Universidad de Málaga

Escuela de Ingenierías Industriales

Departamento de Ingeniería de Sistemas y Automática

Trabajo Fin de Grado

Desarrollo de una interfaz software para el control de un manipulador móvil con ruedas omnidireccionales

Grado en Ingeniería en Tecnologías Industriales

Autor: Rodrigo Castro Ochoa

Tutor: Juan Manuel Gandarias Palacios
Cotutor: Jesús Manuel Gómez de Gabriel

30 de noviembre de 2024

Declaración de Originalidad del Trabajo Fin de Grado

D./Dña. Rodrigo Castro Ochoa
DNI/Pasaporte: 26968110N. Correo electrónico: rcastro@uma.es
Titulación: Grado en Ingeniería en Tecnologías Industriales
Título del Proyecto/Trabajo: Desarrollo de una interfaz software para el control de un manipulador móvil con ruedas omnidireccionales

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin.

Asimismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a 30 de noviembre de 2024

Fdo.: Rodrigo Castro Ochoa

Resumen

Desarrollo de una interfaz software para el control de un manipulador móvil con ruedas omnidireccionales

Autor: Rodrigo Castro Ochoa
Tutor: Juan Manuel Gandarias Palacios
Cotutor: Jesús Manuel Gómez de Gabriel
Departamento: Departamento de Ingeniería de Sistemas y Automática
Titulación: Grado en Ingeniería en Tecnologías Industriales
Palabras clave Robótica; Manipulador Móvil; ROS; Teleoperación

Este proyecto consiste en el desarrollo de una interfaz de control y teleoperación para un manipulador móvil omnidireccional, utilizando la plataforma robótica RAFI, previamente desarrollada en el Departamento de Ingeniería de Sistemas y Automática. El sistema combina la locomoción de una base omnidireccional con la manipulación de un brazo robótico colaborativo de 7 grados de libertad. Aunque ambos robots, base y manipulador, operan de manera independiente, el objetivo de este trabajo es lograr que funcionen de forma simultánea.

Para ello, se emplea ROS, un marco de trabajo diseñado para facilitar el desarrollo de sistemas robóticos, permitiendo la integración de actuadores, sensores y sistemas de control. El proyecto se enfoca en la implementación de un sistema distribuido, la adaptación de los controladores de la base y el manipulador, y el diseño de una interfaz de teleoperación mediante un mando de videojuegos. Este mando permite al usuario enviar comandos a los controladores en tiempo real.

Los experimentos realizados validan la efectividad de la interfaz, logrando una operación eficiente de ambos robots. Las contribuciones más relevantes incluyen el control simultáneo de la base y el manipulador, y la implementación de un sistema de teleoperación adaptable a futuros desarrollos.

Abstract

Development of a software interface for the control of a mobile manipulator with omnidirectional wheels

Author: Rodrigo Castro Ochoa

Supervisor: Juan Manuel Gandarias Palacios

Cosupervisor: Jesús Manuel Gómez de Gabriel

Departamemt: Departamento de Ingeniería de Sistemas y Automática

Degree: Grado en Ingeniería en Tecnologías Industriales

Keywords: Robótica; Manipulador Móvil; ROS; Teleoperación

This project involves developing a control and teleoperation interface for an omnidirectional mobile manipulator, using the RAFI robotic platform, previously developed in the Department of Systems and Automatic Engineering. The system combines the locomotion of an omnidirectional base with the manipulation capabilities of a collaborative robotic arm with 7 degrees of freedom. Although both robots, the base and the manipulator, operate independently, the goal of this work is to have them function simultaneously.

To achieve this, ROS, a framework designed to facilitate the development of robotic systems, is used, allowing the integration of actuators, sensors, and control systems. The project focuses on the implementation of a distributed system, the adaptation of the base and manipulator controllers, and the design of a teleoperation interface using a video game controller. This controller allows the user to send commands to the robot controllers in real time.

The experiments conducted validate the effectiveness of the interface, achieving efficient operation of both robots. The most relevant contributions include the simultaneous control of the base and manipulator, as well as the implementation of a teleoperation system adaptable to future developments.

A mis padres y a mi hermana,
por su apoyo

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a mis tutores, Juanma y Jesús, por brindarme la oportunidad de llevar a cabo este Trabajo de Fin de Grado, el cual me ha permitido desarrollarme académicamente. Gracias por vuestra orientación a lo largo de todo el proceso. También agradezco a todas las personas que han trabajado anteriormente en la plataforma robótica RAFI, cuyo esfuerzo ha sentado las bases para este proyecto. A mis padres, les doy las gracias por mantener su confianza en mí incluso cuando yo mismo dudaba de mis capacidades. A mi hermana, le agradezco su constante cariño y apoyo. Asimismo, quiero agradecer a mis amigos por acompañarme en este camino. Finalmente, quiero expresar mi gratitud al personal de la Escuela de Ingenierías Industriales por facilitarme acceso al laboratorio. A todos vosotros, gracias.

Índice

Índice de Figuras

Figura **Página**

Índice de Tablas

Tabla **Página**

CAPÍTULO 1

Introducción

Contenido

En este capítulo de introducción se explica la motivación del proyecto, así como el estado del arte relativo a manipuladores, plataformas móviles e interfaces de teleoperación. Además, se tratarán los objetivos que se buscan cubrir durante la ejecución de este proyecto. Finalmente, se expondrá la estructura de esta memoria.

1.1. Motivación

La robótica ha experimentado una gran evolución en las últimas décadas impulsada por las necesidades humanas. En sus inicios, los robots sustituyeron a los humanos en tareas peligrosas. Posteriormente, se adaptaron a una industria que demandaba mayor flexibilidad e inteligencia. En la actualidad, la robótica se está adaptando a las nuevas necesidades humanas, especializándose en robótica de campo y de servicio. La robótica de campo se enfrenta a entornos más dinámicos, como agricultura, minería o exploración subacuática, requiriendo mayor capacidad de adaptación y movilidad. La robótica de servicio se centra en robots diseñados para interactuar directamente con los humanos y ayudar en tareas cotidianas como asistencia en el hogar, atención médica o servicios públicos. Las nuevas aplicaciones de la robótica implican desafíos en la interacción humano-robot. [1]

En este contexto, los manipuladores móviles han cobrado especial relevancia. Estos sistemas combinan la movilidad de una base robótica con un brazo manipulador para realizar tareas complejas en distintos entornos, lo que les permite operar eficazmente en entornos no estructurados. Sin embargo, supone un reto la integración de la movilidad y la manipulación. Así como la interacción con otros robots y humanos. [2]

La interacción humano-robot (Human-Robot Interaction, HRI) ha ganado especial importancia en sectores como la manufactura, atención médica y robótica de servicio. HRI se puede dividir en cuatro áreas: supervisión de robots en tareas rutinarias, teleoperación en entornos peligrosos, vehículos automatizados e interacción social entre robots y humanos [3]. Cada una de estas áreas plantea desafíos como la capacidad de los robots de interpretar y predecir las intenciones humanas; y la necesidad de desarrollar algoritmos que permitan una colaboración fluida y segura entre ambos actores. Estos desafíos son clave en entornos compartidos con personas, donde la capacidad de adaptación y percepción del entorno es fundamental para garantizar la eficiencia y seguridad.

Uno de los campos más avanzados en HRI es el de la colaboración humano-robot en entornos industriales, donde los manipuladores colaborativos responden a la necesidad de trabajar de manera segura junto a los humanos. Los robots pueden realizar tareas repetitivas y peligrosas mientras que los humanos supervisan y aportan capacidad y resolución de problemas [4]. Para garantizar la seguridad, los manipuladores colaborativos constan de sistemas de control que limitan su fuerza en caso de contacto accidental con un operador.

Para garantizar una colaboración exitosa entre humanos y robots se deben diseñar interfaces intuitivas que permitan a los usuarios interactuar y supervisar a los robots de manera intuitiva [4]. Estas interfaces deben permitir el intercambio de información entre el robot y el humano.

La motivación principal de este trabajo es avanzar en la integración de la movilidad y manipulación en un único sistema robótico controlable. Para ello es necesario el desarrollo de una interfaz que permita una comunicación adecuada entre los tres actores involucrados: plataforma móvil, manipulador colaborativo y operador humano. Esto mejorará el uso de robots en entornos reales donde se requiere adaptabilidad, precisión y seguridad.

1.2. Estado del arte

1.2.1. Manipuladores colaborativos

Los robots colaborativos, también llamados *cobots*, están diseñados para trabajar de manera eficiente y segura junto a humanos en entornos industriales. Entre sus aplicaciones se encuentra la manipulación, empaquetado, paletizado, soldadura, ensamblaje y gestión de materiales peligrosos. Esto permite sustituir al operario en tareas repetitivas reduciendo el riesgo de errores asociados al cansancio o distracción [5].

Los riesgos intrínsecos de la colaboración entre manipuladores y humanos exigen unas medidas estrictas para garantizar la seguridad de los últimos. Para ello, los modos de colaboración están regulados en cuatro niveles por la norma ISO 10218-1/2, tal y como recoge [6]. La figura 1.1 muestra los cuatro modos de colaboración:

1. Parada supervisada de seguridad (SMS): Existe un área colaborativa conjunta donde trabajan el robot y el humano pero no de manera simultánea. Para que el robot inicie el movimiento, el humano debe abandonar el área colaborativa. Este tipo de cooperación es ideal para colocación manual de objetos o inspección visual. SMS requiere un interruptor que deje sin electricidad a los actuadores después de que el movimiento haya terminado. Cuando el operario entra en el área colaborativa, el robot entra en un modo de parada segura y cuando el operario la abandona, este continua con su ciclo de operación.

2. Guía manual (GM): El operador modifica la configuración articular del robot mientras éste contrarresta su propio peso para no modificar su posición. El operador se encuentra en contacto directo con el robot. Este modo de colaboración requiere que el robot conste de parada supervisada de seguridad y velocidad.

3. Monitorización de separación y velocidad (SSM): Permite la presencia del operador dentro del espacio del robot a través de sensores de monitorización de presencia. El robot opera a máxima velocidad y reduce su velocidad progresivamente a medida que el operador invade su espacio de trabajo.

4. Limitación de potencia y fuerza (PFL): Permite al operador trabajar invadiendo el espacio de trabajo del robot gracias al uso de modelos de control para gestionar las colisiones entre el robot y el operador.

La figura 1.2 muestra dos manipuladores colaborativos. La serie UR (UR3, UR5 UR10) se trata de diferentes robot desarrollados por Universal Robotics. La figura 1.2a muestra el UR10e, un manipulador de 6 articulaciones de revolución que soporta una carga útil de 12.5 kg, indicado para tareas relativamente pesadas [7]. La figura 1.2b muestra el Kuka LBR iiwa 14 R820, otro manipulador relevante con 7 articulaciones de rotación especialmente diseñado para tareas de investigación ya

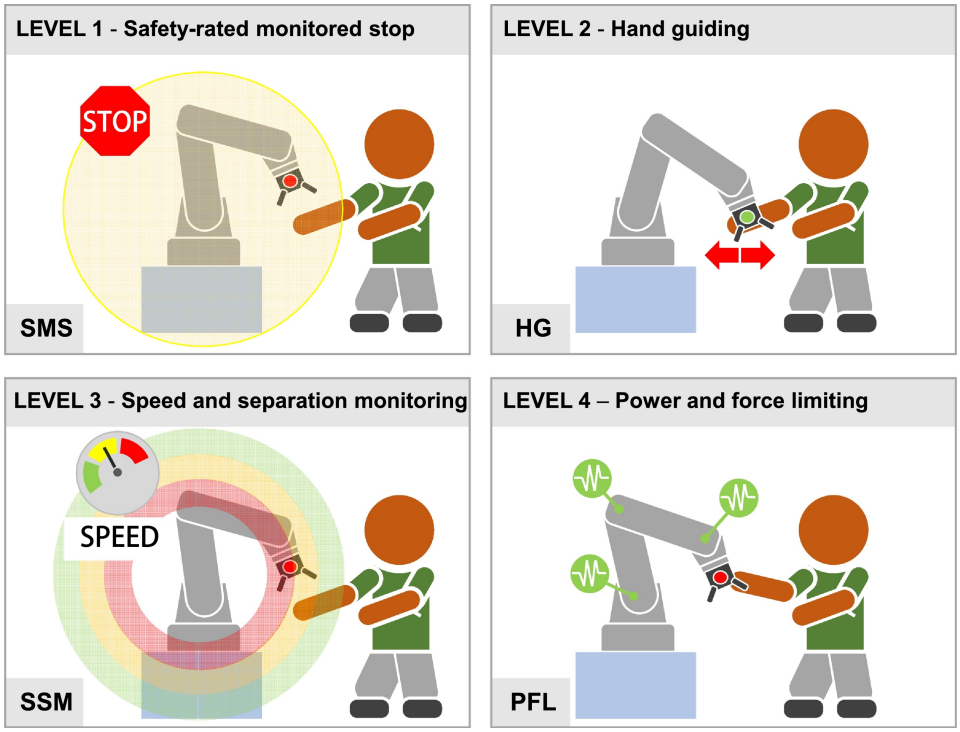


Figura 1.1: Cuatro modos de colaboración identificados según la norma ISO 10218-1/2. Fuente: [6]

que es accesible a través de una interfaz de tiempo real llamada "Fast Robot Interface". Soporta una carga útil de 14 kg [8].

1.2.2. Manipuladores móviles

Los manipuladores móviles se fundamentan en la locomanipulación, combinando la locomoción de las plataformas robóticas con la manipulación de los brazos robóticos, permitiendo realizar tareas complejas y precisas. Durante la década de 1980, los investigadores comenzaron a investigar sobre los manipuladores móviles para mejorar la capacidad de los robots de interactuar con su entorno más allá de un espacio fijo.

1.2.2.1. Entornos estructurados

Un entorno estructurado es aquel donde el robot opera en un escenario predecible y controlado. El robot Shakey, desarrollado entre 1966 y 1972 por el Standford Research Institute, fue uno de los primeros desarrollos de una plataforma móvil dotada de sensores y algoritmos de planificación. Usaba su propio cuerpo para interactuar con el entorno, por ejemplo, empujando cajas. Su sistema

[1]Universal Robots, *UR10e*, URL: https://www.universal-robots.com/products/ur10-robot/, Fecha de acceso: 10 de septiembre de 2024.

[2]KUKA, *LBR iiwa*, URL: https://www.kuka.com/es-es/productos-servicios/sistemas-de-robot/robot-industrial/lbr-iiwa, Fecha de acceso: 10 de septiembre de 2024.

(a) UR10e. Fuente: [1]

(b) Kuka LBR iiwa 14 R820. Fuente: [2]

Figura 1.2: Ejemplos de manipuladores colaborativos usados en la industria

Figura 1.3: El robot Shakey desarrollado por SRI entre 1966 y 1972. Fuente: [3]

estaba organizado en cinco niveles de jerarquía, siendo los niveles más bajos los encargados de acciones de bajo nivel como el movimiento y los más altos los encargados de la planificación y supervisión de tareas complejas [9].

Futuros desarrollos dotaron a las plataformas móviles de manipuladores para mejorar su interacción con el entorno. La figura 1.4 muestra un prototipo de del robot PR1. Se trata de un sistema móvil equipado con dos manipuladores como brazos y 25 GDL desarrollado en 2008. Consta de una carga útil de 5 kg por brazo y articulaciones de baja impedancia mecánica. El PR1 fue diseñado para realizar tareas domésticas y de asistencia. Cada brazo tiene 7 GDL. Este robot utiliza una interfaz

[3] IEEE Spectrum, *Shakey the robot*, URL: https://spectrum.ieee.org/sri-shakey-robot-honored-as-ieee-milestone, Fecha de acceso: 9 de septiembre de 2024.

Figura 1.4: Imagen del prototipo de PR1. Fuente: [10]

de teleoperación para demostrar la funcionalidad del hardware, permitiendo su control desde una estación de trabajo compuesta por dos dispositivos de entrada: Phantom 3.0 y un pedal [10].

La figura 1.5 muestra el Phantom 3.0, un dispositivo háptico de 6 GDL desarrollado por "SenAble Technologies". Proporciona retroalimentación táctil y permite la interacción precisa en entornos virtuales. Consta de un lápiz o un dedal que permite al usuario apreciar la estructura de objetos 3D [11]. Aplicado al robot, traduce los movimientos de la mano del operador en comandos en tiempo real mientras que replica las fuerzas que experimenta éste mediante la retroalimentación háptica.

Varios años más tarde, apareció el sucesor del PR1. El PR2 se trata de un sistema robótico desarrollado por "Willow Garage". Consta de una plataforma móvil y dos brazos manipuladores de 7 GDL. Además, cuenta con sensores avanzados como cámaras estéreo y escáneres láser. Una de sus características más relevantes es su control y programación con ROS. ROS es un marco de software flexible, desarrollado por la misma empresa, que permite la integración de sensores, actuadores y sistemas de control de forma sencilla. La relación entre PR2 y ROS fue simbiótica: mientras que el PR2 fue diseñado para facilitar el desarrollo de software robótico, ROS fue creado para satisfacer las necesidades de un manipulador móvil. Versiones tempranas de ROS fueron testadas y refinadas usando este robot. Willow Garage fomentó la creación de una comunidad de colaboración e intercambio de conocimiento gracias a la donación de 11 unidades del PR2 a diferentes instituciones de investigación en todo el mundo[13]. ROS es un software de código abierto, lo que facilita su uso por investigadores y empresas, convirtiéndose en un estándar para la creación y programación de robots.

El PR2 ha sido servido como plataforma de investigación de diversas aplicaciones robóticas. Uno

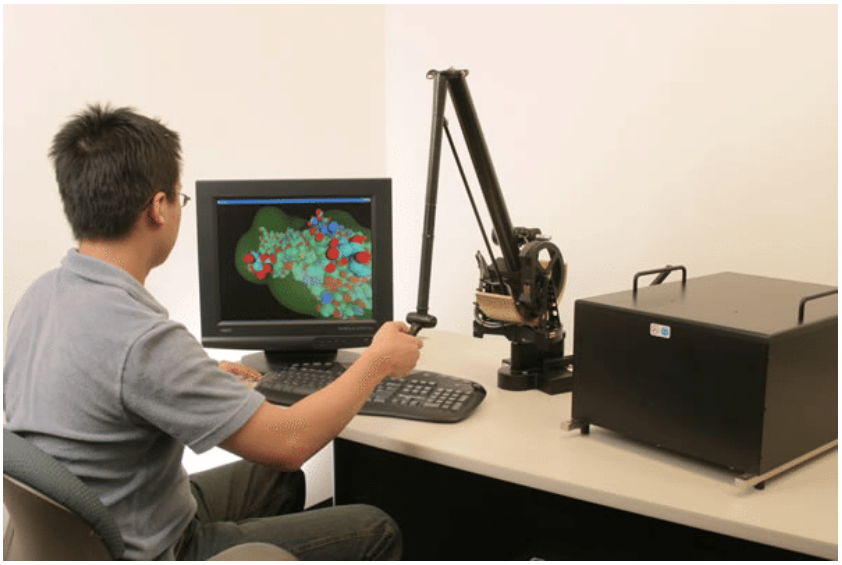


Figura 1.5: Dispositivo háptico "Phantom Premium 3.0" con 6 GDL desarrollado por SensAble. Fuente: [12]

Figura 1.6: El robot PR2 cogiendo una bebida de una nevera. Fuente: [14]

de los proyectos que han usado este robot ha sido el encargado de desarrollar una aplicación para buscar y servir bebidas de manera autónoma por [14]. La figura 1.6 muestra el robot cogiendo una bebida de una nevera. Para planificación y ejecución de tareas, se propone el uso de SMACH, una librería de Python encargada de diseñar e implementar máquinas de estados finitos (FMS) en ROS. Esto permite controlar comportamientos complejos que requieren múltiples estados y transiciones entre ellos.

Figura 1.7: Fotografía del primer prototipo del robot Centauro desarrollado por el IIT en 2018. Fuente: [15]

1.2.2.2. Entornos no estructurados

En entornos no estructurados, los manipuladores móviles deben enfrentarse a terrenos impredecibles y obstáculos. Los robots con patas presentan una solución para este tipo de entornos. Al utilizar puntos de apoyo aislados, el robot puede mantener el equilibrio independientemente de la irregularidad del terreno. El número de apoyos es importante: Los robots bípedos ofrecen mayor flexibilidad y movilidad pero su equilibrio está comprometido ante perturbaciones externas. Los robots con múltiples apoyos, como los cuadrúpedos o hexápodos demuestran mayor estabilidad y robustez [15].

En este contexto surge el robot Centauro, desarrollado en 2018 por el "Italian Institute of Technology" (ITT). La figura 1.7 muestra el primer prototipo de este robot. La parte superior tiene forma antropomórfica y consta de dos brazos manipuladores. La parte inferior consta de cuatro piernas con ruedas. En conjunto, el robot consta de unas dimensiones correspondientes a un humano medio alcanzando una altura máxima de 1.70 m [15].

1.2.3. Evolución de las interfaces de teleoperación

La teleoperación manual sigue siendo clave para el control de robots en situaciones donde no es viable la autonomía total. Aunque la IA se encuentra en constante avance, en entornos no estructurados o con obstáculos sigue siendo esencial la intervención humana gracias a su capacidad de resolución de problemas.

