else:
        return arm_b, arm_h
```

La función `angle_arm` calcula los ángulos de rotación del brazo y el ángulo de apertura del hombro.

```
def Tc(Displacement, Rotation):
    d1 = Displacement
    beta = Rotation[1]
    alpha = Rotation[2]
    gamma = Rotation[3]
    Td1 = trans1(d1)
    Ty = troty(alpha)
    Tx = trotx(beta)
    Tz = trotz(gamma)

    matrix_rounded = Td1 @ Ty @ Tx @ Tz
    return matrix_rounded
```

La función `Tc` calcula una matriz de transformación compuesta a partir de un vector de desplazamiento y un vector de rotación.

```
def angle_forearm(Displacement1, Rotation1, Displacement2, Rotation2,
    ↪ Displacement3, Rotation3, Displacement4, Rotation4):
    TSp_Sh = Tc(Displacement1, Rotation1)
    TSh_A = Tc(Displacement2, Rotation2)
    TA_FA = Tc(Displacement3, Rotation3)
    TFA_H = Tc(Displacement4, Rotation4)

    T0_Sp
    T0_Sh = T0_Sp @ TSp_Sh
    T0_A = T0_Sh @ TSh_A
    T0_FA = T0_A @ TA_FA
    T0_H = T0_FA @ TFA_H
```

```
    p_FA_A = T0_A[:3, 3] - T0_FA[:3, 3]
    p_FA_H = T0_H[:3, 3] - T0_FA[:3, 3]

    cos_angle = np.dot(p_FA_A, p_FA_H) / (np.linalg.norm(p_FA_A) * np.linalg.
        ↪ norm(p_FA_H))
    elbow_angle = np.degrees(np.arccos(cos_angle))

    elbow_angle = 180 - elbow_angle

    return elbow_angle
```

La función `angle_forearm` calcula el ángulo del codo a partir de varios vectores de desplazamiento y rotación.

```
def is_arm_left(shoulderDisplacement, shoulderRotation, armDisplacement,
    ↪ armRotation, foreArmDisplacement, foreArmRotation, handDisplacement,
    ↪ handRotation):
    T_shoulder = Tc(shoulderDisplacement, shoulderRotation)
    T_arm = Tc(armDisplacement, armRotation)
    T_forearm = Tc(foreArmDisplacement, foreArmRotation)
    T_hand = Tc(handDisplacement, handRotation)

    initial_position = np.array([0, 0, 0, 1])

    shoulder_position = T_shoulder @ initial_position

    T_total = T_arm @ T_forearm @ T_hand

    final_position = T_total @ np.array([0, 0, 0, 1])

    relative_position = final_position + shoulder_position

    return not relative_position[0] < 0
```

La función `is_arm_left` determina si un brazo está a la izquierda o a la derecha en relación con el sistema de referencia. Esta función se utiliza para ajustar los ángulos de la base del brazo.

```
def calculate_hand_orientation(armRotation, foreArmRotation, handRotation):
    angulo = handRotation[1] - foreArmRotation[1] - armRotation[1]
    print("Angulo mu eca_2 sin tratar " + str(angulo))
    angulo -= 90
    if angulo > 0:
        angulo = 0
    if angulo < -180:
        angulo = -180
    return angulo
```

La función `calculate_hand_orientation` calcula la orientación de la mano en función de las rotaciones del brazo, antebrazo y mano.

Estas funciones permiten determinar la posición del brazo a partir de ángulos que se puedan utilizar para el UR5e y ahora se procederá a explicar el hilo principal y en donde se usan las funciones principales para pasar los ángulos.

5.7.2. Recepción y Transmisión de Datos Tratados a UR5e

El siguiente código realiza la importación de varios módulos necesarios para el funcionamiento del programa:

```
#!/usr/bin/env python3
import rclpy
import time
import rtde_control
import numpy as np
from rclpy.node import Node
from my_cpp_interfaces.msg import DataRightBVH
from my_py_pkg import transformation_angle
import math
import time
```

La clase `BoneData` se utiliza para almacenar y procesar los datos relacionados con el brazo derecho.

El constructor inicializa los atributos de la clase con los datos proporcionados:

```
class BoneData:
    def __init__(self, name, rotation, position):
        self.RightShoulder = [name[0], rotation[0], rotation[1], rotation[2]]
        self.RightArm = [name[1], rotation[3], rotation[4], rotation[5]]
        self.RightForeArm = [name[2], rotation[6], rotation[7], rotation[8]]
        self.RightHand = [name[3], rotation[9], rotation[10], rotation[11]]
        self.RightShoulderDisplacement = [position[0], position[1], position
            ↪ [2]]
        self.RightArmDisplacement = [position[3], position[4], position[5]]
        self.RightForeArmDisplacement = [position[6], position[7], position
            ↪ [8]]
        self.RightHandDisplacement = [position[9], position[10], position[11]]
        self.name = name
```

Este método imprime los datos del hombro, brazo, antebrazo y mano:

```
    def printData(self):
        print("-----------------------------------")
        print("Datos del Hombro: ")
        print(str(self.RightShoulder))
        print("            Desplazamiento del Hombro: ")
        print(str(self.RightShoulderDisplacement))
        print("Datos del Brazo: ")
        print(str(self.RightArm))
        print("            Desplazamiento del Brazo: ")
        print(str(self.RightArmDisplacement))
        print("Datos del Antebrazo: ")
        print(str(self.RightForeArm))
        print("            Desplazamiento del Antebrazo: ")
        print(str(self.RightForeArmDisplacement))
        print("Datos de la Mano: ")
        print(str(self.RightHand))
        print("            Desplazamiento de la Mano: ")
        print(str(self.RightHandDisplacement))
        print("-----------------------------------")
```

Estos métodos calculan los ángulos de las diferentes partes del brazo utilizando funciones del módulo `transformation_angle`:

```
    def arm_b(self):
        arm_b, _ = transformation_angle.angle_arm(
            self.RightArmDisplacement, self.RightArm,
            self.RightForeArmDisplacement)
        is_left = transformation_angle.is_arm_left(
            self.RightShoulderDisplacement, self.RightShoulder,
            self.RightArmDisplacement, self.RightArm,
            self.RightForeArmDisplacement, self.RightForeArm,
            self.RightHandDisplacement, self.RightHand
        )

        if(is_left):
          arm_b = 180 - arm_b
        print("ngulo de la base")
        print(arm_b)
        return math.radians(arm_b)

    def arm_h(self):
        _, arm_h = transformation_angle.angle_arm(self.RightArmDisplacement,
            ↪ self.RightArm, self.RightForeArmDisplacement)
        print("ngulo del hombro")
        print(arm_h)
        return math.radians(arm_h)

    def arm_c(self):
        arm_c = transformation_angle.angle_forearm(
            self.RightShoulderDisplacement, self.RightShoulder,
            self.RightArmDisplacement, self.RightArm,
            self.RightForeArmDisplacement, self.RightForeArm,
            self.RightHandDisplacement, self.RightHand
        )
        print("ngulo del codo")
        print(arm_c)
        return -math.radians(arm_c)

    def arm_m1(self):
        _, arm_m1 = transformation_angle.angle_arm(self.
            ↪ RightForeArmDisplacement, self.RightForeArm, self.
            ↪ RightHandDisplacement)
        arm_m1 = arm_m1 - 90
        print("ngulo del codo")
        print(arm_m1)
        return math.radians(arm_m1)

    def arm_m2(self):
        arm_m2 = transformation_angle.calculate_hand_orientation(
            self.RightArm,
            self.RightForeArm,
            self.RightHand)
        print("ngulo del codo")
        print(arm_m2)
        return math.radians(arm_m2)
```

La clase `ImportData` se extiende de `Node` y se utiliza para suscribirse a los datos del brazo derecho y enviar comandos de movimiento al robot.