La figura 1.8 muestra un diagrama de decisión del operador sobre cuando es necesario pasar el control a un operador humano. El operador elige un modo de control: control autónomo, control semi autónomo o teleoperación manual. Cuando el control autónomo se encuentra con problemas

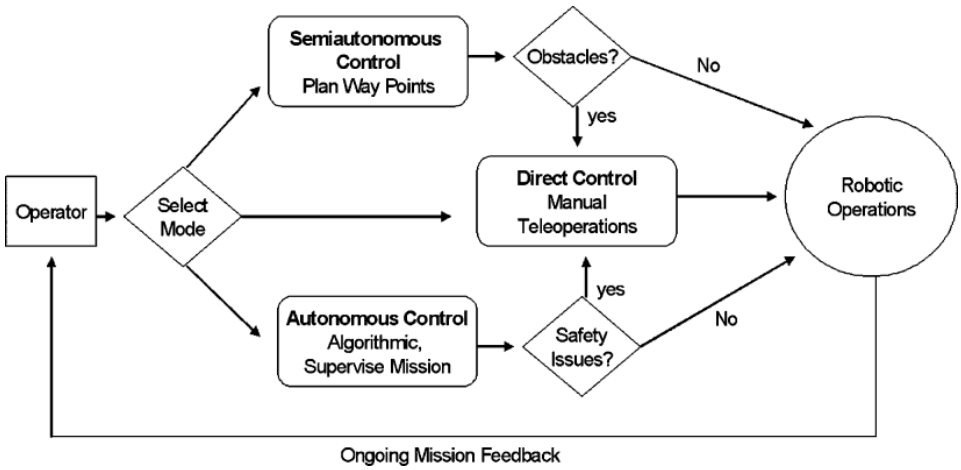


Figura 1.8: Diagrama de decisión durante el control de un robot con habilidades autónomas, semi autónomas y teleoperación manual. Fuente: [16]

de seguridad o el control semi autónomo se encuentra con un obstáculo, delegan el control a la teleoperación manual [16].

Los principales desafíos de la teleoperación están relacionados con la naturaleza humana del operador y la manipulación remota. El rendimiento del operador está condicionado por sus habilidades motoras y su capacidad para construir un modelo mental del entorno con información limitada. La estimación de distancias y detección de obstáculos puede ser difícil. La manipulación remota está limitada por la calidad de los sistemas de entrada y salida tales como campo de visión limitado, incapacidad de inferir profundidad, degradación de la imagen o retrasos en entrada y salida. El uso de interfaces de teleoperación que integren diferentes indicadores y controladores multi modales es fundamental para garantizar una comunicación efectiva. Existen indicadores visuales, de audio y hápticos, que informan a través del sentido del tacto ya sea con presión o vibración. Respecto a los controladores, se encuentra el control por voz o gestos, como joysticks o seguidores oculares [16].

El hardware de teleoperación debe adecuarse al tipo de robot. Se ha desarrollado un control para un robot antropomórfico mediante un sensor de profundidad Kinect. Este sensor es capaz de identificar hasta 20 articulaciones del cuerpo humano, que posteriormente, la interfaz de control se encarga de transformar en comandos para el robot [17]. La figura 1.9 muestra la teleoperación de otro robot mediante reconocimiento de gestos de la mano. Una cámara detecta la figura que forma la mano y comanda el robot en consecuencia [18].

La figura 1.10 muestra el dispositivo de teleoperación híbrido propuesto por [19]. Se trata de un joystick capaz de controlar 2 GDL y simular la inclinación del entorno que soporta el robot. Además, el operador puede reconocer la diferencia entre su comando y el movimiento real del robot mediante la acción de una articulación de rotación. En otras palabras, el operador es capaz de percibir un error en la trayectoria por el movimiento de la articulación inferior.

El estudio [20] propone el uso de realidad virtual para garantizar una experiencia inmersiva durante la teleoperación remota por parte del operador. Para ello, el robot captura los datos RGB-D

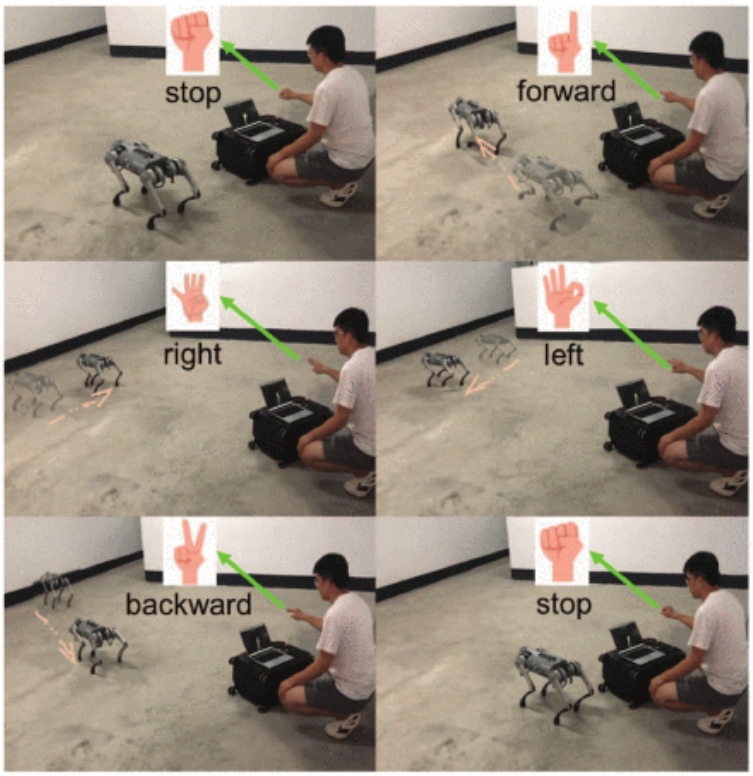


Figura 1.9: Teleoperación de un robot mediante el uso de una cámara que lee e interpreta los gestos realizados por el usuario

del entorno. Posteriormente, se reconstruye el entorno mediante un modelo 3D y se transmite al usuario mediante un casco de realidad virtual. Esto permite una exploración en tercera persona, de manera independiente de la vista y posición actual del robot.

1.3. Objetivos y contribución

La contribución de este proyecto es dotar de una interfaz de software común para el control y teleoperación de los dos robots que conforman RAFI. El desarrollo de esta solución y su posterior análisis será descrito en esta memoria.

Los objetivos parciales de este trabajo son:

1. Establecer una comunicación efectiva entre todos los componentes del sistema mediante el desarrollo e implementación de un sistema distribuido. El sistema distribuido permitirá la interconexión fluida entre los distintos robots que componen RAFI, garantizando que

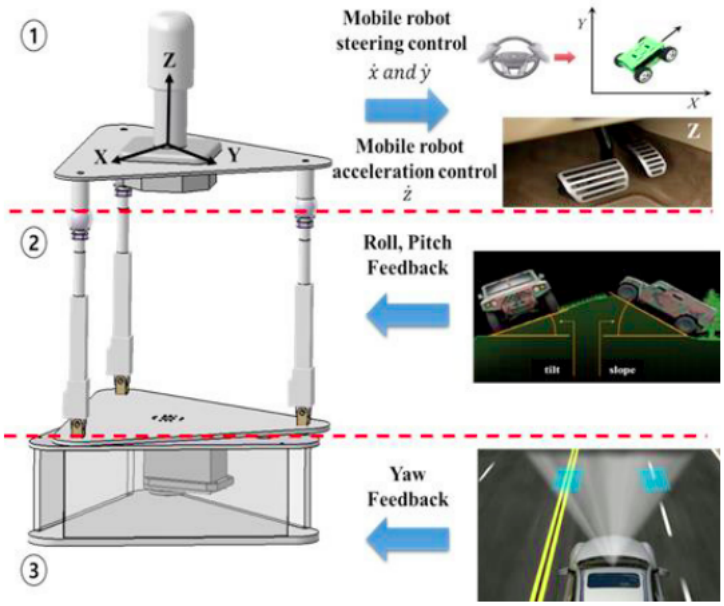


Figura 1.10: Dispositivo híbrido de teleoperación capaz de controlar y mostrar una respuesta háptica al entorno del robot. El primer nivel muestra el joystick de 2 GDL. El segundo nivel muestra tres articulaciones prismáticas capaz de modificar la inclinación del plano sobre el que se sustenta el joystick. El tercer nivel muestra una articulación de revolución capaz de reflejar el error en la trayectoria del robot mediante giro. Fuente: [19]

tanto la base móvil como el manipulador puedan ser operados simultáneamente o de forma independiente.

2. Adaptar el software desarrollado en anteriores trabajos para la plataforma RAFI, enfocado en controlar y teleoperar la base. Se realizarán modificaciones y mejoras que permitan integrar este código con la nueva estructura del sistema distribuido. Se desea asegurar que los comandos de teleoperación de la base sean precisos, respondiendo de manera efectiva a las entradas del usuario.

3. Implementar herramientas para la gestión del manipulador mediante terminal tales como el desbloqueo de los frenos y la activación del modo FCI. Estas herramientas permitirán preparar el manipulador para la teleoperación, asegurando que esté listo para ejecutar comandos externos de manera segura y eficiente.

4. Adaptar el controlador de impedancia cartesiana en el manipulador para que pueda ser usado por la interfaz de teleoperación.

5. Adaptar el controlador de velocidad cartesiana en el manipulador. El controlador de ejemplo proporcionado por el fabricante consiste en la ejecución de un bucle de movimientos. Se pretende adaptar el controlador para que acepte como entrada una velocidad cartesiana.

6. Desarrollo de la interfaz de software encargada del control simultáneo de ambos robots.

7. Desarrollo de una interfaz de teleoperación simultánea para ambos robots. La interfaz de teleoperación permitirá que el usuario pueda controlar los movimientos de ambos robots en tiempo real, asegurando que las acciones se coordinen de manera fluida y sin conflictos.

1.4. Estructura de la memoria

Esta memoria tiene la siguiente estructura: El capítulo 2 describe el robot RAFI, diferenciando entre la base móvil y el manipulador colaborativo. También se explican las herramientas de software utilizadas y el trabajo previo realizado sobre el robot. El capítulo 3 explica la manera en la que se ha llegado a la solución, detallando la implementación del sistema distribuido para garantizar la comunicación, el sistema de control del robot y la interfaz de teleoperación. El capítulo 4 muestra y analiza el resultado de la solución aportada. Por último, el capítulo 5 realiza una valoración final del proyecto en función de los objetivos y resultados obtenidos. Además, se proponen futuras líneas de trabajo a partir de este trabajo.

CAPÍTULO 2

Contexto y Marco Teórico

Contenido

En este capítulo se explicará en que consiste la plataforma robótica RAFI y las partes de las que se compone. Después, se desglosarán las herramientas de software utilizadas. Por último, se describirá el trabajo previo que sentó las bases de establecer una interfaz de software para controlar y teleoperar el robot.

2.1. Robot de Asistencia Física Inteligente (RAFI)

RAFI se compone de un brazo colaborativo comercial y una plataforma móvil diseñada y fabricada durante el desarrollo de varios proyectos de investigación por profesores del departamento de "Ingeniería de Sistemas y Automática" de la Escuela de Ingenierías Industriales de la Universidad de Málaga.

Es el resultado de un proyecto de investigación para el diseño de un manipulador móvil para asistencia física (RAFI) en el que han contribuido una serie de Trabajos de Fin de Estudios. La última aportación fue Francisco Carbonero que trabajó en su TFG en la implementación de una versión previa del controlador de la base y su teleoperación [21]. Su trabajo se encuentra descrito en la sección 2.3.

2.1.1. Base omnidireccional

La base del manipulador consiste en una plataforma móvil omnidireccional de acero. Sus dimensiones son 50 × 64 × 50 cm. Monta cuatro ruedas mecanum [1] convirtiendo la base en un sistema holonómico.

Un sistema holonómico es un sistema robótico cuya cantidad de grados de libertad coincide con el número de variables de control independientes que tiene. Esto significa que el sistema puede moverse en cualquier eje posible dentro de su espacio de trabajo sin restricciones, logrando movimientos omnidireccionales, ya que puede desplazarse lateralmente, frontalmente y girar sin necesidad de realizar maniobras complejas.

La base dispone de dos placas dobles controladoras de velocidad para los motores. Se tratan del modelo RoboClaw de la marca BasicMicro. Cada placa controla dos motores. La figura 2.1 muestra un esquema del conexionado de las ruedas con sus respectivas placas.

Respecto a la batería, se tratan de baterías de fosfato de hierro y litio (LiFePO4) con capacidad de 480 Ah a 24 V, lo que equivale a 1536 Wh, con una tensión nominal por celda de 3,2 V. La batería consta del sistema de gestión *123SmartBMS gen3*. Este sistema permite la monitorización en tiempo real de la carga y descarga a través de la aplicación móvil *123SmartBMS*. Esta aplicación usa bluetooth para comunicarse con el sistema.

La base cuenta con un pulsador de emergencia que desconecta la alimentación del sistema. También incorpora una pantalla táctil. El objetivo original de la pantalla era mostrar la salida HDMI de una Raspberry Pi 4 que tiene acoplada por la parte trasera. Esta placa actuaría como cerebro de la base, sin embargo, debido a su bajo rendimiento, se sustituyó por un PC a bordo, en adelante, PC integrado que se ubica dentro de la base. En la figura 2.2 se muestran etiquetados los principales componentes de la plataforma robótica.

[1] Una rueda mecanum es un tipo especial de rueda que permite movimiento omnidireccional a vehículos o robots. Está compuesta por una serie de rodillos angulados que, al combinar la rotación de varias ruedas mecanum, permiten que un robot pueda moverse en cualquier dirección: hacia delante, atrás, lateralmente o en diagonales, sin necesidad de cambiar la orientación del robot. Esto confiere una alta maniobrabilidad a los robots que las incorporan.

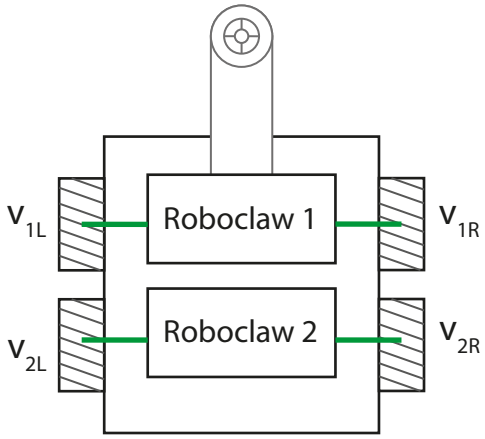
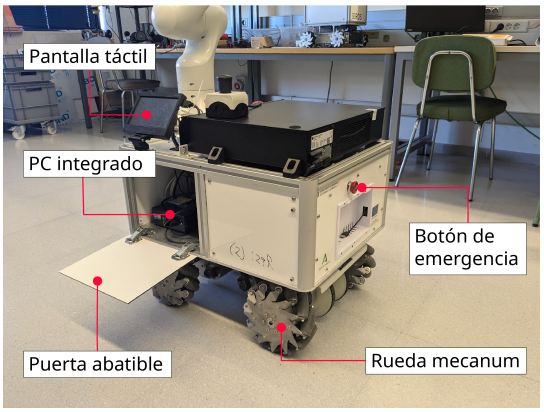
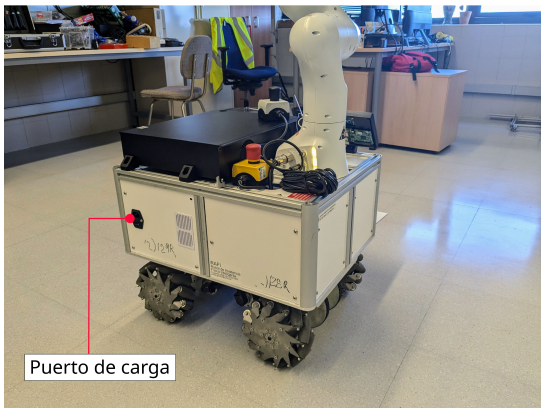


Figura 2.1: Esquema simplificado del conexionado entre los motores y las placas Roboclaw. Las ruedas delanteras están conectadas a la placa 1 mientras que las ruedas traseras están conectadas a la placa 2



(a) Perspectiva trasera-izquierda



(b) Perspectiva delantera-derecha

Figura 2.2: Imágenes de la plataforma robótica con sus componentes etiquetados

2.1.2. Manipulador Franka Emika Panda

El manipulador Franka Emika Panda, renombrado posteriormente como Franka Emika Robot (FER) por parte del fabricante, se trata de un manipulador colaborativo redundante. Esta diseñado para modificar su impedancia mecánica en situaciones que requieren una interacción segura con el entorno. Consta de 7 GDL, uno más que un punto en el espacio, lo que le permite un alto grado de maniobrabilidad ya que es capaz de alcanzar una posición y orientación determinada del efector final con distintas configuraciones articulares. Está desarrollado y producido por la empresa alemana "Franka Emika GmbH".

La figura 2.3 muestra etiquetados los distintos componentes que forman el manipulador. Estos son:

- Brazo: Brazo con siete articulaciones de revolución. Consta de sensores de par en cada una de

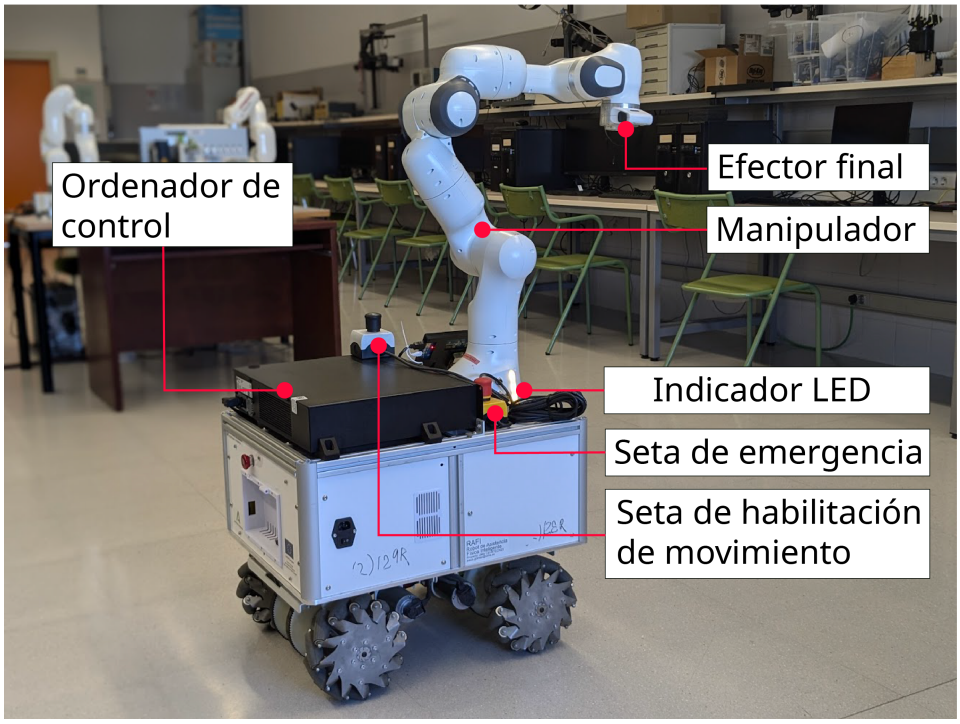


Figura 2.3: Imagen del manipulador Franka Emika Panda con etiquetas de las partes más relevantes

sus articulaciones. De forma opcional, se le puede acoplar una pinza o mano robótica en el efector final, aunque no se ha considerado para el desarrollo de este proyecto.

- Control: Ordenador de control del manipulador.
- Emergency stop device: Dispositivo de emergencia encargado de cortar la alimentación del brazo y ordenador de control a petición del operario.
- External enabling device: Dispositivo encargado de habilitar el movimiento automático del brazo.
- Indicador LED: Informa del estado del manipulador en todo momento. La tabla 2.1 describe los distintos estados que puede tomar el LED.

El Franka Emika Panda consta de diferentes modos de control:

- **Desk**: Proporciona una interfaz gráfica de programación desde el navegador web.
- **Pilot**: Permite modificar la configuración articular del brazo de forma manual al apretar el botón de hombre muerto ubicado en el efector final.
- **Franka Control Interface (FCI):** Se trata de un control de bajo nivel con fines de investigación. Permite controlar, de forma externa, el brazo de manera articular o cartesiana. Para ello se

Color	Descripción
blanco	Interacción segura entre el operador y el manipulador
azul	El manipulador está habilitado para moverse
cyan	La tarea se ejecutará después de un tiempo de espera
verde	Movimiento automático del manipulador
amarillo	El manipulador tiene los frenos activos
rosa	Se han recibido señales de habilitación conflictivas
rojo	Ha ocurrido un error

Tabla 2.1: Descripción de los diferentes estados que puede tomar el LED del manipulador

indica el par o configuración articular deseada; o la posición y orientación deseada del efector final.

Para modificar la pose del manipulador, es requisito indispensable desbloquear los frenos. Además, el uso del modo FCI requiere su activación. Ambas cosas se hacen desde el Desk de Franka. Como alternativa, en el apartado 3.1.2.3 se propone un método para la activación del modo FCI y desbloqueo de frenos desde la terminal, mediante un código cliente del Desk.

2.1.3. PC integrado

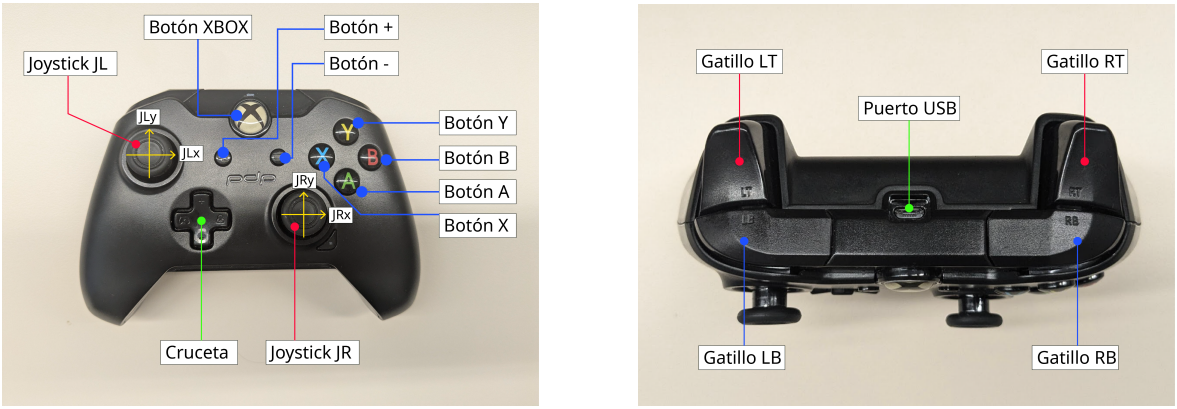
Se trata del PC integrado dentro de la plataforma móvil, su finalidad es ser el núcleo computacional encargado del control de la base y del manipulador. Se trata del modelo AD03 de la marca ACE Magician. Consta de 16GB de memoria RAM y 512 de almacenamiento. Respecto al procesador, monta un Intel Alder Lake N95 de cuatro núcleos. Consta de cinco puertos USB-A y un puerto USB-C. Respecto a las comunicaciones, dispone de dos puertos Ethernet, tarjeta de red inalámbrica y Bluetooth. Respecto a salida de video, tiene un puerto HDMI y un puerto de DisplayPort.

2.1.4. Hardware de teleoperación

Se ha desarrollado una interfaz de teleoperación para demostrar que se puede operar de forma adecuada el robot. Esta interfaz consta de una parte software, que se explica en el apartado 3.3 y una parte hardware que es operada por el usuario. El usuario es capaz de enviar comandos de velocidades o poses deseadas a los controladores mediante un mando. Se ha elegido un mando genérico de XBOX. Se trata del modelo PDP 048-082 y se conecta al PC mediante USB. La figura 2.4 muestra dos perspectivas del mando junto con las etiquetas de los distintos componentes.

El mando está compuesto por botones, joysticks, gatillos y una cruceta:

- Botón: Dispositivo de entrada que actúa mediante pulsación. Su funcionamiento es binario, luego tiene dos estados: presionado (1) o no presionado (0).



(a) Vista frontal

(b) Vista trasera

Figura 2.4: Imágenes del mando utilizado durante la teleoperación.

- Joystick: Es un dispositivo de entrada analógico que permite controlar 2 GDL. Detecta la posición de control en un rango continuo de valores que va desde -1 a 1. En reposo, su salida es 0. Además, los joysticks consta de un botón si se pulsan.

- Gatillos: Son botones ubicados en la parte trasera del mando. Este mando consta de dos tipos de gatillos: analógicos y digitales. Los gatillos analógicos permiten un rango de entrada continuo en un eje entre 0 y 1 según la presión aplicada y controlan 1 GDL. Los gatillos digitales se comportan como botones, registrando únicamente dos estados: presionado (1) o no presionado (0).

- Cruceta: Se trata de un dispositivo de entrada híbrido entre un botón y un joystick. Consta de dos ejes, sin embargo, solo admite 3 valores en cada uno: -1 (dirección negativa), 0 (reposo) y 1 (dirección positiva).

Estos componentes se pueden resumir en dos tipos de actuadores: ejes y botones lógicos. Cada joystick cuenta con dos ejes, la cruceta cuenta con dos ejes y dos gatillos analógicos con un eje cada uno. En total, 8 ejes. El mando consta de 11 botones formados por los dos gatillos digitales, los 2 botones de los joysticks y el resto de botones.

La nomenclatura de los joysticks y gatillos facilita su identificación. Constan de denominaciones compuestas por dos letras. En el caso de los joysticks contienen una J. En los gatillos, contienen una B, de *bottom*, para la fila inferior y una T, de *top* para la fila superior. La otra letra se corresponde con la mano que lo acciona: L ,de *left*, para la mano izquierda y R, de *right*, para la mano derecha. De esta forma, el joystick derecho es JR y el joystick izquierdo es JL. Los gatillos superiores son LT y RT y los gatillos inferiores son LB y RB.

2.2. Herramientas y librerías de software utilizadas

2.2.1. ROS

Robot Operating System, o ROS, es un marco de trabajo compuesto por bibliotecas y herramientas open-source diseñado para facilitar el desarrollo de sistemas robóticos. ROS permite integrar todos los elementos que conforman un robot como actuadores, sensores y sistemas de control de forma sencilla.

La arquitectura de ROS se basa en un modelo distribuido que permite la ejecución de múltiples procesos independientes denominados nodos. Cada nodo se encarga de una función específica dentro del sistema robótico. La comunicación entre nodos no es directa, sino que se realiza por medio de topics. Los topics actúan como canales de comunicación a través de los cuales los nodos pueden publicar o recibir mensajes:

- Publicaciones: Cuando un nodo genera datos para ser compartidos con otros nodos, lo hace publicando esta información en forma de mensaje en un topic especifico. Los nodos interesados en esta información se subscriben a este topic para recibir y procesar los datos en tiempo real.

- Subscripciones: Cuando un nodo necesita acceder a los datos generados por otros nodos se subscriben al topic correspondiente. Al subscribirse, el nodo recibe los mensajes que se publican en ese topic, permitiéndole utilizar la información para sus propias tareas.

ROS incluye herramientas gráficas como rqt_graph, que facilitan la visualización de la estructura del sistema. Los nodos son representados como círculos y los topics como rectángulos. Las flechas indican la dirección de la comunicación: desde los nodos a los topics para las publicaciones y desde los topics a los nodos para las subscripciones. Esta herramienta ayuda a entender las relaciones y flujos de datos entre los distintos nodos y topics contenidos en el sistema.

La comunicación entre nodos en ROS se realiza sobre el protocolo TCP/IP [2], lo que permite que los nodos se comuniquen no solo dentro de un mismo PC, sino también entre PCs conectados a la misma red inalámbrica. Esto aporta flexibilidad en la configuración de sistemas distribuidos.

Los mensajes enviados a través de los topics son datos estructurados que pueden contener información como lecturas de sensores, comandos de actuadores o cualquier otro tipo de dato relevante para el sistema robótico.

ROS soporta varios lenguajes de programación,incluidos Python y C++, facilitando la integración de diferentes componentes y reutilización de código. Además, ROS está integrado con herramientas

[2]TCP/IP (Transmission Control Protocol/Internet Protocol) es un conjunto de protocolos de comunicación que se utilizan para interconectar dispositivos en una red. Establece las reglas para la transmisión de datos entre dispositivos y asegura que los datos se envíen de manera fiable y en el orden correcto. TCP se encarga de la transmisión de datos y la gestión de errores, mientras que IP se ocupa de la dirección y el encaminamiento de los paquetes de datos.

de visualización en tiempo real como RViz, que permiten trabajar con un gemelo digital del sistema robótico, facilitando la realización de pruebas y entrenamientos sin poner en riesgo el sistema físico.

Al momento de redactar este documento, la versión más reciente de ROS es ROS2 Iron Irwini. Sin embargo, se utilizará la versión ROS1 Noetic Ninjemys por especificación del fabricante del manipulador. El manipulador se trata de una versión descatalogada y no compatible con ROS2. No obstante, existen proyectos independientes que están trabajando en la migración del paquete necesario para el control del manipulador a ROS2. [3]

2.2.2. libfranka

Libfranka es una biblioteca de C++ encargada de implementar el cliente para el uso de FCI. Esta información ha sido obtenida de la documentación de la librería del fabricante [4]. Maneja la comunicación con el Control y se encarga de:

- Ejecución de comandos que no son de tiempo real.

  Estos comandos, basados en el protocolo TCP/IP, se ejecutan fuera del bucle de control de tiempo real. Abarca todos los comandos de la mano y algunos relacionados con la configuración de parámetros del brazo.

- Ejecución de comandos en tiempo real para ejecutar el bucle de control.

  Los comandos de tiempo real están basados en UDP [5] y requieren una conexión a 1kHz con el Control. Existen dos tipos de interfaces de tiempo real: **Generadores de movimiento**, que definen el movimiento del robot en el espacio articular o cartesiano, o **Controladores** que definen los pares que deben enviarse a las articulaciones del robot.

  Los generadores de movimiento son:

  - Posición articular q_c.
  - Velocidad articular $\dot{q_c}$
  - Pose cartesiana ${}^{O}T_{EE,c}$
  - Velocidad cartesiana ${}^{O}\dot{T}_{EE,c}$

  Los controladores son:

  - Controlador externo de par τ_d

[3] LCAS, *franka_ros2*, GitHub, URL: https://github.com/LCAS/franka_arm_ros2, ROS2 port of franka_ros for Franka Emika Robot (FER) Panda Robots, part of Agri-OpenCore (AOC) project, Fecha de acceso: 1 de septiembre de 2024.

[4] Franka Emika, *libfranka*, URL: https://frankaemika.github.io/docs/franka_ros.html, Franka Control Interface (FCI) - libfranka, Fecha de acceso: 6 de septiembre de 2024.

[5] User Datagram Protocol, UDP, es un protocolo de comunicación no orientado a conexión que permite la transmisión rápida de datos sin garantías de entregas o control de errores. Su uso está enfocado para aplicaciones donde la velocidad es prioritaria frente a la fiabilidad

  - Controlador interno de impedancia articular
  - Controlador interno de impedancia cartesiana

  Se pueden combinar las dos interfaces:

  - Un generador de movimiento y un controlador interno para seguir el movimiento comandado.
  - Un controlador interno e ignorar las señales de generación de movimiento.
  - Un generador de movimiento y un controlador externo para usar la cinemática inversa del Ordenador de Control en el controlador externo.

- Lectura del estado del robot (`robot state`) para obtener datos de los sensores.

  Proporciona:

  - Señales de nivel articulares: ángulo del motor y de articulación estimados, par articular y par externo estimado y colisión y contactos articulares.
  - Señales de nivel cartesianas: Posición cartesiana, colisión cartesiana.
  - Señales de interfaz: últimos valores comandados.

- Acceso a la biblioteca de modelos (`Model Library`) para calcular los parámetros cinemáticos y dinámicos deseados.

  Proporciona:

  - Cinemática directa de todas las articulaciones del robot.
  - Matriz jacobiana de todas las articulaciones.
  - Parámetros dinámicos: matriz de inercia, coriolis, vector de fuerza centrífuga y vector de gravedad.

- Procesamiento de la señal: Para facilitar el control del robot bajo conexiones desfavorables, se incluyen funciones encargadas de procesar los valores comandados por el usuario para adecuarlos a los límites de la interfaz. Para ello, se incluye un filtro de paso bajo de primer orden encargado de suavizar la señal comandada y un limitador de tasa que satura el tiempo derivativo de los valores comandados por el usuario.

- Gestión de errores: La tabla 2.2 resume los distintos errores que pueden aparecer durante el uso del modo FCI.

A continuación, se describe en detalle el metapaquete de ROS encargado de gestionar y controlar el manipulador.

Categoría	Errores específicos
Valores iniciales incorrectos	`joint_motion_generator_start_pose_invalid`, `cartesian_position_motion_generator_start_pose_invalid`, `cartesian_motion_generator_start_elbow_invalid`, `cartesian_motion_generator_elbow_sign_inconsistent`
Violación de límites de posición	`joint_motion_generator_position_limits_violation`, `cartesian_motion_generator_joint_position_limits_violation`
Violación de límites de velocidad articulares	`joint_motion_generator_velocity_limits_violation`
Discontinuidades articulares	`joint_motion_generator_velocity_discontinuity`, `joint_motion_generator_acceleration_discontinuity`
Violación de límites de velocidad (cartesiano)	`cartesian_motion_generator_velocity_limits_violation`
Discontinuidades cartesianas	`cartesian_motion_generator_velocity_discontinuity`, `cartesian_motion_generator_acceleration_discontinuity`
Violación del límite de par	`controller_torque_discontinuity`
Errores reflejo (par estimado supera limites)	`cartesian_reflex`, `joint_reflex`
Auto colisión	`self_collision_avoidance_violation`
Límite de sensor de par alcanzado	`tau_j_range_violation`
Potencia máxima alcanzada	`power_limit_violation`
Límites articulares o cartesianos alcanzados	`joint_velocity_violation`, `cartesian_velocity_violation`

Tabla 2.2: Resumen de errores comunes durante el uso de FCI

2.2.3. franka_ros

Se trata del metapaquete de ROS desarrollado por el fabricante del manipulador para integrar la librería libfranka en ROS. Este paquete actúa como puente entre el hardware del manipulador y el software en ROS, proporcionando una interfaz de alto nivel para el control y la supervisión del robot.

La información de este apartado ha sido extraída de la documentación completa del metapaquete [6].

Entre los paquetes que contiene, los más relevantes son:

- `franka_description`: Este paquete contiene la descripción del robot y el efector final en términos cinemáticos, límites articulares, superficies visibles y espacio de colisión.
- `franka_hw`: Este paquete contiene la abstracción de hardware del robot para el framework de ROS control basando en libfranka.

[6] Franka Emika, *franka_ros*, URL: https://frankaemika.github.io/docs/franka_ros.html, Franka Control Interface (FCI) - franka_ros, Fecha de acceso: 1 de septiembre de 2024.

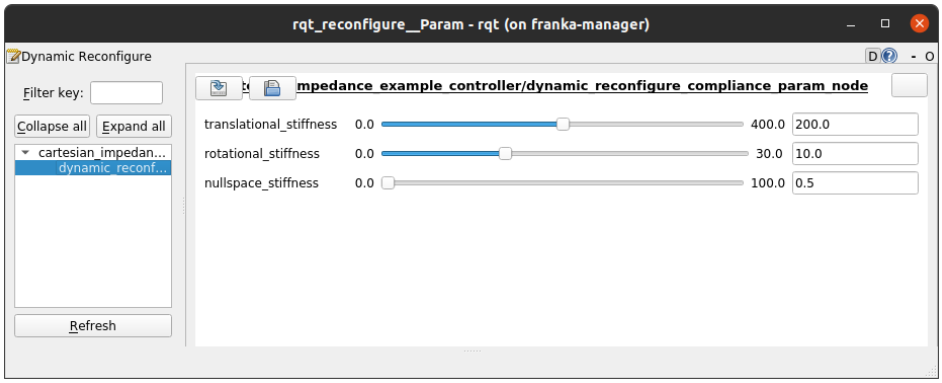


Figura 2.5: Captura de pantalla de la ventana que permite modificar los parámetros dinámicos del control de impedancia de Franka

- `franka_control`: Proporciona el nodo `franka_control_node`.
- `franka_visualization`: Contiene nodos encargados de publicar el estado articular del robot para su visualización en RViz.
- `franka_example_controllers`: Implementa controladores de ejemplo. Cada ejemplo contiene un archivo de lanzamiento independiente encargado de iniciar el control en el robot.

  Entre los controladores de ejemplo más relevantes se encuentran el controlador de impedancia cartesiana y el controlador de velocidad cartesiana.

  Controlador de ejemplo de impedancia cartesiana Permite modificar el punto de equilibrio del manipulador mediante una visualización en tiempo real. Para ello, se usa el visor RViz. Esta visualización contiene un marcador, llamado `equilibrium_pose_marker`, que permite modificar la posición y orientación del efector final del manipulador. La figura 2.6 muestra una captura de pantalla de RViz durante la ejecución del control de impedancia. El usuario modifica la posición y orientación del efector final en el espacio modificando las flechas y las ruletas con el ratón.

  Este código permite modificar algunos parámetros dinámicos del control de impedancia. La figura 2.5 muestra la ventana que permite reconfigurar estos parámetros dinámicos durante la ejecución.

  Los aspectos teóricos del control de impedancia se definen en la sección 3.2.2.1.

  Controlador de ejemplo de velocidad cartesiano Consiste en la ejecución de un ciclo de movimiento en bucle. No permite interacción con el usuario. Su ejecución requiere que el manipulador tenga una configuración articular inicial determinada. Los aspectos teóricos del control de velocidad cartesiano se definen en la sección 3.2.2.2.

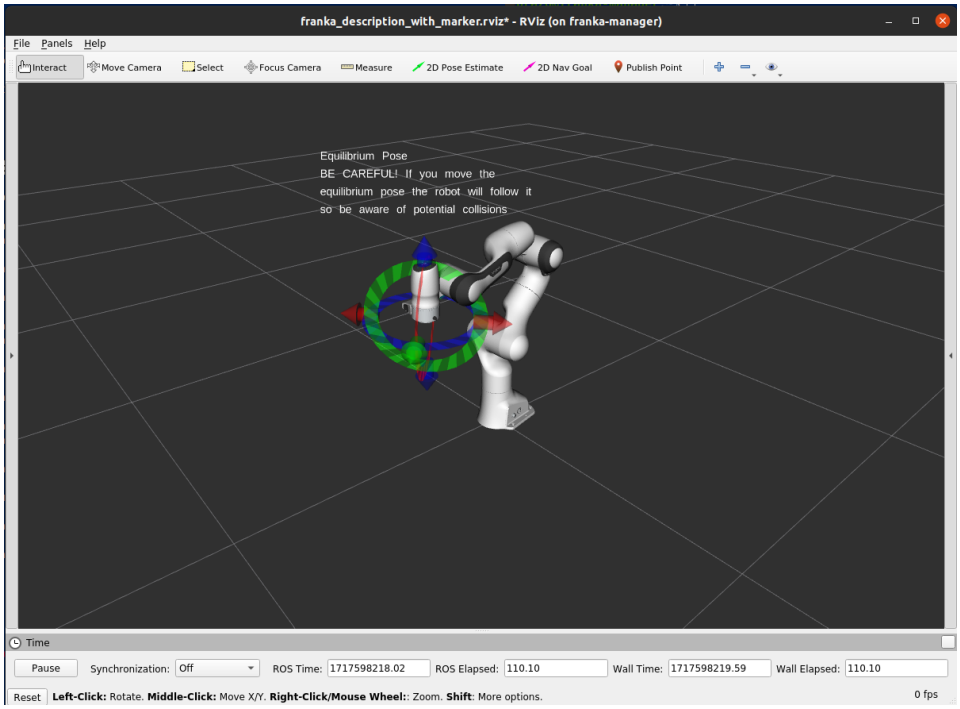


Figura 2.6: Captura de pantalla de RViz durante la ejecución del control de impedancia de franka_example_controllers

- `franka_msgs`: Este paquete contiene los mensajes usados por los paquetes `franka_hw` y `franka_control` para publicar el estado del robot u ofrecer características de libfranka en el ecosistema de ROS.

2.3. Funciones básicas de navegación ROS para un manipulador móvil omnidireccional

El Trabajo de Fin de Grado titulado "Funciones básicas de navegación ROS para un manipulador móvil omnidireccional", realizado por Francisco de Paula Carbonero Navarro, se centró en la implementación inicial del controlador de los motores de la plataforma móvil en ROS [21]. Este trabajo estableció las bases necesarias para la integración de las funciones de navegación y teleoperación mediante el uso de un mando y de un sensor LiDAR [7]. El sensor LiDAR no se ha utilizado en este proyecto.

La aportación de este trabajo fue la documentación de las características de la base, especificaciones y componentes, la creación de un espacio de trabajo de ROS e implementación de los paquetes

[7] Light Detection and Ranging, LiDAR, es una tecnología de detección de obstáculos que utiliza pulsos de luz láser para medir distancias. Un sensor LiDAR emite pulsos de láser hacia el entorno y mide el tiempo que tarda el reflejo en volver al sensor. Con estos datos, se puede crear un mapa tridimensional del entorno. Dentro del ámbito de la robótica, se utiliza para detección de obstáculos y generación de modelos 3D del espacio.

de ROS necesarios para controlar y teleoperar los motores de la base. El capitulo *Desarrollo del trabajo* abarca las siguientes secciones:

- Primeros pasos y problemas: Relata las dificultades encontradas durante el desarrollo del trabajo.
- Modificación y creación de nodos: Describe la creación del espacio de trabajo de ROS y su estructura.
- `joy_ros`: Explica el paquete encargado de leer el estado del mando y hacerlo disponible para ROS. Se detalla su topología y el uso de la herramienta *Jstest-gtk* para modificar diferentes aspectos del mando como el valor de los ejes o el mapeo de botones. También se incluyen dos capturas de pantalla de los códigos más relevantes.
- `ydlidar`: Documenta la integración del sensor LiDAR en ROS, detallando la topología del paquete.
- `roboclaw_ros`: Explica el paquete encargado de controlar y teleoperar la base. Se desglosa la topología del paquete, un diagrama de nodos y topics del sistema de control de la base (llamado como Grafo de funcionamiento de roboclaw_ros). Además, detalla las modificaciones realizadas en el código original para adaptarlo al uso de dos placas controladoras y la implementación de la cinemática de la base.

  El bucle de control del nodo de teleoperación permite seleccionar entre dos escalas de velocidad: modo normal (se activa con el botón LB) y modo turbo (se activa con el botón RB). El modo turbo permite comandar velocidades más altas que el modo normal. El joystick JR comanda las velocidades cartesianas lineales mientras que el joystick JL controla la velocidad angular (rotación de la base sobre le eje Z).

Durante el capítulo 3 de la presente memoria se ha profundizado en la documentación del funcionamiento del controlador de la base, en el apartado 3.2.1, y su integración en ROS, en el apartado 3.1.2.2. También se ha documentado la integración del mando dentro de ROS en el apartado 3.1.2.1. Además, el apartado 3.3.1 desglosa la creación de un paquete exclusivo para teleoperación del mando y propone un método alternativo al uso de dos únicas escalas de velocidad usando el código base propuesto por el trabajo de Francisco Carbonero 2.3.

El trabajo realizado con anterioridad sobre la plataforma RAFI supone el punto de partida en el desarrollo del robot, y permite que cada componente del robot pueda operarse de forma independiente. Un usuario podría conectarse al manipulador y programarlo, o al PC integrado en la base y operarlo. Pero no existe, hasta ahora, una estructura que permita al usuario conectarse al robot RAFI e interactuar de forma simultánea tanto con la base como con el manipulador de forma simultánea. Por lo tanto, es necesario el desarrollo de una interfaz que integre estas funcionalidades y permita comunicar cada elemento del sistema entre sí, incluyendo al propio usuario.

CAPÍTULO 3

Metodología

Contenido

En este capítulo, se detallan los desarrollos técnicos llevados a cabo en el marco de este proyecto. Se presenta la metodología empleada, abordando tres aspectos fundamentales. En primer lugar, se describe la implementación del sistema distribuido, incluyendo la creación de una red de nodos y la creación de infraestructura de comunicación inalámbrica. A continuación, se analiza el sistema de control del robot, abarcando el controlador de la base y distintos controladores para el manipulador. Finalmente, se presenta la interfaz de teleoperación,

Figura 3.1: Código QR del repositorio de GitHub que contiene los desarrollos de este proyecto. https://github.com/TaISLab/Rafi

que permite la interacción con el sistema mediante el uso de mando. Este capítulo proporciona una visión detallada de las soluciones implementadas para lograr los objetivos del proyecto.