```
class ImportData(Node):
    def __init__(self):
```

```
        super().__init__("import_data_node")
        self.number_subscriber_ = self.create_subscription(
            DataRightBVH, "data_right_arm", self.callback_data, 10)
        self.get_logger().info("ImportData has been started.")
        self.iteration_count = 0

    def callback_data(self, msg):
        rightArm = BoneData(msg.name, msg.rotation,  msg.position)
        self.iteration_count += 1
        print(self.iteration_count)
        print(rightArm.printData())
        rtde_c.moveJ([rightArm.arm_b(), rightArm.arm_h(), rightArm.arm_c(),
            ↪ rightArm.arm_m1(), rightArm.arm_m2(), 0], 1, 1)
```

Se establece una interfaz de control RTDE para comunicarse con el robot:

```
rtde_c = rtde_control.RTDEControlInterface("192.168.0.100")
```

La clase `rtde_control.RTDEControlInterface` se utiliza para establecer una conexión con el robot UR5e. En este caso, se especifica la dirección IP del robot como "192.168.0.100". Una vez establecida la conexión, se pueden enviar comandos al robot.

La función `moveJ` se utiliza para mover el robot a una posición específica en el espacio de las articulaciones. Esta función toma como parámetros una lista de posiciones de las articulaciones, una velocidad y una aceleración. En el ejemplo proporcionado, se mueve el robot a una posición inicial con los valores de las articulaciones especificados:

```
rtde_c.moveJ([1.5708, 0, 0, -1.5708, -1.5708, 0], 1, 1)
```

La función principal inicializa el sistema y ejecuta el nodo:

```
def main(args=None):
    rtde_c.moveJ([1.5708, 0, 0, -1.5708, -1.5708, 0], 1, 1)
    rclpy.init(args=args)
    node = ImportData()
    rclpy.spin(node)
    rtde_c.stopJ()
    rclpy.shutdown()

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

En la función `main`, se inicializa el sistema ROS 2 con `rclpy.init()`, se crea una instancia de la clase `ImportData` y se llama a `rclpy.spin()` para mantener el nodo en ejecución. Antes de finalizar, se envía un comando para detener cualquier movimiento en curso del robot utilizando `rtde_c.stopJ()` y se cierra el sistema ROS 2 con `rclpy.shutdown()`.

En resumen, el código establece una conexión con el robot UR5e mediante la librería `rtde_control`, recibe datos de un tópico ROS 2, procesa esos datos para calcular los ángulos de las articulaciones del brazo derecho y envía comandos de movimiento al robot en función de estos cálculos.

Capítulo 6

Experimentos y Resultados

6.1. Experimento

6.1.1. Configuración Experimental

La configuración experimental consistió en el brazo robótico UR5e de 6 ejes utilizado como guía para el movimiento, el traje de Perception Neuron y el sistema de teleoperación, como se describe en el Capítulo 3.

6.1.2. Protocolo Experimental

Se distanciaron todos los dispositivos y se elevó el brazo para minimizar el riesgo de colisión. Posteriormente, se configuró el sistema Perception Neuron y se conectó al PC con Windows mediante un cable USB. En el software de Axis Neuron se estableció la configuración descrita en el Capítulo 3 y se seleccionó el modo 'single arm'. La altura del esqueleto fue fijada y se realizaron los movimientos requeridos para la calibración de los sensores.

A continuación, se ejecutó el software en Windows y se esperó a que los datos recopilados por Axis Neuron se imprimieran en la terminal, lo que indicaba que el nodo de Windows leía y publicaba los datos correctamente. Con la parte de Windows funcionando, se procedió a ejecutar el software en Linux, el cual recopilaba los datos. Una vez ejecutado, se verificó que los datos impresos en la terminal de Linux coincidieran con los datos publicados desde Windows. Con esta verificación completada, se tomó directamente el control del brazo robótico.

El proceso de experimentación se llevó a cabo de la siguiente manera:

- Traslaciones a lo largo de los ejes cartesianos:

  - Traslaciones de 30 cm a lo largo de los ejes x, y y z, con pausas de 1,5 s entre movimientos en dirección positiva y negativa.

- Rotaciones alrededor de los ejes cartesianos:

  - Rotaciones de 45° alrededor de los ejes x, y y z, con pausas de 1,5 s entre rotaciones en dirección positiva y negativa.