Los paquetes de ROS desarrollados e implementados en este proyecto se encuentran en el repositorio del proyecto RAFI. Puede acceder a ellos escaneando el código QR de la figura 3.1 o mediante la siguiente dirección https://github.com/TaISLab/Rafi.

3.1. Implementación del sistema distribuido

Un sistema distribuido es un conjunto de dispositivos interconectados que trabajan de manera coordinada para lograr un objetivo común. Estos dispositivos operan de forma independiente pero se comunican entre sí mediante una red para compartir información, datos y recursos. De esta forma, se aprovecha la capacidad computacional de cada uno de ellos. ROS permite el uso de una arquitectura distribuida donde diferentes nodos se ejecutan en distintos PCs pero se comunican entre sí. Esto optimiza el rendimiento y eficiencia del sistema. Las limitaciones técnicas, principalmente el bajo rendimiento gráfico de la comunicación SSH, han motivado la elección de esta arquitectura.

3.1.1. Descripción del sistema distribuido

Este apartado describe la arquitectura del sistema distribuido, la configuración de la red de nodos distribuidos de ROS y la creación de una red inalámbrica.

3.1.1.1. Arquitectura del sistema distribuido

La figura 3.2 ilustra la arquitectura del sistema distribuido utilizado en este proyecto. Se distinguen dos núcleos computacionales:

- PC integrado: Ejecuta los nodos que interactúan con hardware conectado a él. Estos son el controlador del manipulador y el controlador de la base.
- PC de desarrollo: Ejecuta el nodo relacionado con la visualización de RViz y otros nodos necesarios durante el desarrollo y depuración.

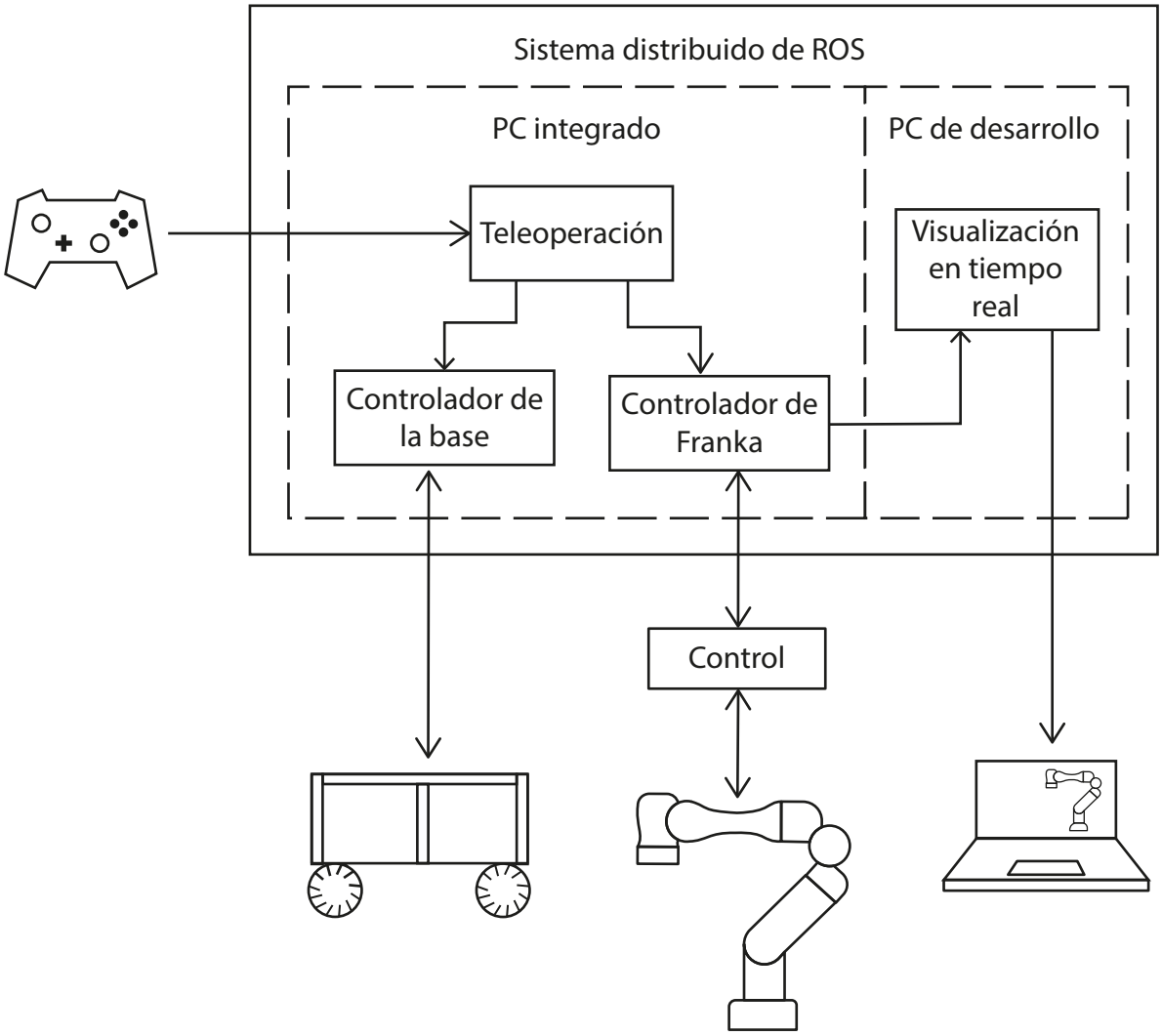


Figura 3.2: Arquitectura del sistema distribuido

El nodo de teleoperación se puede ejecutar desde ambas máquinas aunque debe ser desde la misma en la que está conectado el USB del mando.

3.1.1.2. Configuración de la Red de Nodos Distribuidos de ROS

La figura 3.3 muestra un esquema de la red de nodos distribuidos de ROS. En este esquema, dos PC se encuentran conectados a la misma red, donde uno de ellos actúa como servidor, conocido como *ROS MASTER*. El otro ordenador actúa como cliente de ese servidor. Esta red usa el protocolo IPC/IP [1] para comunicarse.

[1] El protocolo IPC/IP, Inter-Process Communication over Internet Protocol, permite la comunicación entre procesos que se ejecutan en diferentes máquinas a través de redes IP. Este protocolo combina las capacidades de IPC, para permitir el intercambio de datos entre programas, y el IP, que permite que los datos viajen correctamente a través de una red global o local.

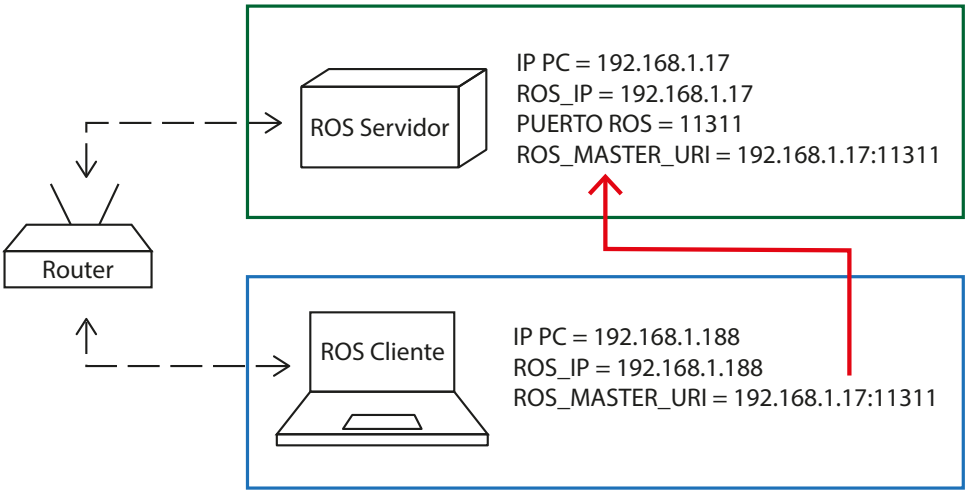


Figura 3.3: Esquema de la red de nodos distribuidos de ROS

ROS utiliza variables de entorno para la configuración de la red. La variable `ROS_MASTER_URI` determina la ubicación del servidor en la red. Esta variable debe estar configurada en cada máquina para que los nodos puedan comunicarse con el servidor. La forma general de esta variable es:

```
http://IP_ROS_MASTER:PUERTO
```

De forma predeterminada, el puerto es 11311. Esta información aparece durante la ejecución del comando `roscore`.

Configuración de la máquina servidor Esta maquina contiene el ROS MASTER. Debe configurarse con los siguientes comandos:

```
$ roscore
$ export ROS_MASTER_URI = http://192.168.1.17:11311
$ export ROS_HOSTNAME=192.168.1.17
$ export ROS_IP=192.168.1.17
```

Donde:

- roscore: Inicia ROS.
- `ROS_MASTER_URI`: Contiene la dirección IP del ROS MASTER y el puerto utilizado.
- `ROS_HOSTNAME` y `ROS_IP`: Contienen la dirección IP de esta máquina.

Configuración de la máquina cliente Esta máquina se debe configurar para identificar el servidor. No es necesario ejecutar `roscore`. Los comandos necesarios son:

```
$ export ROS_MASTER_URI = http://192.168.1.17:11311
$ export ROS_HOSTNAME=192.168.1.188
$ export ROS_IP=192.168.1.188
```

Dónde:

- `ROS_MASTER_URI`: Debe coincidir con la dirección IP del ROS MASTER.
- `ROS_HOSTNAME` y `ROS_IP`: Contienen la dirección IP de la máquina cliente.

3.1.1.3. Configuración de una red inalámbrica en el laboratorio

Se ha creado una red inalámbrica encargada de dotar de infraestructura a todo el sistema distribuido. A continuación, se detalla su procedimiento de configuración:

1. **Conexión del router a Internet**: El router necesita un punto Ethernet con conexión a internet. No todas las rosetas permiten la creación de una red inalámbrica. Se ha elegido la roseta R13Q8 del laboratorio.

2. **Configuración básica del router**: Esto incluye la asignación de un nombre de red (SSID) y una contraseña. La configuración se realiza desde la dirección IP del router: `192.168.1.1`

   La identidad de la red es:

   ```
   SSID: ASUS_RAFI
   Contraseña: RED_24%rco
   ```

   La nueva credencial de acceso a la configuración del router es:

   ```
   Usuario: admin
   Contraseña: admin_RAFI
   ```

3. **Implementación de filtrado MAC como medida de seguridad adicional**. Se trata de una técnica de seguridad en redes que permite controlar el acceso inalámbrico a un router basándose en la dirección MAC [2] de los dispositivos. Solo los dispositivos con direcciones MAC permitidas pueden conectarse a la red. No protege ante conexiones por Ethernet. Los únicos dispositivos autorizados son el PC integrado y el PC de desarrollo.

[2] Una dirección MAC, *Media Access Control*, es un identificador único asignado a la tarjeta de red de un dispositivo, utilizado para identificarlo de manera específica en una red local.

La instalación de un router en el laboratorio requiere permiso del Servicio Central de Informática (SCI) de la universidad. La obtención de dicho permiso implica garantizar la seguridad de la red, motivo por el que se ha implementado el filtro MAC. Como medida de seguridad adicional, se recomienda la desconexión del router de la red eléctrica cuando no se necesite la red inalámbrica.

3.1.2. Interconexión y comunicación entre componentes

Este apartado describe cómo se conectan los distintos componentes físicos dentro del sistema distribuido, incluyendo su comunicación con ROS.

3.1.2.1. Integración del mando

El mando se conecta al PC a través de un puerto USB. Se puede conectar tanto al PC de desarrollo como al PC integrado. En este caso, se ha optado por la conexión con el PC integrado para garantizar la supervisión del sistema robótico durante su uso.

Para integrar el mando en ROS, se utiliza un driver genérico para mandos en Linux, que se encarga de traducir las señales del mando en mensajes ROS. Este driver se implementa como un paquete de ROS, cuyo funcionamiento implica la ejecución de un nodo llamado `joy_node`. Este nodo publica mensajes del tipo *sensor_msgs/Joy* [3] que contienen información sobre el estado actual de cada uno de los botones y ejes del mando. El paquete `Joy` y su documentación se encuentran disponibles en la documentación oficial de ROS. [4]

El mensaje `sensor_msgs/Joy` tiene la siguiente estructura:

- Header: Este campo incluye una marca de tiempo que indica el momento exacto en el que se recibe la señal del mando.
- Axes: Es un array de valores *float32* que representan las mediciones de los ejes del mando.
- Buttons: Es un array de valores *int32* que contiene el estado de los botones del mando. Cada elemento corresponde a un botón y tiene un valor de 1 si el botón está presionado o 0 si no lo está.

El nodo `joy_node` se encarga de publicar un topic con la información de este mensaje, poniéndolo a disposición de todos los nodos que deseen subscribirse a él.

3.1.2.2. Integración de Roboclaw

El sistema cuenta con dos controladores Roboclaw, cada uno encargado de gestionar los motores de la base móvil. Estos controladores requieren de dos puertos USB para la conexión física con el PC integrado.

[3] Documentación de ROS, *Joy Message*, URL: http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Joy.html, Contiene información sobre el mensaje de ROS, Fecha de acceso: 17 de agosto de 2024.

[4] Documentación de ROS, *joy*, URL: http://wiki.ros.org/joy, Contiene información sobre el paquete de ROS, Fecha de acceso: 17 de agosto de 2024.

Para la integración con ROS se utiliza el paquete `roboclaw_ros`. Este paquete contiene tanto el driver como el nodo de ROS necesario para la comunicación con los controladores. Se puede acceder a la documentación del paquete desde la web del desarrollador. [5]

El nodo de ROS para los controladores Roboclaw se configura mediante un archivo de lanzamiento dentro del paquete. Este es el `roboclaw.launch` y contiene los parámetros necesarios para la configuración. Estos son:

- dev: Especifica el puerto USB donde está conectado el Roboclaw. Al ser dos , estos son:

```
1       <arg name="dev1" default="/dev/ttyACM0"/>
2       <arg name="dev2" default="/dev/ttyACM1"/>
```

- baud: Define la velocidad de comunicación en baudios. Por defecto, 115200.
- adress: Dirección del controlador en la red de dispositivos. Estos son:

```
1       <arg name="address1" default="128"/>
2       <arg name="address2" default="129"/>
```

- max_speed: Velocidad máxima para los motores en metros por segundo. Por defecto, 0,8.
- ticks_per_meter: Número de ticks del encoder por metro de movimiento. Este parámetro está relacionado con el cálculo de la odometría.
- base_width: Distancia entre las ruedas y el suelo, medida en metros.

Este archivo contiene la información necesaria para iniciar el nodo.

El topic `/cmd_base_vel` contiene un mensaje del tipo `geometry_msgs/Twist`. Este mensaje contiene los comandos de velocidad para controlar la base móvil. El nodo `roboclaw_node` se suscribe a este topic y traduce el contenido en señales interpretables por los motores.

El nodo publica en el topic `/odom` de tipo `nav_msgs/Odometry`, que proporciona información sobre la posición y orientación en el espacio a lo largo del tiempo. Sin embargo, en este proyecto no se ha usado ese topic.

El mensaje `geometry_msgs/Twist` se utiliza para expresar velocidades en el espacio libre, dividiéndose en componentes lineales y angulares. Sus componentes son:

- linear: Es un vector de tipo *Vector3* que representa la velocidad lineal en las tres componentes x, y, z. No se utiliza la ultima componente ya que la base no puede modificar su posición en el eje z.

[5] Repositorio de GitHub del paquete *roboclaw_ros*, URL: https://github.com/sonyccd/roboclaw_ros, El repositorio contiene información detallada sobre la instalación y configuración del paquete. Fecha de acceso: 17 de agosto de 2024.

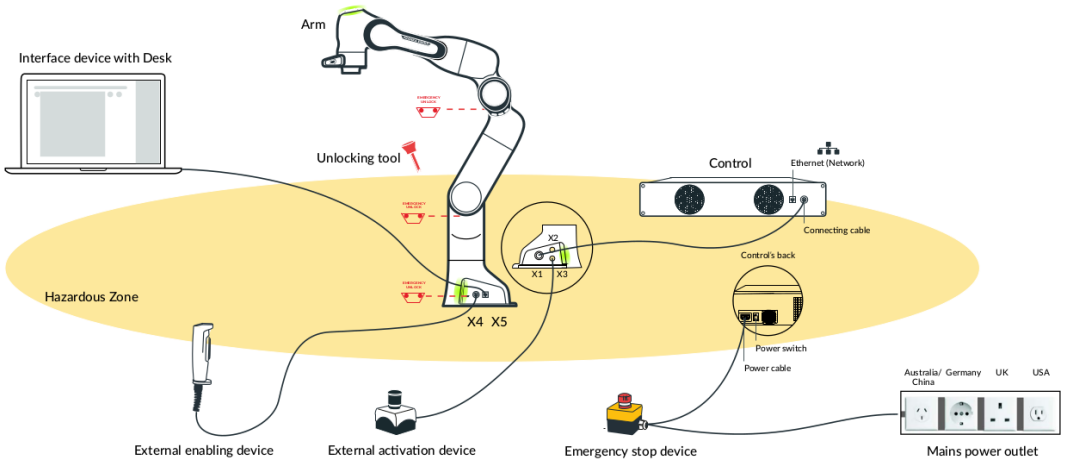


Figura 3.4: Diagrama de conexionado de los componentes del Franka Emika Panda. Fuente:[6]

- angular: Es un vector de tipo *Vector3* que representa la velocidad angular alrededor de los tres ejes x, y, z. Solo se utiliza la tercera componente, encargada de la rotación en el plano XY.

3.1.2.3. Integración del manipulador

El proceso de integración del manipulador en el sistema distribuido se divide en distintas fases: conexión física entre componentes, configuración de la conexión entre el control y el PC integrado, configuración del PC integrado y preparación del manipulador.

Conexión física entre componentes La figura 3.4 ilustra las conexiones entre los distintos componentes del sistema:

- Manipulador y Control: El manipulador se conecta a control mediante el conector X1, el cual se encuentra en la base lateral del manipulador y en la parte trasera del control.
- Control y PC integrado: La conexión entre el control y el PC integrado se realiza a través de un cable Ethernet. Esta conexión requiere una configuración específica de red en el PC.
- Dispositivo de Activación Externo y Manipulador: Este dispositivo se conecta indistintamente al puerto X2 o X3 del manipulador.
- Dispositivo de parada de Emergencia: Este dispositivo está integrado en el cable de alimentación del control.
- Dispositivo de habilitación externo: Aunque no se ha utilizado durante este proyecto, este dispositivo puede conectarse al puerto X4 del manipulador, permitiendo su uso simultáneo con el Dispositivo de activación externo. También lo puede reemplazar.

Configuración de la comunicación Ethernet Se deben configurar de forma específica los parámetros de la red Ethernet entre el PC integrado y el Control. Para ello, se configura una IP estática en el PC integrado con los siguientes parámetros:

- Dirección IP: 172.16.0.1
- Máscara de red: 24

Configuración del PC integrado El PC integrado actúa como puente entre el Control y ROS. Se debe configurar el sistema instalando la librería *libfranka* y el metapaquete *franka_ros*. Este proceso está desglosado paso a paso en la documentación oficial de Franka Emika. [7]

Preparación del manipulador: Activación del FCI y desbloqueo de frenos Para hacer uso del modo FCI, se debe activar cada vez que se inicia el manipulador. Además, se deben desbloquear los frenos. Se ha implementado un paquete de ROS llamado `franka_lock_unlock` [8] , desarrollado por un usuario de GitHub, que permite realizar esta tarea directamente desde la terminal.

El comando necesario para activar el FCI y desbloquear los frenos es:

```
1    rosrun franka_lock_unlock __init__.py -u -l -w -r -p -c 172.16.0.2
        RAFI rafi_ISA!'
```

Para facilitar su uso, se ha creado un alias:

```
1    alias franka_unlock='source ~/ros_projects/franka_ws/catkin_ws/
        devel/setup.sh && rosrun franka_lock_unlock __init__.py -u -l -
        w -r -p -c 172.16.0.2 RAFI rafi_ISA!'
```

Este código debe permanecer en ejecución para mantener una conexión persistente. Para revertir esta acción, el proceso puede ser detenido manualmente desde la terminal con la combinación ctrl+c.

Lanzamiento del controlador Con todos los componentes conectados y configurados correctamente, el sistema está ahora listo para operar en conjunto con ROS. El funcionamiento y lanzamiento del controlador del manipulador se desglosa en 3.2.2.

[6] Imagen tomada del manual en inglés del manipulador Franka Emika Panda: FRANKA EMIKA ROBOT'S INSTRUCTION HANDBOOK, página 104, 2021. URL: https://franka.de/documents

[7] Franka Emika, *Installation on Linux*, URL: https://frankaemika.github.io/docs/installation_linux.html, Guía de instalación de libfranka, franka_ros y configuración del real-time kernel, Fecha de acceso: 11 de junio de 2024.

[8] jk-ethz, *franka_lock_unlock*, GitHub, URL: https://github.com/jk-ethz/franka_lock_unlock, Paquete de ROS que activa el FCI y desbloquea los frenos del Franka Emika Panda, Fecha de acceso: 4 de junio de 2024.

3.1.2.4. Comunicación entre PCs

La conexión SSH permite acceder y controlar un PC de forma remota sin necesidad de acceso físico al dispositivo. Sin embargo, se debe tener en cuenta que para aplicaciones con altas demandas gráficas el rendimiento se reduce considerablemente. Para garantizar una conexión efectiva, ambos ordenadores deben estar en la misma red.

Se quiere establecer la conexión SSH desde el PC de desarrollo al PC integrado. Éste es el proceso:

- Obtener la dirección IP del PC integrado. Utiliza el comando `$ hostname -I` para encontrar su dirección IP.
- Conectar el PC de desarrollo al PC integrado:

```
$ ssh -X usuario@direccion_ip
```

  El usuario es ”brazo” y la dirección IP es ”192.168.1.17”.

- Introducir la contraseña: ”brazo”.