Se probó un mínimo de 5 veces cada movimiento para evaluar la respuesta del sistema. La velocidad de los movimientos fue fija, marcada por el código. Los datos de posición y orientación fueron registrados y analizados para evaluar la precisión y exactitud del sistema de teleoperación.

Cada serie de movimientos tuvo una duración total de 3-2 minutos y calibraciones entre series.

6.2. Resultado y Percepción Pública

El sistema logró capturar los datos de movimiento en tiempo real y transmitirlos eficazmente entre los ordenadores, permitiendo que el brazo robótico imitara los movimientos humanos con alta precisión. Las pruebas, evaluadas de manera perceptual, demostraron la robustez y eficiencia del sistema en la teleoperación del brazo robótico UR5e.

Se observó que el brazo robótico presentaba movimientos bruscos en ciertas ocasiones. Esto se debe a que el robot recibe continuamente nuevos ángulos como metas a alcanzar. Mientras se dirige a un ángulo objetivo, otros objetivos pueden entrar en cola, lo que resulta en movimientos interrumpidos. Esta característica es más evidente cuando el movimiento del brazo es lento, ya que el robot sigue leyendo y tratando de alcanzar cada ángulo que llega como nueva meta, provocando los tirones observados.

Adicionalmente, se identificó una especie de desviación (drift) en el movimiento del robot. Cuando el brazo se posiciona en una extensión total recta, la parte final del robot tiende a elevarse. Este fenómeno podría atribuirse al drift, y se considera un aspecto a mejorar para afinar la reproducción del movimiento del robot de manera más precisa.

Figura 6.1: UR5e imitando los movimientos.

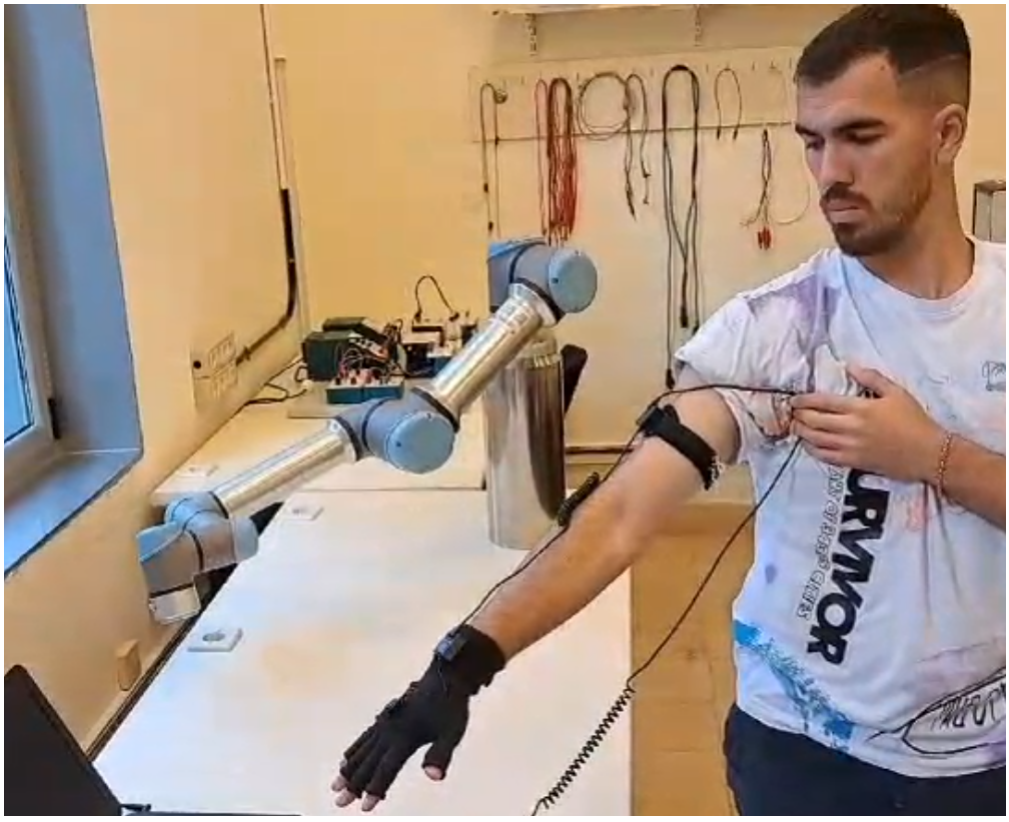

Figura 6.2: Otra posición con el UR5e.

Para evaluar el trabajo de la imitación se realizo un cueationario de seis preguntas al público sobre el sistema de teleoperación del brazo robótico UR5e, basado en un video subido a Instagram[26]. Este cuestionario tenía como objetivo recoger las reacciones y opiniones de las personas sobre la precisión,

suavidad y eficiencia del brazo robótico imitando movimientos humanos.

6.2.1. Resultados del Cuestionario

El cuestionario fue respondido por un total de 10 personas. Los resultados obtenidos fueron los siguientes:

- **Precisión de los movimientos**: El 40 % de los evaluadores calificaron la precisión de los movimientos del brazo robótico como "Muy precisa", mientras que el 50 % la calificaron como "Precisa". Solo el 10 % consideró que la precisión era "Moderadamente precisa". Esto sugiere que la mayoría de los evaluadores consideraron que el sistema logra una imitación fiel de los movimientos humanos.

- **Movimientos bruscos**: El 70 % de los participantes no observó ningún movimiento brusco, mientras que el 30 % indicó haber observado muy pocos movimientos bruscos. Esto indica que, aunque el sistema es mayormente suave, hay ocasiones en las que pueden ocurrir tirones.

- **Desviación (drift)**: En cuanto a la desviación cuando el brazo está en posición extendida, el 40 % de los evaluadores no observó ninguna desviación, el 40 % notó muy poca desviación, y el 20 % indicó haber observado algo de desviación. Esto sugiere que el fenómeno de elevación en la parte final del robot es mínimo pero perceptible.

- **Mejoras en el rendimiento**: En cuanto a la necesidad de mejoras en el rendimiento del brazo robótico, el 30 % de los evaluadores consideraron que se requieren muy pocas mejoras, otro 30 % opinó que no se necesitan mejoras, un 30 % indicó que se necesitan algunas mejoras, y un 10 % sugirió que se necesitan muchas mejoras.

- **Recomendación**: La recomendación del sistema para aplicaciones prácticas fue positiva, con el 50 % de los evaluadores indicando que definitivamente recomendarían el sistema y el otro 50 % recomendándolo con algunas mejoras.

En conclusión, los resultados del cuestionario indican que el sistema de teleoperación del brazo robótico UR5e es preciso y eficiente en la imitación de movimientos humanos. Sin embargo, se identificaron áreas de mejora, como la reducción de movimientos bruscos y la corrección de la desviación en posición extendida. La mayoría de los evaluadores recomendaría el sistema para aplicaciones prácticas, aunque algunos sugieren realizar mejoras adicionales

para optimizar su rendimiento.

Capítulo 7

Conclusiones

7.1. Conclusiones

El desarrollo del proyecto ha aprovechado significativamente el SDK NeuronDataReader para capturar datos precisos y en tiempo real del traje Axis Neuron, lo cual ha sido fundamental para la teleoperación del sistema robótico. Esta tecnología ha permitido una lectura eficiente de las señales biomecánicas, facilitando la implementación de comandos complejos a través de movimientos humanos naturales.

El trabajo con ROS2 ha permitido una integración efectiva en ambientes de desarrollo Linux y Windows, enfrentando retos notables en términos de compatibilidad y rendimiento del sistema. La utilización de librerías específicas de Python ha mejorado la manipulación y procesamiento de los datos recibidos, lo cual ha sido crucial para mantener la fluidez y la precisión del sistema de control del robot.

Se han enfrentado desafíos significativos en la calibración y precisión de los movimientos del robot, especialmente en la interpretación de los datos del IMU y su traducción a movimientos mecánicos precisos. A pesar de las limitaciones técnicas y los desafíos de integración, se han logrado avances sustanciales en la comprensión y aplicación de técnicas de control robótico avanzadas.