Para simplificar este proceso, se ha creado un alias para facilitar la conexión SSH de manera rápida:

```
$ rafi_ssh = 'ssh -X brazo@192.168.1.17'
```

3.1.3. Resumen de la implementación del sistema distribuido

En esta sección se ha detallado cómo los diferentes componentes del sistema distribuido se interconectan y comunican entre sí, estableciendo una red robusta para el funcionamiento conjunto del robot. Desde la configuración de la red de nodos distribuidos hasta la integración física de cada dispositivo con ROS, se han cubierto todos los aspectos necesarios para asegurar una comunicación efectiva e interoperabilidad entre todos los elementos del sistema. Con la infraestructura y la comunicación establecidas, el siguiente paso es explorar los esquemas de control del robot.

3.2. Sistema de control del robot

En esta sección se estudia el sistema del control del robot. En primer lugar, se analiza el controlador de la base, explicando el diagrama de entrada y salida del controlador, el modelo matemático que describe su comportamiento y el diagrama de nodos y topics tras su implementación en ROS. En segundo lugar, se estudia el controlador del manipulador, para el cual se han desarrollado dos versiones: controlador de impedancia cartesiana y controlador de velocidad cartesiana. Para cada uno, se explica el diagrama de entradas y salidas del controlador, el modelo matemático que describe su comportamiento y su diagrama de nodos y topics de ROS. Por último, se propone un esquema de control que unifica el control de la base y del manipulador. Se presentan dos esquemas unificados.

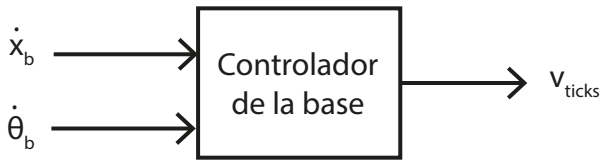


Figura 3.5: Diagrama de las entradas y salidas del controlador de la base

3.2.1. Controlador de la base

El controlador de la base se encarga de gestionar el movimiento del robot mediante el ajuste de las velocidades angulares de los motores. Su función principal es transformar las velocidades deseadas en el espacio cartesiano ($\dot{x}_b$, $\dot{y}_b$ y $\dot{\theta}_b$) en señales de control para los motores.

Este controlador se trata de un control de velocidad. Mientras que el control de posición se enfoca en alcanzar una posición específica, el control de velocidad regula la dinámica del movimiento del robot en tiempo real, adaptando las velocidades angulares de los motores a las velocidades cartesianas deseadas.

Diagrama de entradas y salida del controlador La figura 3.5 muestra el diagrama de entradas y salidas del controlador. Las entradas son:

- Velocidad lineal deseada ($\dot{x}_b$). Se trata de un vector que contiene la velocidad en el eje X e Y.
- Velocidad angular deseada ($\dot{\theta}_b$)

La salida del controlador es un vector de velocidades angulares en ticks por segundo. Tiene la forma:

$$v_{ticks} = \begin{bmatrix} v_{1r_{ticks}} & v_{1l_{ticks}} & v_{2r_{ticks}} & v_{2l_{ticks}} \end{bmatrix} \tag{3.1}$$

Dónde:

- $v_{1r_{ticks}}$: Velocidad, en ticks, para la rueda delantera derecha.
- $v_{1l_{ticks}}$: Velocidad, en ticks, para la rueda delantera izquierda.
- $v_{2r_{ticks}}$: Velocidad, en ticks, para la rueda trasera derecha.
- $v_{2l_{ticks}}$: Velocidad, en ticks, para la rueda trasera izquierda.

Estas salidas se envían directamente a los motores a través de las interfaces de control.

Modelo matemático El modelo matemático del controlador se fundamenta en la cinemática diferencial del robot, la cual permite controlar las cuatro ruedas motrices de manera independiente.

Los parámetros del modelo son:

- B: Ancho de la base del robot (BASE_WIDTH), medido en metros
- R: Radio de las ruedas en metros
- T: Ticks por metro (TICKS_PER_METER), una constante de conversión de metros por segundo a ticks por segundo.

La cinemática diferencial del robot se describe mediante las siguientes ecuaciones que relacionan la velocidad lineal y angular deseada del robot con las velocidades angulares de cada uno de los motores:

$$\begin{aligned} v_{1r} &= \dot{x}_b + \dot{y}_b + \frac{\dot{\theta}_b \cdot \mathrm{B}}{2} \\ v_{1l} &= \dot{x}_b - \dot{y}_b - \frac{\dot{\theta}_b \cdot \mathrm{B}}{2} \\ v_{2r} &= \dot{x}_b - \dot{y}_b + \frac{\dot{\theta}_b \cdot \mathrm{B}}{2} \\ v_{2l} &= \dot{x}_b + \dot{y}_b - \frac{\dot{\theta}_b \cdot \mathrm{B}}{2} \end{aligned} \tag{3.2}$$

Estas ecuaciones calculan las velocidades de las ruedas en metros por segundo pero se deben convertir a unidades de ticks por segundo, que son las utilizadas en el control de los motores:

$$v_{1r_{ticks}} = v_{1r} \cdot T_{v1r} \tag{3.3}$$

Por último, las velocidades en ticks por segundo son enviadas, a través de las placas Roboclaw, a los motores mediante las siguientes funciones de Python:

```
1     self.roboclaw1.SpeedM1M2(self.address1, v1l_ticks, v1r_ticks)
2     self.roboclaw2.SpeedM1M2(self.address2, v2r_ticks, v2l_ticks)
```

En resumen, el controlador traduce las velocidades deseadas en el plano cartesiano a comandos precisos que regulan la velocidad angular de cada rueda.

Diagrama de nodos y topics del controlador La figura 3.6 muestra el diagrama de nodos y topics del controlador de la base durante su ejecución en ROS. La entrada se realiza a través del nodo `/cmd_base_vel`, que recibe la velocidad deseada en forma de mensajes del tipo `/geometry_msgs::Twist`. El nodo del controlador `/roboclaw_node` está subscrito al nodo `/cmd_base_vel` y se encarga de procesar estos mensajes para convertirlos en señales de control que son enviadas a las placas Roboclaw. Estas señales ejecutan movimiento angular en los motores.

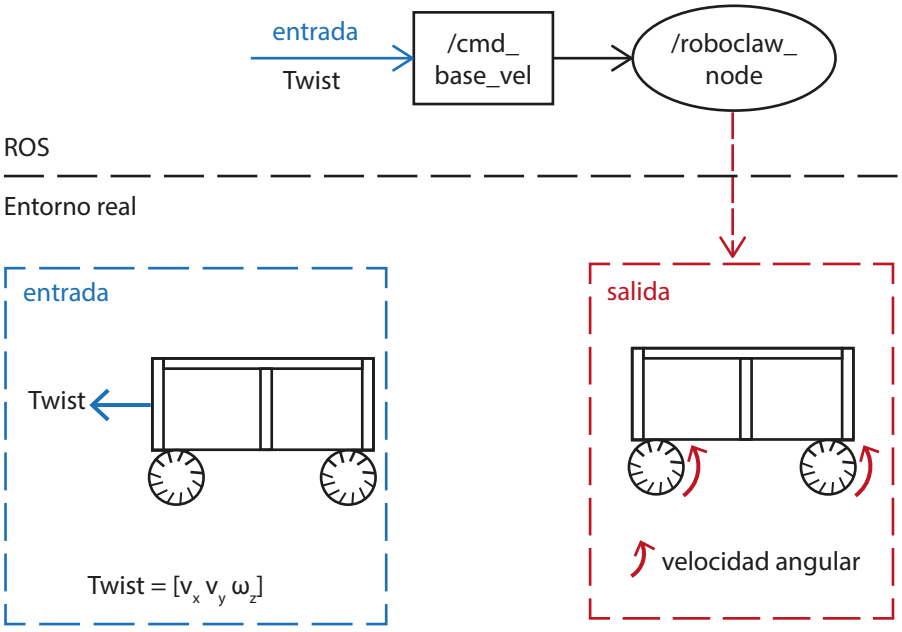


Figura 3.6: Diagrama de nodos y topics del controlador de la base

3.2.2. Controlador del manipulador

El control del manipulador es un aspecto fundamental en la operación del manipulador ya que determina cómo el robot interactúa con su entorno y ejecuta tareas. En este apartado, se describen los dos enfoques de control utilizados para gestionar el comportamiento del efector final en el espacio cartesiano: el control de impedancia cartesiano y el control de velocidad cartesiana.

Ambos métodos permiten al manipulador realizar movimientos precisos y adaptarse a diferentes entornos, pero lo hacen de manera complementaria. El control de impedancia cartesiano se enfoca en la interacción segura y controlada con el entorno, mientras que el control de velocidad cartesiana permite ejecutar movimientos de manera directa y eficiente. A continuación, se presentan los detalles de cada uno de estos enfoques.

3.2.2.1. Controlador de impedancia cartesiano

El controlador de impedancia cartesiano es una técnica fundamental en robótica para garantizar una interacción segura entre un robot y su entorno. Este controlador regula la dinámica del manipulador en el espacio cartesiano, ajustando la respuesta del robot a fuerzas y pares externos para alcanzar los objetivos de posición y orientación deseados. Se ha modificado el controlador incluido dentro del paquete `franka_example_controllers` para aceptar entradas de equilibriumPose directamente, eliminando la subscripción al `equilibrium_pose_marker` de RViz.

Diagrama de entradas y salidas del controlador La figura 3.7 muestra un diagrama simplificado de entradas y salidas del controlador de impedancia cartesiano del manipulador. Las entradas son:

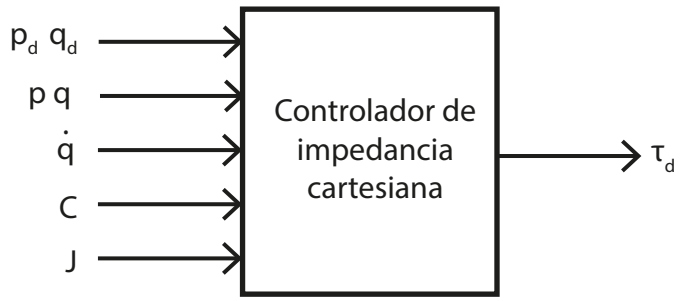


Figura 3.7: Diagrama de entradas y salidas del controlador de impedancia cartesiano de Franka

- Posición y orientación deseadas del efector final (p_d y q_d).
- Posición y Orientación actual del efector final (p y q).
- Velocidad articular actual ($\dot{q}$)
- Matriz de Coriolis (C)
- Matriz Jacobiana del manipulador (J)

La salida del controlador es:

- Par deseado para los actuadores ($\tau_{d_1}, \tau_{d_2}, ...\tau_{d_7}$)

Modelo matemático A continuación se detalla el modelo matemático utilizado por el controlador para el cálculo del par deseado a partir de una posición y orientación deseada del efector final.

1. **Cálculo del error en el espacio cartesiano** El error total en el espacio cartesiano se calcula combinando el error en posición y el error en orientación. A continuación, se desglosan los pasos para calcular estos errores:

   a) **Error en posición**

   El error en posición se obtiene restando el vector de posición deseado del vector de posición actual:

   $$e_{pos} = p - p_d \tag{3.4}$$

   Donde:

   - p es la posición actual del efector final.
   - p_d es la posición deseada.

   b) **Cálculo del error en orientación**

   Para calcular el error en orientación utilizando cuaterniones, se siguen los siguientes pasos:

1) **Alineación de cuaterniones**

Para evitar rotaciones innecesarias, si el producto escalar entre el cuaternión actual y el cuaternión deseado es negativo, se niega el cuaternión actual [9] :

$$\begin{aligned} &\text{Si } q_d \cdot q < 0, \text{ entonces } q \rightarrow -q \\ &\text{Si } q_d \cdot q > 0, \text{ entonces } q \rightarrow q \end{aligned} \tag{3.5}$$

2) **Cuaternión de diferencia**

Con los cuaterniones alineados correctamente, se calcula el cuaternión diferencia:

$$q_{error} = q^{-1} \cdot q_d \tag{3.6}$$

Donde:

- q es el cuaternión que representa la orientación actual.
- q_d es el cuaternión que representa la orientación deseada.
- q_{error} es el cuaternión de error que representa la diferencia entre la orientación actual y la deseada.

Las tres componentes vectoriales (x, y, z) del cuaternión de diferencia se extraen para formar parte del error en orientación.

3) **Transformación al marco de referencia base**

El error en orientación se transforma del marco de referencia del efector final al marco de referencia base del robot:

$$e_{ori} = -R \cdot q_{error} \tag{3.7}$$

Donde:

- R es la matriz de rotación que transforma las coordenadas al marco base.
- El término $-R \cdot q_{error}$ ajusta el signo del error para que las correcciones se apliquen en la dirección correcta en el contexto global del robot.

c) **Error total en el espacio cartesiano**

El error total se representa como una matriz 6x1 que combina el error en posición y el error en orientación:

$$e = \begin{bmatrix} e_{\text{pos}} \\ e_{\text{ori}} \end{bmatrix} = \begin{bmatrix} p - p_d \\ -R \cdot (q^{-1} * q_d) \end{bmatrix} \tag{3.8}$$

Donde:

- e es la matriz de error en el espacio cartesiano.

[9] Los cuaterniones tienen la propiedad de que q y $-q$ representan la misma rotación en el espacio tridimensional, lo que significa que ambos describen la misma orientación.

- p y p_d son la posición actual y deseada, respectivamente.
- q y q_d son la orientación actual y deseada, respectivamente, expresada en cuaternión.
- R es la matriz de rotación que transforma las coordenadas al marco base. Es la matriz de rotación contenida dentro de la matriz O_T_EE.
- Del producto $(q^{-1} \cdot q_d)$ solo se utilizan las componentes vectoriales (x, y, z).

2. **Cálculo del Par deseado**

   Aunque el modelo de impedancia se formula en el espacio articular, el error se calcula en el espacio cartesiano. Se utiliza la matriz Jacobiana para transformar este error en los pares articulares deseados.

   El controlador distingue entre el par de la tarea (τ_{task}) y el par del espacio nulo (τ_{ns}). El primero se encarga de corregir el error y el segundo gestiona el espacio nulo del manipulador. El par se calcula de la siguiente manera:

   a) **Par de la tarea (τ_{task})**: Es el encargado de corregir el error en posición y orientación:

   $$\tau_{task} = J^T(-K_c \cdot e - D_c(J \cdot \dot{q})) \tag{3.9}$$

   Donde:

   - J es la matriz jacobiana del manipulador, que relaciona la velocidad de las articulaciones con la velocidad del efector final.
   - K_c es la matriz de Rigidez cartesiana.
   - D_c es la matriz de amortiguamiento cartesiano.
   - $\dot{q}$ son las velocidades actuales de las articulaciones.

   b) **Par del espacio nulo (τ_{ns})**: Es consecuencia de trabajar con un manipulador redundante. Permite calcular la forma de alcanzar una configuración articular con el menor número de giros:

   $$\tau_{ns} = (I - J^T \cdot (J^T)^+) \cdot (K_{ns} \cdot (q_{d_{ns}} - q) - (2 \cdot \sqrt{K_{ns}}) \cdot \dot{q}) \tag{3.10}$$

   Donde:

   - I es la matriz identidad de 7x7.
   - $(J^T)^+$ es la pseudoinversa de la traspuesta de la matriz jacobiana.
   - K_{ns} es la matriz de rigidez del espacio nulo, que define la resistencia al cambio de la configuración deseada de las articulaciones.

   c) **Par total deseado (τ_d)**:

   $$\tau_d = \tau_{task} + \tau_{ns} + C \tag{3.11}$$

   Donde:

- C es el vector de fuerzas de Coriolis.

3. **Saturación de Par**

   Se utiliza para evitar que el par aplicado supere el máximo. La función `saturate` compara el valor del par deseado con el par máximo permitido en cada articulación y, en caso de superarlo, se elige este último como deseado:

   $$\tau_{d_{\text{sat}}} = \text{saturate}(\tau_d, \tau_{J_d}) \tag{3.12}$$

   Este es el par que se comanda a los actuadores para alcanzar una posición y orientación deseada.

Diagrama de nodos y topics del controlador En la figura 3.8 se observa el diagrama de nodos y topics del controlador de impedancia del manipulador. La entrada es la flecha verde, que indica el `equilibrium_pose` deseado al controlador. El `equilibrium_pose` es un tipo de mensaje de ROS que almacena la posición y orientación del efector final del manipulador. Para mover el manipulador a una determinada posición, se debe indicar el`equilibrium_pose` deseado.

Respecto a las salidas, el nodo `joint_state_publisher` se encarga de mandar los comandos del espacio articular al manipulador para hacer efectivo el movimiento. Para ello, publica en el topic `/joint_states`. Respecto a la salida para la ejecución de RViz, este se subscribe al topic `/TF` para poder visualizar el estado actual del manipulador haciendo uso de sus matrices de transformación, que contienen información cartesiana sobre la posición, orientación de cada uno de los eslabones.

3.2.2.2. Controlador de velocidad cartesiano

El controlador de velocidad cartesiana permite comandar directamente velocidades cartesianas al efector final utilizando la interfaz de velocidad cartesiana proporcionada por Franka Emika. Originalmente, este controlador estaba diseñado para ejecutar una secuencia de movimientos en bucle sin entradas externas. Se ha modificado para aceptar la velocidad deseada como entrada y suprimiendo el bucle de movimiento.

Diagrama de entradas y salidas del controlador En la figura 3.9 se muestra el diagrama de entradas y salidas del controlador de velocidad del manipulador. Las entradas del controlador son:

- Velocidad cartesiana deseada: Incluye velocidades lineal y angulares del efector final en el espacio cartesiano.
- Estado actual del robot (`robotState`). Proporciona información sobre la configuración actual del robot, necesaria para garantizar la supervisión del sistema antes de aceptar comandos de velocidad.

La salida del controlador es:

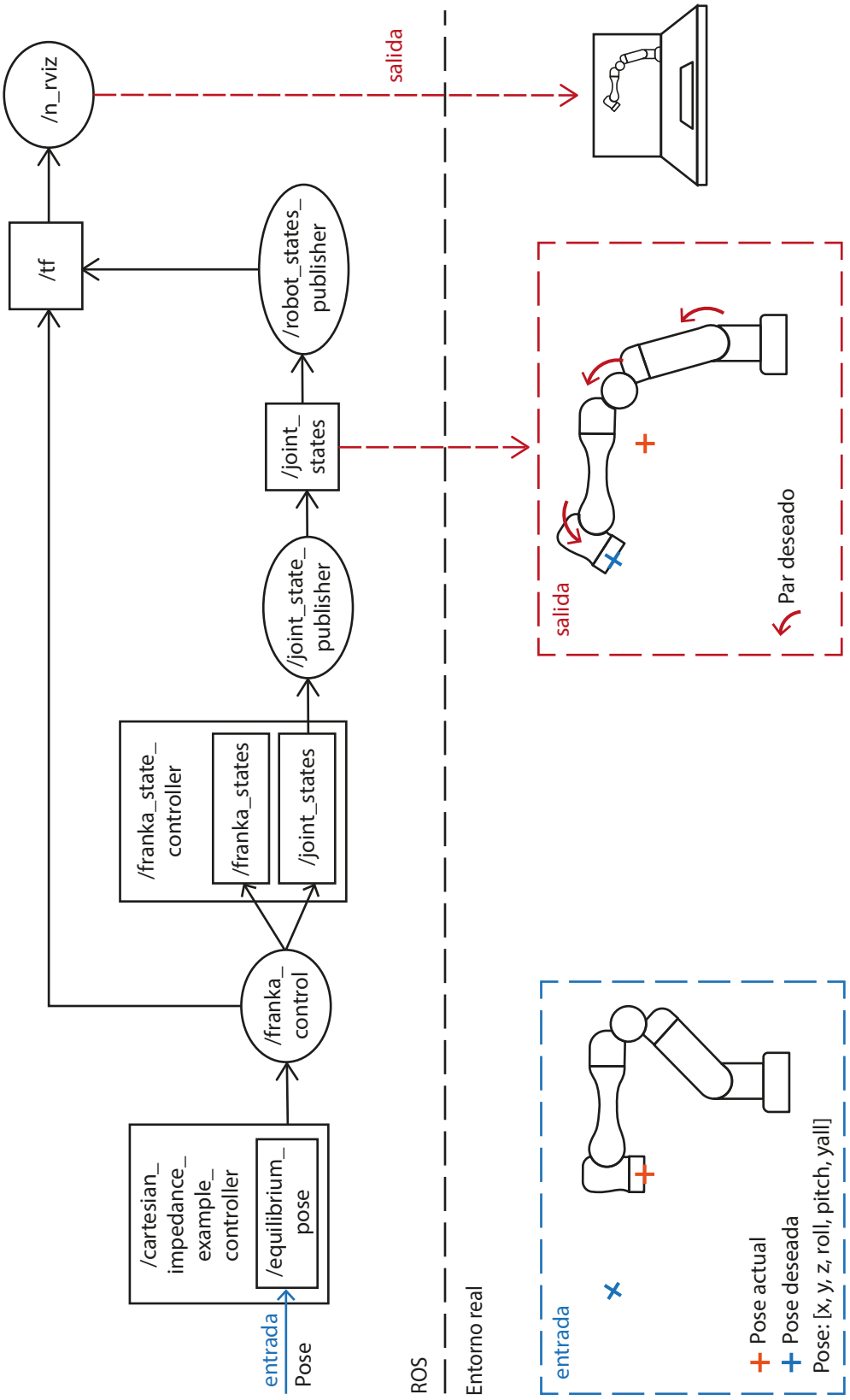


Figura 3.8: Diagrama de nodos de ROS del controlador de impedancia del manipulador

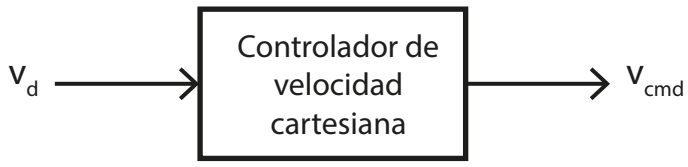


Figura 3.9: Diagrama de entradas y salidas del controlador de velocidad del manipulador

- Vector de velocidad cartesiana aplicada: Las velocidades lineales y angulares aplicadas directamente al efector final en el espacio cartesiano. Este vector tiene la siguiente forma:

$$v_{cmd} = \begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix} \tag{3.13}$$

  Dónde:

  - v_x, v_y y v_z son las velocidades lineales deseadas en las direcciones x, y, z.
  - ω_x, ω_y y ω_z son las velocidades angulares deseadas alrededor de los ejes x, y, z del efector final.