El sistema desarrollado demostró ser capaz de capturar los datos de movimiento en tiempo real y transmitirlos eficazmente entre los ordenadores, permitiendo que el brazo robótico imitara los movimientos humanos con al-

ta precisión. Las pruebas realizadas confirmaron la robustez y eficiencia del sistema en la teleoperación del brazo robótico UR5e. Aunque se observaron movimientos bruscos debido a la continua recepción de nuevos ángulos como metas a alcanzar, el sistema fue capaz de mantener una comunicación efectiva en tiempo real entre los nodos de Windows y Linux.

El proyecto ha contribuido a la literatura existente proporcionando insights prácticos sobre la implementación de sistemas de teleoperación avanzados y ha establecido una base sólida para futuras investigaciones, que podrían explorar la integración de inteligencia artificial para mejorar la autonomía y adaptabilidad del sistema robótico.

En conclusión, este proyecto ha demostrado el potencial de las tecnologías de captura de movimiento y teleoperación en la robótica, ofreciendo nuevas posibilidades para aplicaciones futuras en campos como la medicina, la educación y la industria. Estos resultados subrayan la importancia de continuar avanzando en la investigación y desarrollo en estas áreas, destacando las oportunidades para mejorar la precisión, la eficiencia y la aplicabilidad de los sistemas de teleoperación robótica.

7.2. Trabajos Futuros

Existen varias direcciones prometedoras para extender y mejorar el trabajo realizado en este proyecto:

Aplicación Móvil para Teleoperación

Un desarrollo futuro significativo podría ser la creación de una aplicación móvil utilizando Flutter, que facilite la teleoperación del brazo robótico. Flutter, con su administrador de paquetes que incluye una librería dedicada a ROS2, permite una integración robusta con dispositivos móviles. Esto permitiría a los usuarios controlar el brazo robótico desde cualquier lugar, ofreciendo una solución más flexible y accesible.

Mejora de Algoritmos de Control

La implementación de algoritmos de control más avanzados podría reducir los movimientos bruscos observados. Algoritmos predictivos o de aprendizaje automático pueden anticipar los próximos movimientos y ajustar la respuesta del brazo robótico en consecuencia, mejorando la fluidez del movimiento, he incluso añadir inteligencia artificial para esa predicción.

Implementación de un Gancho Controlado por Perception Neuron

Una posible mejora del sistema es la adición de un gancho en el extremo del brazo robótico, el cual podría ser controlado también mediante el sistema Perception Neuron. Esto ampliaría las capacidades del brazo robótico, permitiendo su uso en aplicaciones que requieran la manipulación precisa de objetos, como la carga y descarga en entornos industriales o la realización de tareas de rescate en situaciones de emergencia. El control del gancho mediante los movimientos del operador mejoraría la versatilidad y utilidad del sistema de teleoperación.

Capítulo 8

Anexos

UNIVERSAL ROBOTS

UR5e
Ficha técnica

El UR5e brinda máxima flexibilidad para aplicaciones de carga media con una carga útil de hasta 5 kg y un alcance de 850 mm.

Hasta la fecha, hemos entregado más de 50.000 robots industriales colaborativos UR a clientes de todas las industrias y de todo el mundo. UR5e es uno de los cuatro cobots e-Series, cada uno con una combinación de carga útil y alcance particular. e-Series brinda una flexibilidad increíble y una facilidad de uso incomparable para su aplicación.