El controlador de velocidad cartesiana permite un control directo del efector final del manipulador en el espacio cartesiano. El robot, a través de su controlador interno, se encarga automáticamente de realizar la cinemática inversa para convertir estos comandos en movimientos articulares. Por tanto, el único tratamiento de la información que hace el controlador es reorganizar el mensaje recibido a un vector de 6 elementos.

Las tareas principales del controlador son:

- Inicialización: Verifica que el robot esté en la posición inicial esperada antes de aceptar comandos. Esto se hace comparando la configuración articular actual con una configuración predefinida.
- Recepción de comandos de velocidad: El controlador se subscribe a un topic de ROS que recibe mensajes de tipo `geometry_msgs::Twist`, los cuales contienen las velocidades lineales y angulares deseadas para el efector final.
- Aplicación de la velocidad: En cada ciclo de actualización, el controlador aplica el comando de velocidad directamente al efector final mediante la función `setCommand`, que ejecuta el movimiento en el espacio cartesiano.

Diagrama de nodos y topics En la figura 3.10, se muestra el diagrama de nodos y topics del controlador de velocidad del manipulador durante su ejecución. El topic `/cmd_franka_vel` es la entrada, dónde se publica la velocidad deseada. El nodo `/franka_control` actúa como puente entre el topic anterior y el topic `/joint_states` dentro del controlador interno del manipulador `/franka_state_controller`. El nodo `joint_state_publisher` publica la velocidad articular deseada, haciendo efectivo el movimiento del manipulador conforme a la entrada.

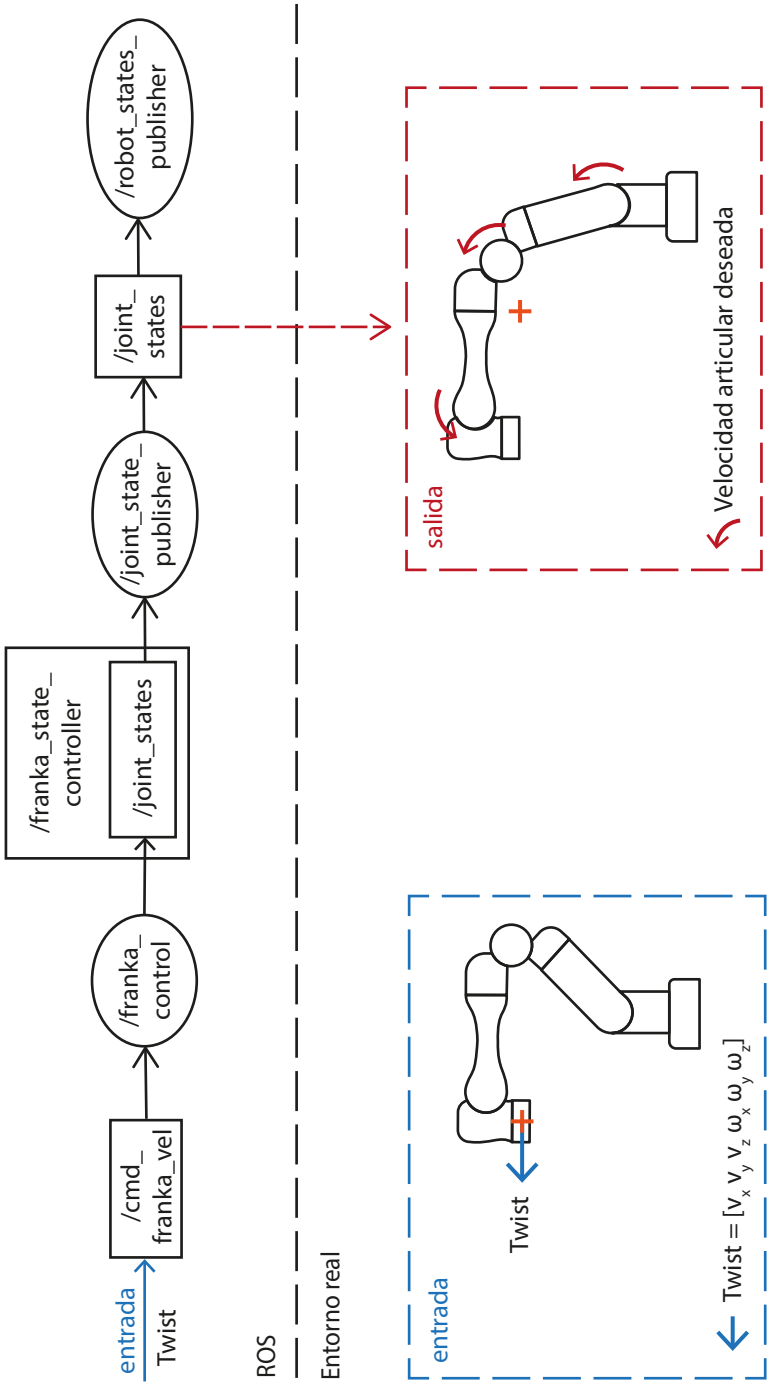


Figura 3.10: Diagrama de nodos y topics del controlador de velocidad del manipulador

3.2.3. Esquema de control de RAFI

Este apartado propone una solución de control conjunta para la base y el manipulador. Esto es conocido como "esquema de control de RAFI". Este esquema es el resultado de la ejecución simultánea del controlador de la base y controlador del manipulador. Se han desarrollado dos esquemas de control distintos como resultado de la integración de los dos controladores del manipulador con el único controlador de la base. Estos esquemas son: Esquema A, para el control de impedancia del manipulador junto con el controlador de la base; y Esquema B para el control de velocidad del manipulador junto con el controlador de la base.

3.2.3.1. Esquema A

Este esquema de control está compuesto por el controlador de velocidad cartesiana de la base y el controlador de impedancia cartesiana del manipulador.

En la figura 3.11 se observa el diagrama de nodos y topics del controlador de la base y el controlador de impedancia del manipulador ejecutándose de manera simultánea. Se observa que el controlador del manipulador acepta una entrada de tipo `geometry_msgs/PoseStamped` con la posición y orientación deseada del efector final. El controlador de la base acepta una entrada de tipo `geometry_msgs/Twist` con información sobre la velocidad lineal en la dirección X e Y y velocidad angular aplicada sobre el eje Z.

3.2.3.2. Esquema B

Este esquema de control está compuesto por el controlador de velocidad cartesiana de la base y el controlador de velocidad cartesiana del manipulador.

En la figura 3.12 se muestra el diagrama de nodos y topics del controlador de la base y del controlador de velocidad del manipulador ejecutándose de manera simultánea. La entrada del controlador del manipulador es del tipo `geometry_msgs/Twist` que contiene información sobre la velocidad lineal en las direcciones X,Y,Z y velocidad angular aplicada sobre los ejes roll, pitch y Yall. Estas velocidades son aplicadas en el efector final. El controlador de la base acepta entradas de velocidad lineal en la dirección X e Y y velocidad angular aplicada sobre el eje Z.

3.3. Interfaz de teleoperación

Esta sección describe el desarrollo de la interfaz de software para la teleoperación de los controladores implementados. Para ello, se han desarrollado distintos paquetes de ROS que contienen nodos de teleoperación encargados de proporcionar una entrada a los controladores en función del estado el mando. Los paquetes desarrollados están basados en un paquete para teleoperación de placas Roboclaw creado por "Mike Purvis", y usado como referencia en el trabajo previo de [21]. En este proyecto se ha modificado ese código de forma sustancial y se ha usado como base para el resto de

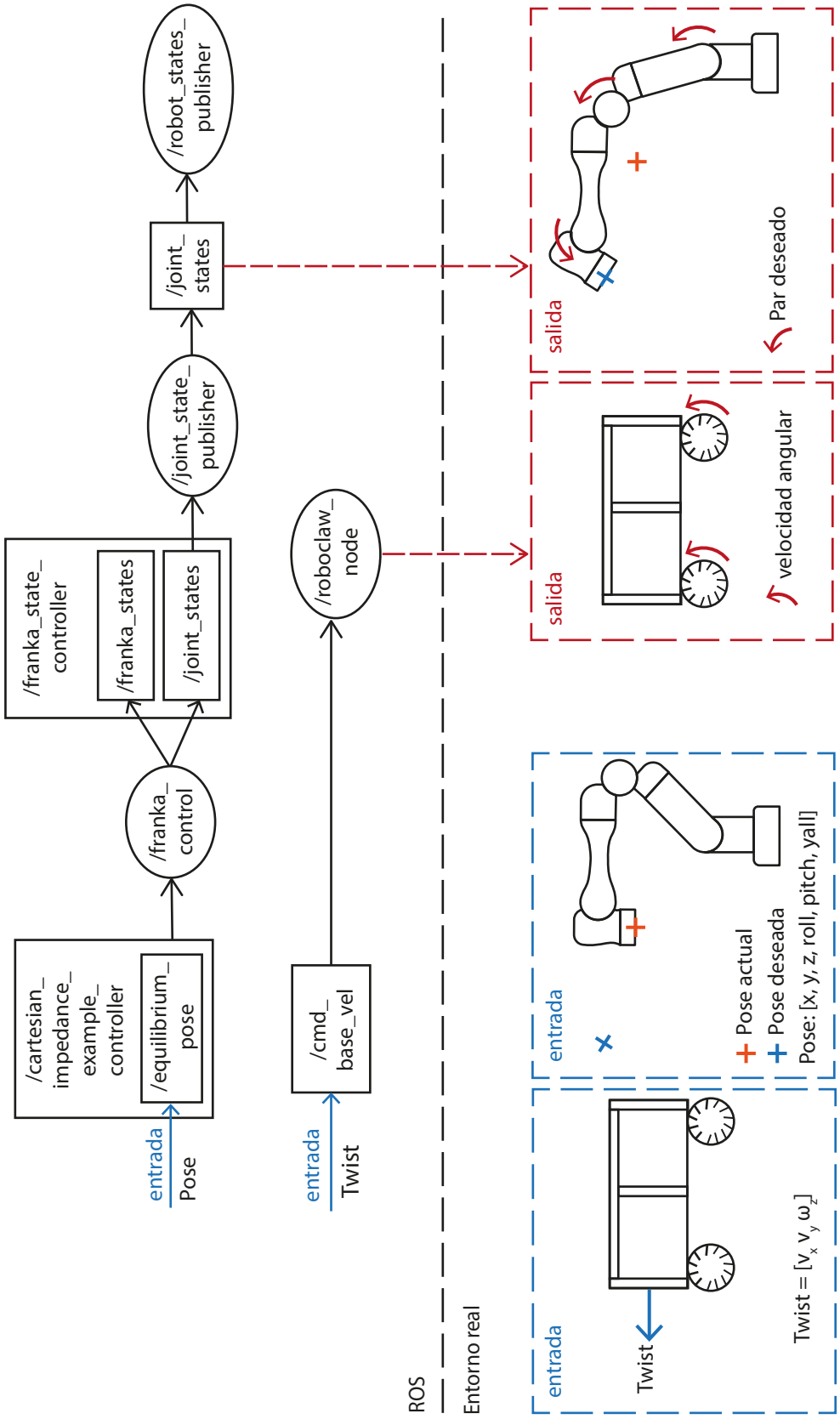


Figura 3.11: Diagrama de nodos y topics del controlador de la base y el controlador de impedancia del manipulador

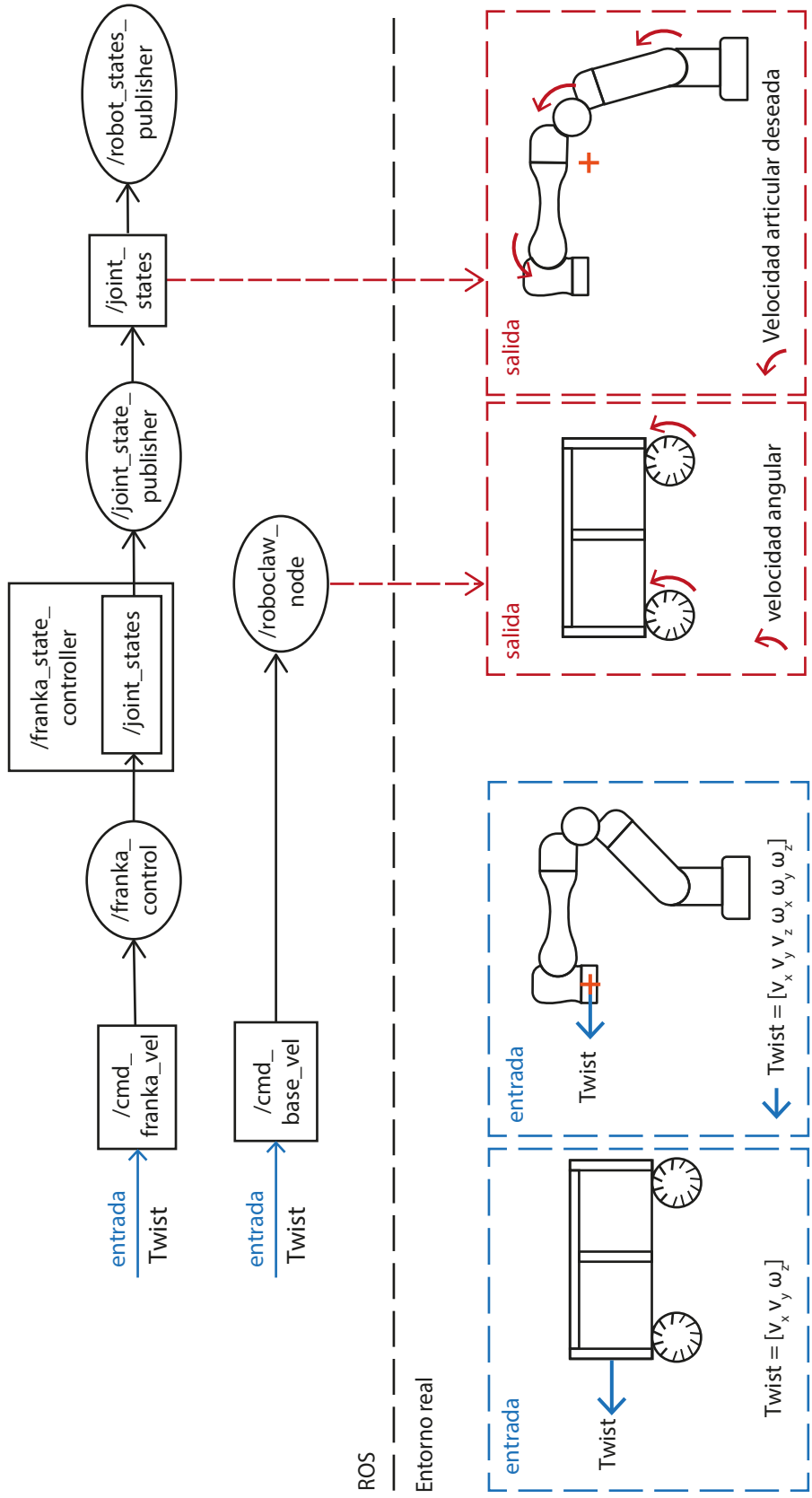


Figura 3.12: Diagrama de nodos y topics del controlador de la base y del controlador de velocidad del manipulador

paquetes de ROS con el objetivo garantizar una estructura homogénea de todos ellos que facilite su comprensión y mantenimiento a largo plazo.

Se ha desarrollado un paquete de ROS para cada controlador que contiene el nodo de teleoperación. La estructura de estos paquetes es la siguiente:

- `src/`: Contiene los archivos fuente del nodo en C++.
- `include/`: Contiene los archivos de cabecera necesarios por los archivos fuente.
- `config/`: Contiene el archivo de configuración `joy.config.yaml` que relaciona botones y ejes del mando con variables utilizadas por los archivos fuente. Al final de cada apartado, se especifica una tabla que describe la relación entre los ejes y botones del mando y las acciones que ejecutan en el robot correspondiente.
- `launch/`: Contiene los archivos de lanzamiento del nodo. Este archivo inicializa el driver del mando, el nodo de teleoperación y el controlador correspondiente.
- `CMakeLists.txt`: Especifica las instrucciones de compilación del paquete.
- `package.xml`: Define las dependencias y librerías necesarias para la compilación y ejecución del paquete.

A continuación, se describen los paquetes encargados de la teleoperación de la base y del manipulador.

3.3.1. Teleoperación de la base

El objetivo de la teleoperación consiste en utilizar el mando como interfaz para introducir comandos de velocidad al controlador de la base. El paquete `joy_base_control` contiene un nodo de teleoperación, `/teleop_twist_node` encargado de enviar comandos de velocidad al controlador en función del estado del mando.

En la figura 3.13 se muestra el diagrama de nodos y topics asociados a la teleoperación del controlador de la base. El nodo `/teleop_twist_node` se subscribe al topic `/joy` para obtener mensajes de tipo `sensor_msgs/Joy`. Tras procesarlos, publica mensajes del tipo `Twist` en el topic `cmd_base_vel`.

El nodo de teleoperación está compuesto por dos archivos archivos fuente:

- `teleop_twist_node.cpp`: Contiene la función principal (`main`) del programa y se encarga de inicializar el nodo ROS y crear una instancia de la clase `TeleopTwistJoy`, que gestiona la teleoperación de la base mediante el mando.
- `teleop_twist_joy.cpp`: Define las funciones necesarias para la teleoperación, incluyendo las subscripciones y publicaciones que realiza el nodo. También describe el comportamiento del bucle de control.

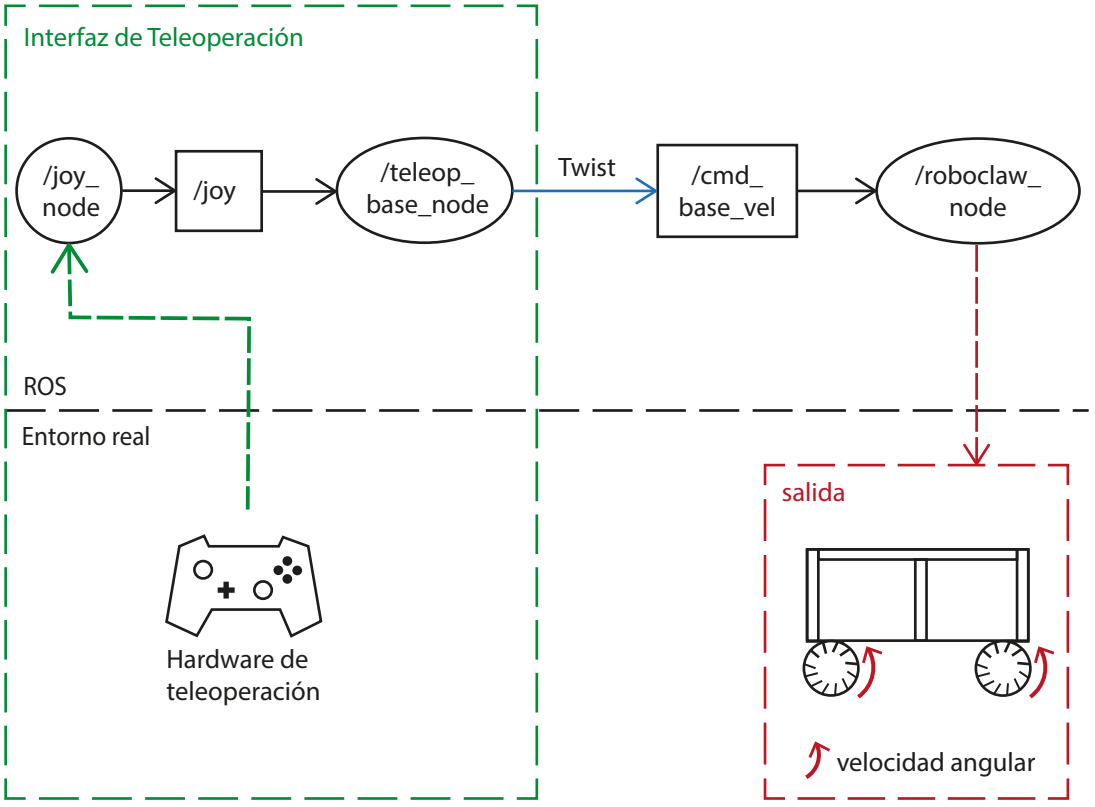


Figura 3.13: Diagrama de nodos y topics de la teleoperación del controlador de la base

Un ejemplo del código de subscripción y publicación es:

```
1          ros::Subscriber joy_sub;
2          ros::Publisher cmd_vel_pub;
```

Este archivo contiene las siguientes funciones miembro:

- `printTwistInfo()`: Formatea e imprime el contenido del tipo `Twist`.
- `ModifyVelocity()`: Modifica la escala de la velocidad lineal y angular. Esta escala permite ajustar el rango de velocidad que se puede comandar desde el mando. Esta escala es la misma para velocidad lineal y angular.
- `joyCallback()`: Define el comportamiento del bucle de teleoperación. El diagrama de decisión del bucle se muestra en la figura 3.14. Se evalúa si está habilitado el movimiento y si se desea modificar la escala de velocidad. En caso afirmativo, se llama a la función `modifyVelocity()`. Si está habilitado el movimiento, pero no se desea modificar la escala, se calcula la velocidad a comandar en función de los valores de los ejes y la escala. Si no está habilitado el movimiento, la velocidad se establece en 0. Antes de reiniciar el bucle, se publica la velocidad y se imprime por consola.

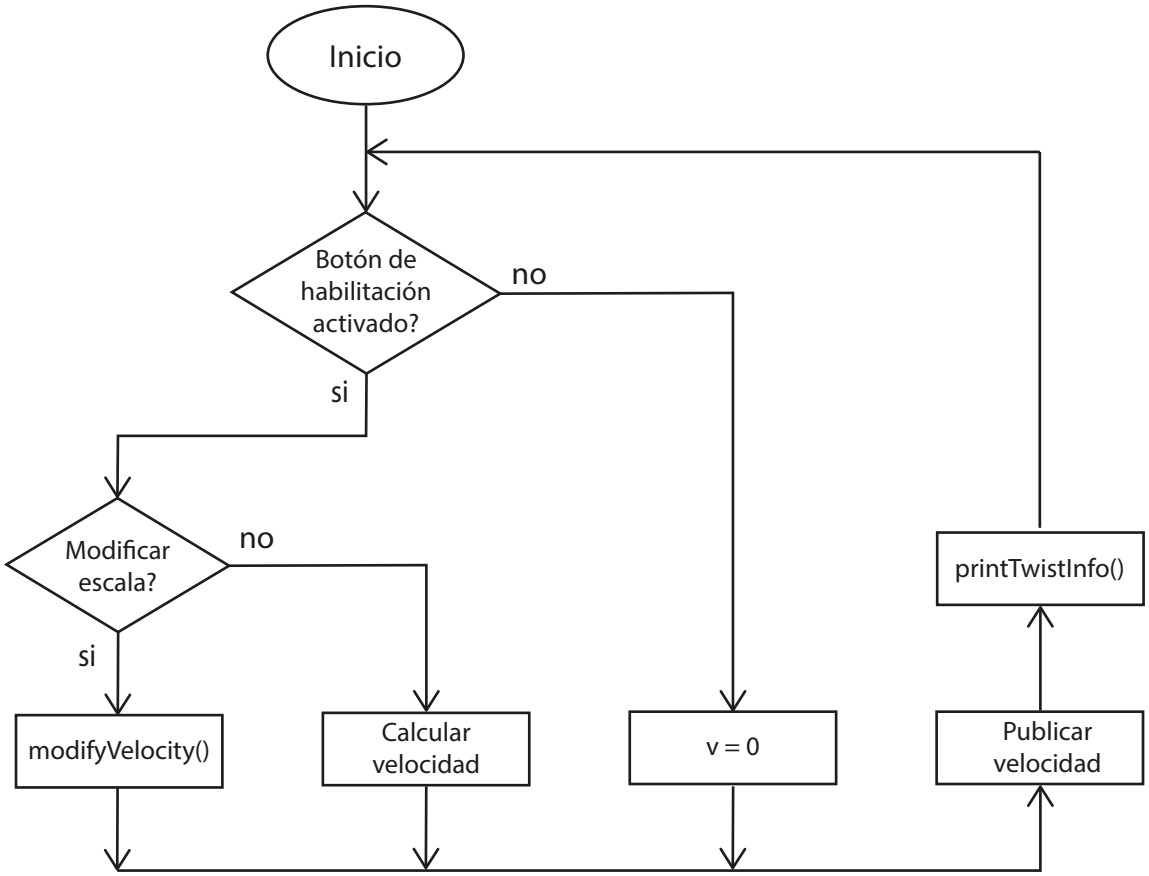


Figura 3.14: Diagrama de decisión del bucle de teleoperación de la base

La tabla 3.1 detalla que eje se encarga del control de la posición y orientación en cada componente, así como el botón de habilitación de movimiento y modificación de escala.

Tabla 3.1: Mapa de ejes y botones del mando para teleoperación del control de la base

Funcionalidad	**Eje/Botón**	**Descripción**
Activar control	Botón B	Habilita el movimiento
Control de V_x	Eje JLy	Velocidad lineal en el eje X del robot
Control de V_y	Eje JLx	Velocidad lineal en el eje Y del robot
Control de ω_z	Eje JRy	Velocidad angular en el eje Z del robot
Incrementar escala	Botón +	Incrementa la escala de velocidad
Decrementar escala	Botón -	Decrementa la escala de velocidad

En resumen, la teleoperación de la base se implementa a través del paquete de ROS `joy_base_control`, que contiene el nodo de teleoperación necesario para convertir el mensaje del mando en comandos de entrada al controlador de la base.

3.3.2. Teleoperación del manipulador

A continuación, se aborda la teleoperación de los dos controladores del manipulador: controlador de impedancia cartesiana y controlador de velocidad cartesiana. Se ha desarrollado un paquete distinto para cada uno de ellos.

3.3.2.1. Teleoperación del manipulador en impedancia

El objetivo de la interfaz de teleoperación es utilizar el mando para introducir comandos de `equilibrium_pose` al controlador de impedancia del manipulador. Para ello, se ha desarrollado el paquete `joy_franka_control` que contiene el nodo de teleoperación del controlador.

La figura 3.15 muestra el diagrama de nodos y topics involucrados en la teleoperación del controlador. El nodo de teleoperación, `teleop_franka_joy`, se subscribe al topic que contiene los mensajes del mando y publica en el topic `/equilibrium_pose` dentro del controlador de impedancia. Además, se subscribe al topic `/franka_states` de dentro del controlador `/franka_state_controller` para obtener el elequilibrium_pose inicial tras iniciar el sistema por primera vez o tras una parada donde se ha modificado de forma manual su posición u orientación.

El nodo de teleoperación se implementa mediante dos archivos fuente:

- `teleop_franka_node.cpp`: Contiene el (`main`) del programa y se encarga de inicializar el nodo ROS y crear una instancia de la clase `TeleopFrankaJoy`, que gestiona la teleoperación del manipulador en impedancia mediante el mando.

- `teleop_franka_joy.cpp`: Define las funciones, subscripciones, publicaciones y el comportamiento del bucle de control

  Este archivo contiene las siguientes funciones miembro:

  - `printEquilibriumPoseInfo()`: Función que imprime de formateada el contenido de un mensaje del tipo `EquilibriumPose`.
  - `obtainEquilibriumPose()`: Calcula el `equilibrium_pose` en función de la matriz de transformación *O_T_EE* obtenida del topic `/franka_states`. Su componente `O_T_EE` alberga la matriz de transformación 4x4 del efector final respecto al sistema 0 del manipulador contiene información sobre la posición y orientación. Se usan las funciones `translation()` y `rotation()` de la biblioteca Eigen para extraer la información de traslación y rotación de esta matriz y posteriormente convertirlo al tipo `PoseStamped` [10].
  - `sendCmdPositionMsg()`: Calcula la nueva posición como incremento de la posición actual en función de la escala, incremento de tiempo y el valor del eje. A continuación se muestra un ejemplo para la componente X:

[10] *Librería Eigen*, URL: https://eigen.tuxfamily.org/index.php?title=Main_Page, Acceso a la librería Eigen para c++, Fecha de acceso: 5 de junio de 2024. Para usar la biblioteca Eigen se debe incluir dentro las librerías de ROS, ubicándola dentro de la carpeta `/opt/ros/noetic/lib`.

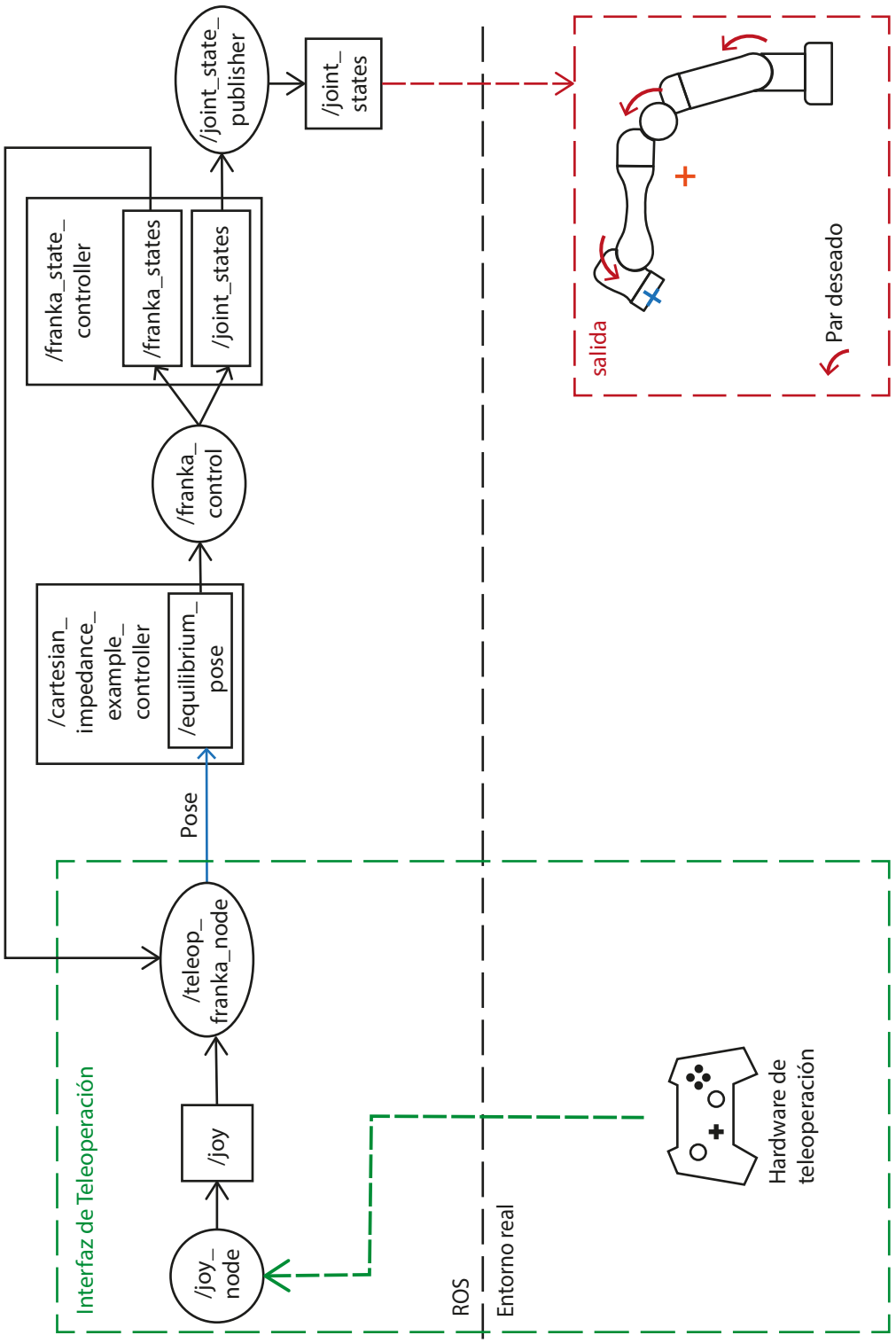


Figura 3.15: Diagrama de nodos y topics de la teleoperación del controlador de impedancia del manipulador

```
increment_position.x = Delta_t * position_max_vel * getVal(
    joy_msg, axis_position_map, "x");
equilibrium_pose.pose.position.x += increment_position.x;
```

Se aplican limites para que el `equilibrium_pose` no se salga del espacio cartesiano de trabajo.

Por último, se aplican limites para que el `equilibrium_pose` no se salga del espacio cartesiano de trabajo y se imprime y publica la posición.

- `sendCmdOrientationMsg()`: El mando envía ángulos Euler, sin embargo, para el cálculo de la nueva matriz de rotación se trabaja con cuaterniones. [11] Los ángulos Euler se convierten al tipo `tf2::Quaternion` usando la función `setRPY` de la librería tf2.

  Se aplica la rotación deseada a la orientación actual:

$$q_{\text{new}} = q_{\text{rot}} \cdot q_{\text{orig}} \tag{3.14}$$

  Se normaliza el cuartenión con la función `normalize()` de tf2:

```
q_new.normalize()
```

  Se convierte el cuaternión al tipo `geometry_msgs/Quaternion` mediante la función `convert()` de tf2:

```
tf2::convert(q\_tf2, q\_geometry\_msg)
```

  Por último, se imprime y publica la orientación.

- `ModifyVelocity()`: Modifica la escala de incremento del `equilibrium_pose`.
- `joyCallback()`: Define el comportamiento del bucle de teleoperación. La figura 3.16 muestra el diagrama de decisión del bucle de teleoperación del controlador.

  En primer lugar, se evalúa si está habilitado el movimiento lineal y se evalúa si se desea modificar la escala. En caso afirmativo, se llama a la función `modifyVelocity()` y se modifica la escala lineal. Si está habilitado el movimiento lineal pero no se desea modificar la escala, se llama a la función `SendCmdOrientation()` y se reinicia el bucle de control.

  En caso de no estar habilitado el movimiento lineal pero si el movimiento angular, se evalúa si se desea modificar la escala. En caso afirmativo, se llama a la función

[11] ROS consta de dos tipos de cuaterniones: el tipo `tf2::Quaternion` es usado para cálculos y el tipo `geometry_msgs/Quaternion` se usa al comandar al manipulador. Diferentes tipos de cuaterniones no pueden ser operados pero si convertidos entre sí.

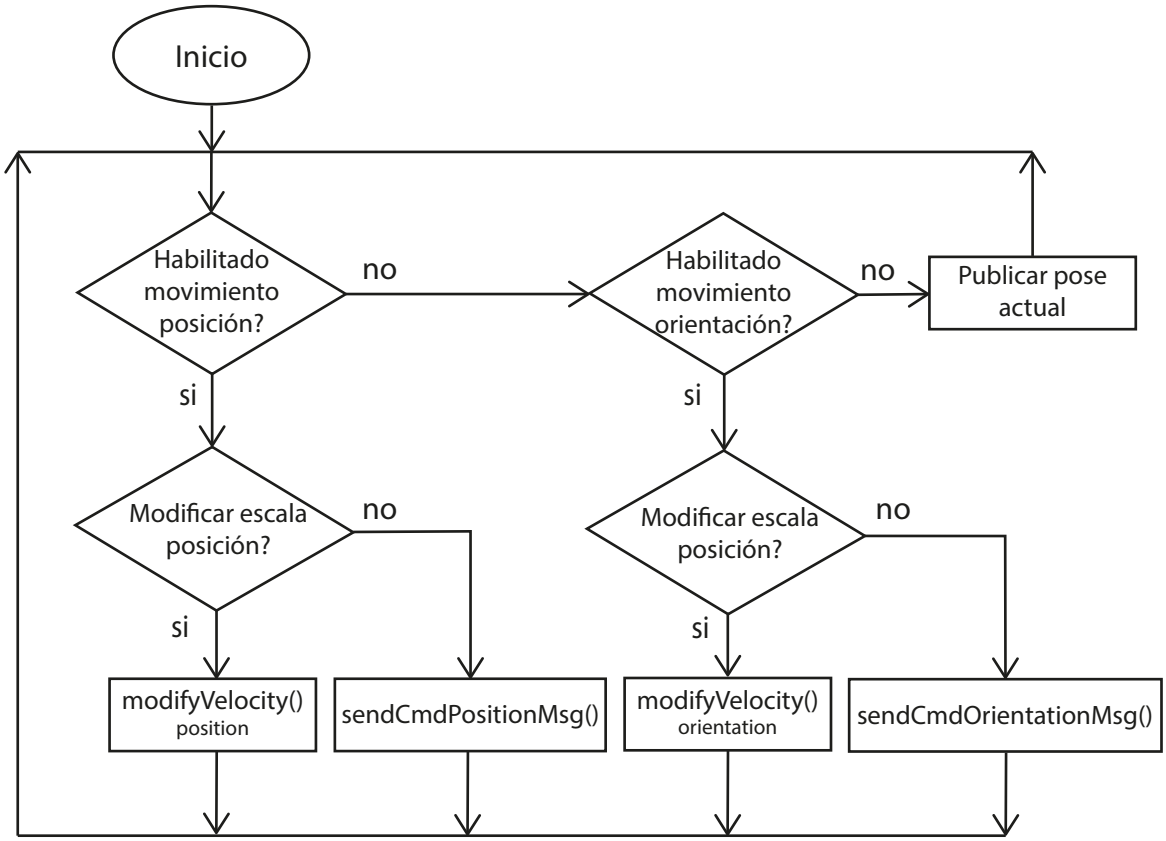


Figura 3.16: Diagrama de decisión del bucle de teleoperación del controlador de impedancia

`modifyVelocity()` para modificar la escala angular. En caso negativo, se llama a la función `SendCmdOrientation()` y se reinicia el bucle.

Si no está habilitado el movimiento lineal ni angular, se publica el `equilibrium_pose` actual y se reinicia el bucle.

La tabla 3.2 detalla que eje se encarga del control de la posición y orientación en cada componente, así como los botones de habilitación de movimiento e incremento y decremento de la escala.

En resumen, la teleoperación del controlador de impedancia cartesiana se basa en comandar el `equilibrium_pose` deseado a través del mando gracias al nodo de teleoperación implementado en el paquete de ROS.

3.3.2.2. Teleoperación del manipulador en velocidad

El objetivo de la interfaz de teleoperación es utilizar el mando para introducir comandos de velocidad al controlador de velocidad cartesiana del manipulador. Se ha desarrollado un paquete, `teleop_franka_vel_joy` que contiene un nodo de teleoperación encargado de traducir la información.

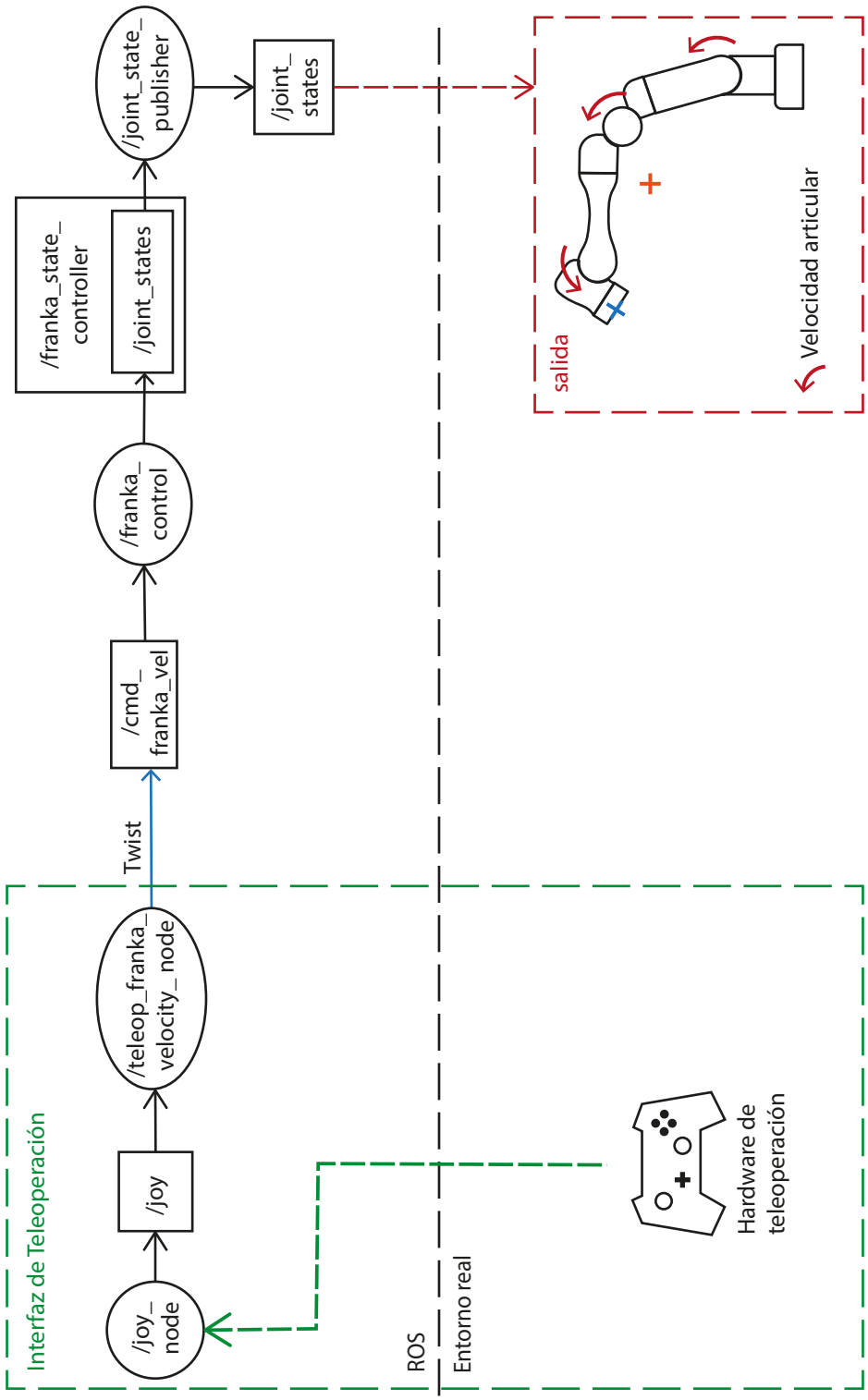


Figura 3.17: Diagrama de nodos y topics durante la teleoperación del controlador de velocidad del manipulador

Tabla 3.2: Mapa de ejes y botones del mando para teleoperación del control de impedancia cartesiana del manipulador

Funcionalidad	Eje/Botón	Descripción
Activar movimiento lineal	Gatillo LB	Habilita el control de posición
Activar movimiento angular	Gatillo LR	Habilita el control de orientación
Control de posición en el eje X	Eje JLy	Posición en el eje X del robot
Control de posición en el eje Y	Eje JLx	Posición en el eje Y del robot
Control de posición en el eje Z	Eje JRy	Posición en el eje Z del robot
Control de orientación en el eje X	Eje JLy	Orientación en el eje X del robot
Control de orientación en el eje Y	Eje JLx	Orientación en el eje Y del robot
Control de orientación en el eje Z	Eje JRy	Orientación en el eje Z del robot
Incrementar escala	Botón +	Incrementa la escala
Decrementar escala	Botón -	Decrementa la escala

La figura 3.17 muestra el diagrama de nodos y topics involucrados en la teleoperación del controlador de velocidad del manipulador. El nodo de teleoperación, `teleop_franka_vel_joy`, se subscribe al topic que contiene los mensajes del mando y publica en el topic `/cmd_franka_vel` un mensaje del tipo `Twist`.