UR5e

Especificaciones

Carga útil	5 kg (11 lbs)
Alcance	850 mm (33,5 in)
Grados de libertad	6 articulaciones giratorias
Programación	Pantalla táctil de 12" con interfaz gráfica de usuario Polyscope

Rendimiento

Consumo promedio máximo	570 W
Consumo típico en régimen de operación moderado	200 W
Seguridad	17 funciones de seguridad configurables
Certificaciones	EN ISO 13849-1, PLd Categoría 3, y EN ISO 10218-1

Sensor de fuerza brida de la herramienta	Fuerza, x-y-z	Par, x-y-z
Rango	50,0 N	10,0 Nm
Resolución	3,5 N	0,2 Nm
Precisión	4,0 N	0,3 Nm

Movimiento

Repetibilidad según la norma ISO 9283	± 0,03 mm	
Movimiento de ejes	Rango de trabajo	Velocidad máxima
Base	± 360°	± 180°/s
Hombro	± 360°	± 180°/s
Codo	± 360°	± 180°/s
Muñeca 1	± 360°	± 180°/s
Muñeca 2	± 360°	± 180°/s
Muñeca 3	± 360°	± 180°/s
Velocidad TCP estándar	1 m/s (39,4 in/s)	

Funciones

Clasificación IP	IP54
Clase ISO 14644-1 Sala limpia	5
Ruido	Menos de 65 dB(A)
Montaje del robot	Cualquier orientación
Puertos de E/S en herramienta	
Entradas digitales	2
Salidas digitales	2
Entradas analógicas	2
Voltaje en E/S de herramienta	12/24 V
Alimentación E/S herramienta	1,5 A (doble pin) 1 A (pin individual)

Características físicas

Huella	Ø 149 mm
Materiales	Aluminio, plástico, acero
Tipo de conector para herramientas	M8 | M8 8-pin
Long. cable del brazo robótico	6 m (236 in) cable incluido. 2 m (472 in) y opciones de alta flexibilidad disponibles.
Peso con cable	20,6 kg (45,4 lbs)
Rango de temperatura de funcionamiento	0-50°C
Humedad	90%RH (sin condensación)

Caja de control

Especificaciones

Clasificación IP	IP44
Clase ISO 14644-1 Sala limpia	6
Rango de temperatura de funcionamiento	0-50°C
Humedad	90%RH (sin condensación)
Puertos de E/S	
Entradas digitales	16
Salidas digitales	16
Entradas analógicas	2
Salidas analógicas	2
Entradas digitales en cuadratura	4
E/S de fuente de alimentación	24V 2A
Comunicación	Frecuencia de control 500 Hz
	Modbus TCP
	PROFINET
	Ethernet/IP
	USB 2.0, USB 3.0
Fuente de alimentación	100-240 VAC, 47-440 Hz

Características físicas

Tamaño de la caja de control (ancho x alto x profundo)	460 mm x 449 mm x 254 mm (18,2 in x 17,6 in x 10 in)
Peso	12 kg (26,5 lbs)
Materiales	Acero pulverizado

Caja de control también disponible en versión OEM.

Consola de programación

Especificaciones

Clasificación IP	IP54
Humedad	90%RH (sin condensación)
Resolución de pantalla	1280 x 800 píxeles

Características físicas

Materiales	Plástico
Peso	1,6 kg (3,5 lbs) con 1 m de cable TP incluido
Longitud del cable	4,5 m (177,17 in)

Consola de programación también disponible con opción 3PE.

Contacto

Universal Robots España
Avenida Tarragona 106
08015 Barcelona
+34 933 158 076

universal-robots.com/es
sales.es@universal-robots.com

Figura 8.1: Anexo A: Ficha técnica UR5e

	Bone Name	Sequence In Data Block
Body	Hips	0
	RightUpLeg	1
	RightLeg	2
	RightFoot	3
	LeftUpLeg	4
	LeftLeg	5
	LeftFoot	6
	Spine	7
	Spine1	8
	Spine2	9
	Neck	10
	Neck1	11
	Head	12
	RightShoulder	13
	RightArm	14
	RightForeArm	15
	RightHand	16
Fingers	RightHandThumb1	17
	RightHandThumb2	18
	RightHandThumb3	19
	RightInHandIndex	20
	RightHandIndex1	21
	RightHandIndex2	22
	RightHandIndex3	23
	RightInHandMiddle	24
	RightHandMiddle1	25
	RightHandMiddle2	26
	RightHandMiddle3	27
	RightInHandRing	28
	RightHandRing1	29
	RightHandRing2	30
	RightHandRing3	31
	RightInHandPinky	32
	RightHandPinky1	33
	RightHandPinky2	34
	RightHandPinky3	35
Body	LeftShoulder	36
	LeftArm	37
	LeftForeArm	38
	LeftHand	39
Fingers	LeftHandThumb1	40
	LeftHandThumb2	41
	LeftHandThumb3	42
	LeftInHandIndex	43
	LeftHandIndex1	44
	LeftHandIndex2	45
	LeftHandIndex3	46
	LeftInHandMiddle	47
	LeftHandMiddle1	48
	LeftHandMiddle2	49
	LeftHandMiddle3	50
	LeftInHandRing	51
	LeftHandRing1	52
	LeftHandRing2	53
	LeftHandRing3	54
	LeftInHandPinky	55
	LeftHandPinky1	56
	LeftHandPinky2	57
	LeftHandPinky3	58

Figura 8.2: Anexo B: Secuencia de datos esqueleto en matriz

Bibliografía

[1] IFR. ¿why service robots are booming worldwide? https://ifr.org/ifr-press-releases/news/why-service-robots-are-booming-worldwide.

[2] Bill Gates. A robot in every home. https://www.scientificamerican.com/article/a-robot-in-every-home-2008-02/.

[3] Thomas B. Sheridan. Human–robot interaction: Status and challenges. https://journals.sagepub.com/doi/full/10.1177/0018720816644364/.

[4] TeleoperaciÓn: Sistemas teleoperados. http://catarina.udlap.mx/u_dl_a/tales/documentos/lmt/padilla_m_o/capitulo2.pdf.

[5] Reporte de trabajo de investigación tutelado: Teleoperación de robots. https://upcommons.upc.edu/bitstream/handle/2117/570/IOC-DT-P-2004-05.pdf.

[6] TUE. A survey on teleoperation. https://pure.tue.nl/ws/portalfiles/portal/4419568/656592.pdf.

[7] Teleoperación: técnicas, aplicaciones, entorno sensorial y teleoperación inteligente. https://upcommons.upc.edu/bitstream/handle/2117/570/IOC-DT-P-2004-05.pdf.

[8] Cirugía robótica. https://es.wikipedia.org/wiki/Cirugia_robotica.

[9] Zeus robotic surgical system. https://en.wikipedia.org/wiki/ZEUS_robotic_surgical_system.

[10] Fundamentos de la robótica. https://eltrasteroloco.files.wordpress.com/2017/03/267380685-fundamentos-de-robotica.pdf.

[11] Unimation. https://en.wikipedia.org/wiki/Unimation.

[12] Enciclopedia Británica. Robot. https://www.britannica.com/technology/robot-technology.

[13] Diccionario Merriam Webster. Robot. https://www.merriam-webster.com/dictionary/robot.

[14] Diccionario Español. Robot. https://dle.rae.es/robot?m=form.

[15] Validity of the perception neuron inertial motion capture system for upper body motion analysis. https://www.sciencedirect.com/science/article/abs/pii/S0263224119308905.

[16] Funcionamiento de la captura de movimiento. https://teseo.es/noticias/que-es-y-como-funciona-la-captura-de-movimiento/.

[17] Axis neuron. https://neuronmocap.com/pages/axis-neuron.

[18] Manual de polyscope. https://s3-eu-west-1.amazonaws.com/ur-support-site/16267/Software_Manual_es_Global.pdf.

[19] neurondatareader. https://shopcdn.noitom.com.cn/newimage/0c51544770fb49d79f814e9446875121.pdf.

[20] Robot operating system (ros). https://www.aer-automation.com/wp-content/uploads/2022/03/ROS_articuloAER.pdf.

[21] Ur$_c$*ontrol*..

[22] Ros2 humble. https://docs.ros.org/en/humble/index.html.

[23] Instalación de ros2 humble para windows. https://docs.ros.org/en/humble/Installation/Windows-Install-Binary.html, .

[24] Chocolatey. https://chocolatey.org/install.

[25] Instalación de ros2 humble para ubuntu. https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html, .

[26] Jose Manuel Illera Rodríguez with smartgraph$_{.\,Publicacióneninstagram.}$.