El nodo de teleoperación se implementa mediante dos archivos fuente:

- `teleop_franka_vel_node.cpp`: Contiene el (`main`) del programa y se encarga de inicializar el nodo ROS y crear una instancia de la clase `TeleopFrankaJoy`, que gestiona la teleoperación del manipulador en velocidad cartesiana mediante el mando.

- `teleop_franka_vel_joy.cpp`: Define las funciones, subscripciones, publicaciones y el comportamiento del bucle de control. Este archivo contiene las siguientes funciones miembro:

  - `printTwistInfo()`: Imprime de manera formateada el contenido del mensaje `Twist`.
  - `sendCmdVel()`: En función de los valores máximos admisibles por el controlador, limita el valor de velocidad, aceleración y tirón translacional y rotacional. Posteriormente, imprime y publica este mensaje.

    Para evitar errores en el control, conocidos como discontinuidades o reflejos, la entrada al controlador debe ser suficientemente suave y con valores máximos limitados. Libfranka incluye un limitador de tasa, LimitRate, encargado de modificar la señal de entrada para adecuarla a los límites del manipulador excepto para los límites articulares después de la cinemática inversa.

    De manera experimental, se ha comprobado que la reducción de un orden de magnitud los valores máximos impuestos por el manipulador garantiza la estabilidad del controlador, sin necesidad de utilizar técnicas adicionales de suavizado de la señal.

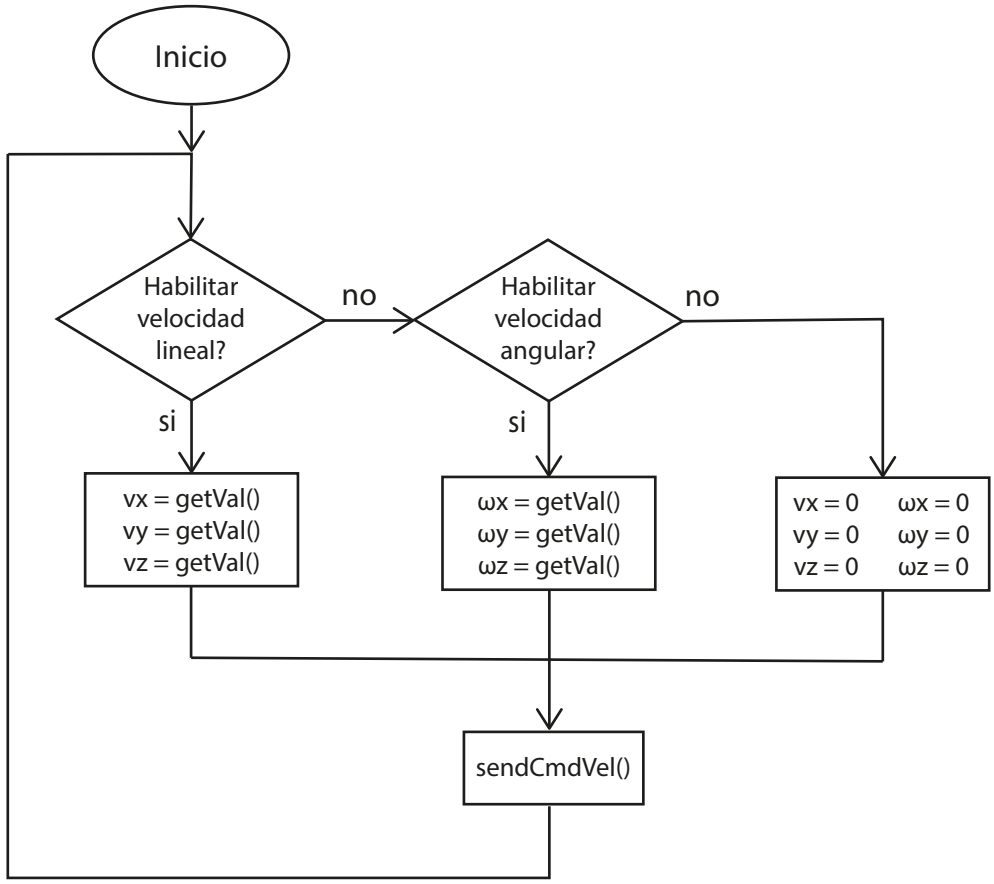


Figura 3.18: Diagrama de decisión del bucle de teleoperación del controlador de velocidad del manipulador

- `joyCallback()`: Define el comportamiento del bucle de teleoperación. La figura 3.18 muestra el diagrama de decisión del bucle de teleoperación del controlador de velocidad del manipulador.

  En primer lugar, se evalúa si está habilitado el movimiento lineal. En caso afirmativo, las velocidades lineales son asignadas en función del valor del eje del mando. En caso negativo, se evalúa si está habilitado el movimiento lineal. En caso afirmativo, las velocidades angulares son asignadas en función del valor del eje del mando. Si no está habilitado el movimiento lineal y angular, la velocidad será igual a cero. Antes de reiniciar el bucle de control, se llama a la función `sendCmdVel`.

La tabla 3.3 detalla que eje se encarga del control de la posición y orientación en cada componente, así como los botones de habilitación de movimiento.

En resumen, la teleoperación del controlador de velocidad cartesiana se basa en introducir comandos de velocidad deseada al controlador a través del mando mediante el uso del nodo de teleoperación como traductor de los mensajes de ROS.

Tabla 3.3: Mapa de ejes y botones del mando para teleoperación del control de velocidad del manipulador

Funcionalidad	Eje/Botón	Descripción
Activar control V	Gatillo LB	Habilita el control de velocidad lineal
Activar control ω	Gatillo RB	Habilita el control de velocidad angular
Control de V_x	Eje JLy	Velocidad lineal en el eje X del robot
Control de V_y	Eje JLx	Velocidad lineal en el eje Y del robot
Control de V_z	Eje JRy	Velocidad lineal en el eje Z del robot
Control de ω_x	Eje JLy	Velocidad angular en el eje X del robot
Control de ω_y	Eje JLx	Velocidad angular en el eje Y del robot
Control de ω_z	Eje JRy	Velocidad angular en el eje Z del robot

3.3.3. Teleoperación del esquema de control de RAFI

Esta sección describe la teleoperación de los esquemas de control de RAFI. Consiste en la ejecución simultánea de los nodos de teleoperación de la base y del manipulador junto con toda la estructura de control. La asignación de botones para cada nodo de teleoperación se ha diseñado para que no interfieran durante la ejecución simultánea de ambos. Esta implementación se encuentra dentro del paquete `rafi_launch_files`. Este paquete no contiene código fuente, sino diferentes archivos de lanzamiento que ejecutan diversas combinaciones de nodos de teleoperación y control. Estos archivos:

- `rafi_impedance.launch`: Lanza el nodo de driver del mando, el controlador de impedancia del manipulador junto a su nodo de teleoperación y el controlador de la base junto con su nodo de teleoperación.
- `rafi_teleop_base.launch`: Lanza el nodo de driver del mando y el controlador de la base junto con su nodo de teleoperación.
- `rafi_teleop_impedance_franka.launch`: Lanza el nodo de driver del mando y el controlador de impedancia del manipulador junto a su nodo de teleoperación.
- `rafi_teleop_velocity_franka.launch`: Lanza el nodo de driver del mando y el controlador de velocidad del manipulador junto a su nodo de teleoperación.
- `rafi_teleop_velocity.launch`: Lanza el nodo de driver del mando, el controlador de velocidad del manipulador junto a su nodo de teleoperación y el controlador de la base junto con su nodo de teleoperación.

Estos archivos de lanzamiento son el resultado de una programación estructurada, donde se ha utilizado el código `<include file = "archivo.launch"/>` para incluir archivos de lanzamiento previamente desarrollados desde un único archivo.

3.3.3.1. Teleoperación del Esquema A

Este esquema corresponde a la teleoperación del controlador de impedancia cartesiana del manipulador ejecutado de manera simultánea con la teleoperación del controlador de la base.

La figura 3.19 muestra el diagrama de nodos y topics durante la teleoperación del esquema de impedancia de RAFI. Los nodos de teleoperación del manipulador y de la base se subscriben al topic `/joy`, procesan la información y publican en la entrada de sus respectivos controladores.

3.3.3.2. Teleoperación del Esquema B

Este esquema se refiere a la teleoperación del controlador de velocidad cartesiana del manipulador junto con la teleoperación del controlador de la base.

La figura 3.20 ilustra el diagrama de nodos y topics durante la teleoperación del esquema de velocidad cartesiana de RAFI. Al igual que en el esquema anterior, los nodos de teleoperación del manipulador y de la base se subscriben al topic `/joy`. Luego procesan la información y publican el mensaje `Twist` en la entrada de sus respectivos controladores.

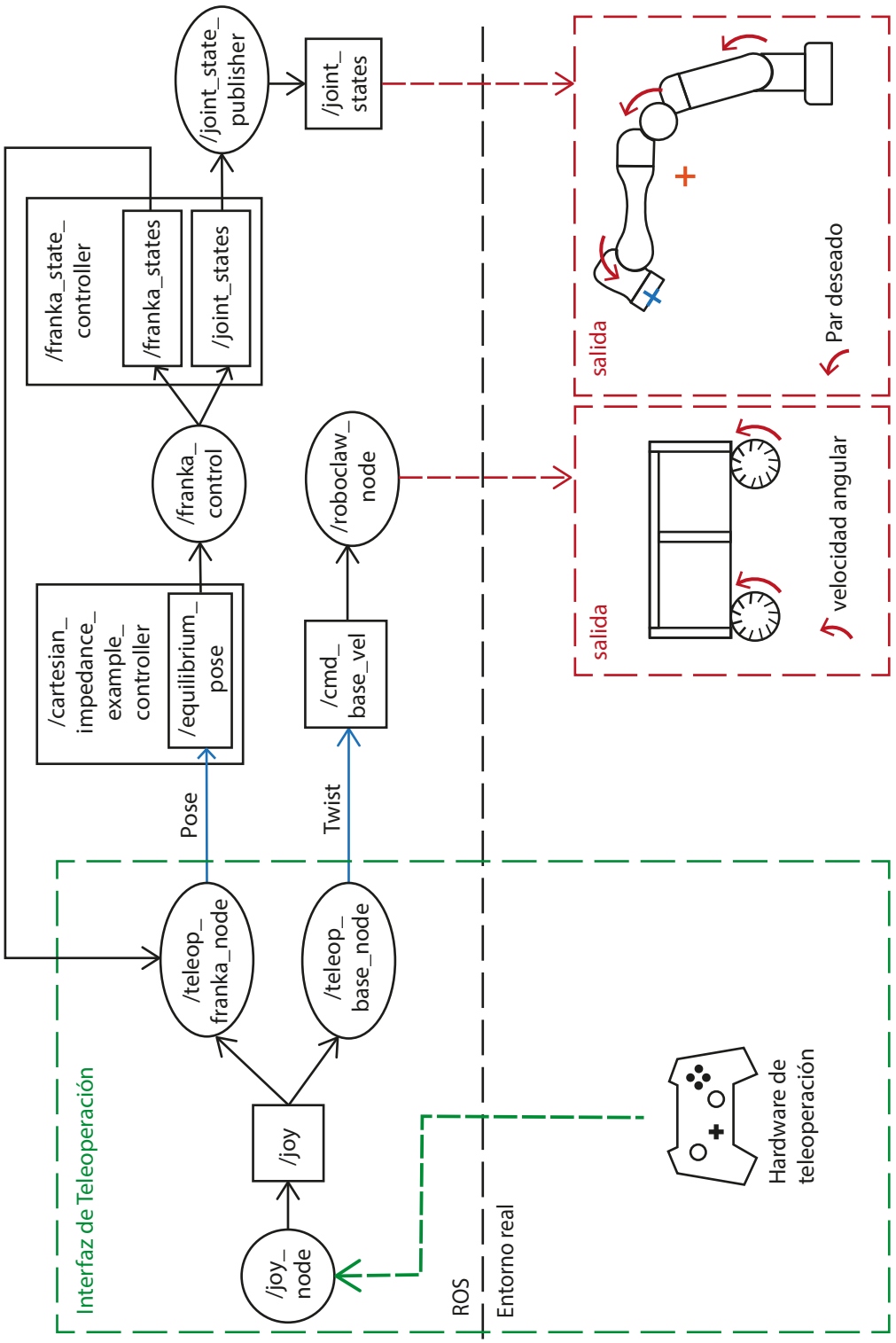


Figura 3.19: Esquema de nodos y topics de la teleoperación del esquema de impedancia de RAFI

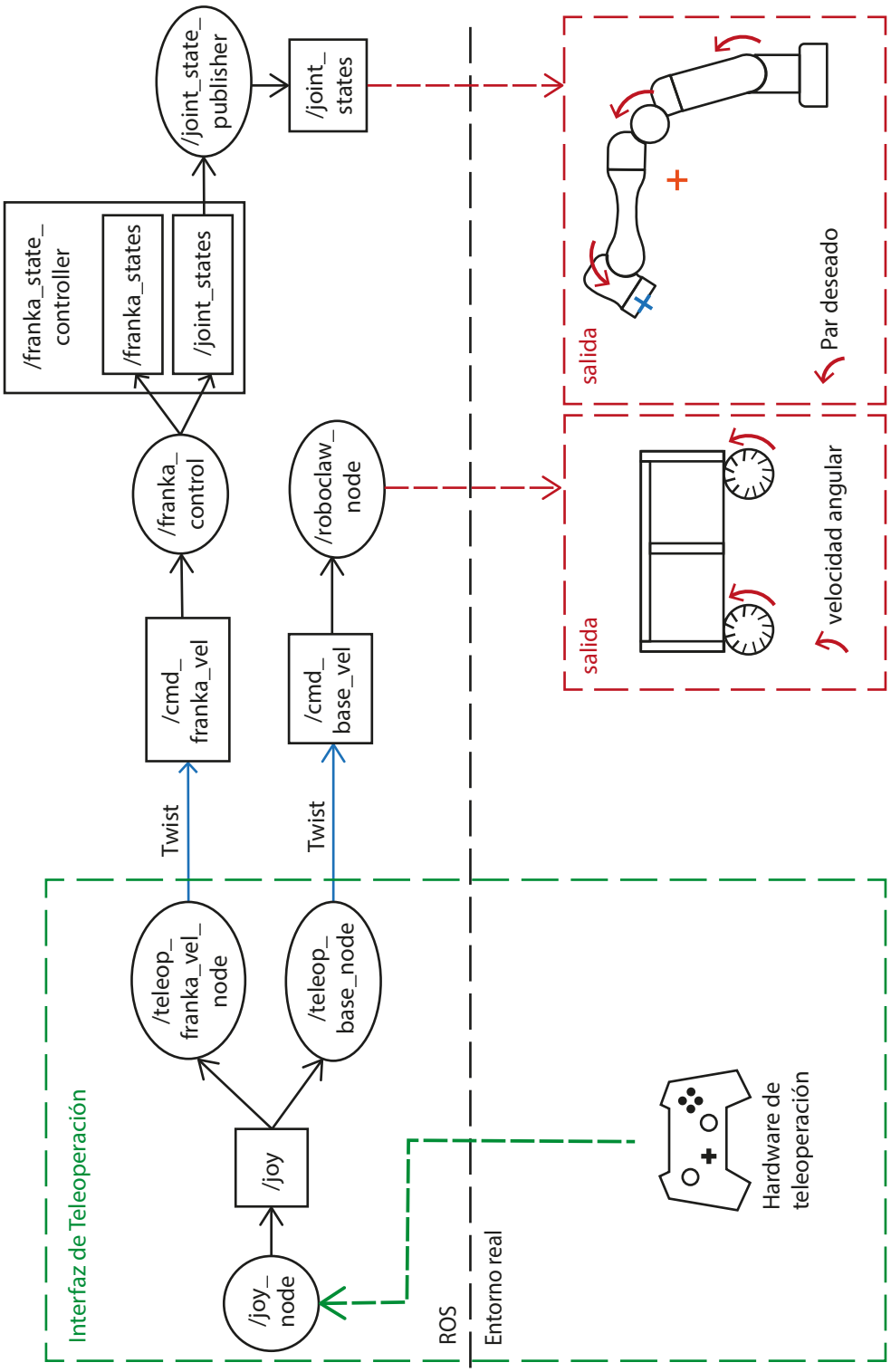


Figura 3.20: Esquema de nodos y topics de la teleoperación del esquema de velocidad cartesiana de RAFI

CAPÍTULO 4

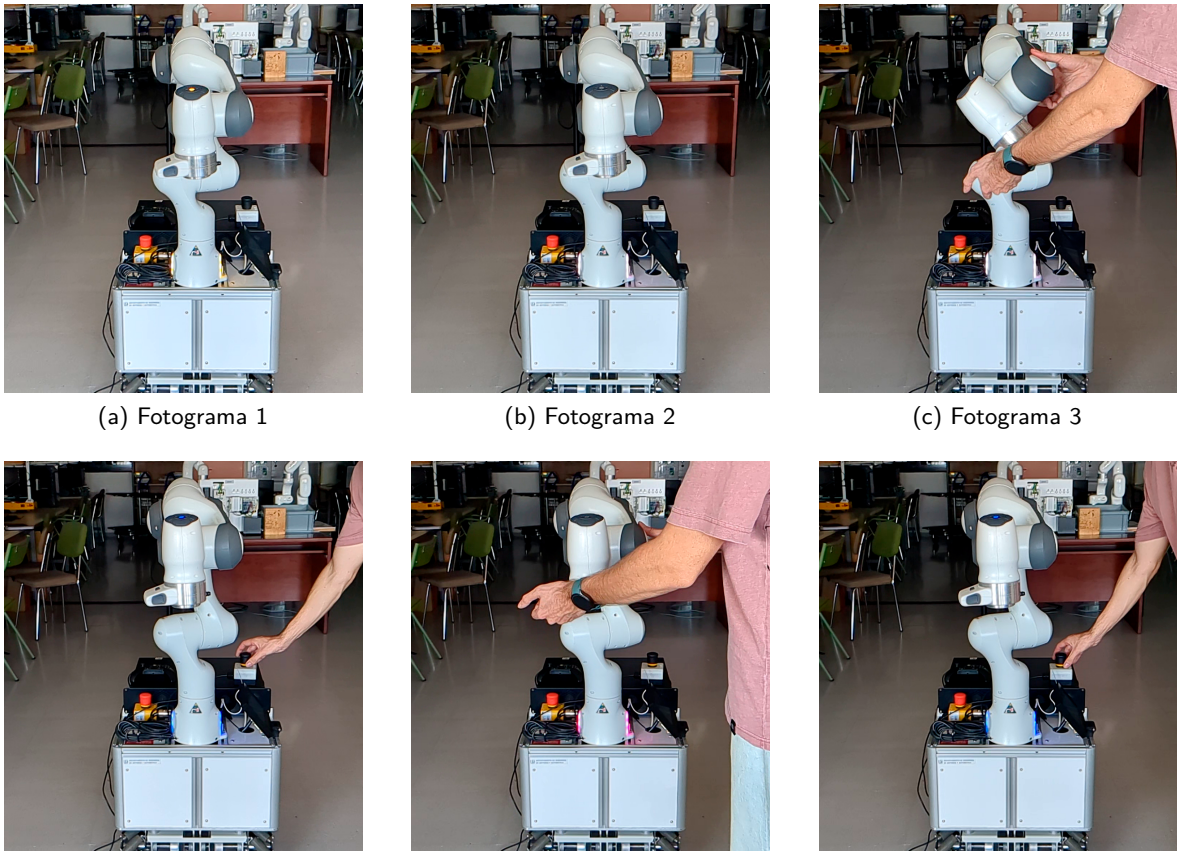
Experimentos y resultados

Contenido

Este capítulo pretende analizar el funcionamiento de las soluciones desarrolladas en el capítulo 3. Cada experimento consta de un vídeo grabado durante su ejecución. Se aportan los fotogramas más significativos. Además, se incluyen los comandos necesarios para la puesta en marcha de cada uno de los experimentos. El enlace del vídeo de los experimentos se encuentra disponible en 3.1.

4.1. Experimento 1. Preparación del manipulador para el movimiento mediante terminal

El objetivo de este experimento es comprobar el funcionamiento del desbloqueo de los frenos del manipulador y activación del modo FCI a través de la terminal. Para ello se utilizará el paquete de ROS `franka_lock_unlock`. Se debe ejecutar el siguiente alias:



(a) Fotograma 1 (b) Fotograma 2 (c) Fotograma 3

(d) Fotograma 4 (e) Fotograma 5 (f) Fotograma 6

Figura 4.1: Secuencia de fotogramas que muestra el desbloqueo del manipulador Franka

```
$ franka_unlock = 'source ~/ros_projects/franka_ws/catkin_ws/devel/
    setup.sh && rosrun franka_lock_unlock __init__.py -u -l -w -r -
    p -c 172.16.0.2 RAFI rafi_ISA!'
```

La figura 4.1 muestra el proceso de desbloqueo del manipulador mediante terminal y la casuística asociada. El fotograma 1 muestra el manipulador tras su encendido, la luz amarilla indica que los frenos están activados. A continuación, se ejecuta el comando `$ franka_unlock`. El fotograma 2 muestra como cambia el color del LED del manipulador a blanco. Esto ocurre porque la seta de habilitación de movimiento se encuentra pulsada y el manipulador se encuentra en el modo de interacción segura. El usuario es capaz de modificar la pose del manipulador pulsando los botones de hombre muerto del efector final, tal y como muestra el fotograma 3. El fotograma 4 muestra al usuario desbloqueando la seta de habilitación de movimiento del manipulador, la luz azul indica que el manipulador se encuentra listo para iniciar movimiento en cualquier momento. En el caso de intentar modificar la pose del manipulador mediante los botones del efector final, el LED se pondrá en color rosa indicando un error, tal y como muestra el fotograma 5. Para solucionarlo, se

debe bloquear y desbloquear la seta de habilitación de movimiento, tal y como se muestra en el fotograma 6. Para bloquear los frenos del manipulador, se debe hacer matar el proceso mediante la combinación de teclas ctrl+c en la terminal en la que se está ejecutando. La ejecución continua de este proceso permite la persistencia del desbloqueo del manipulador, de forma que en el momento que se desee bloquear, se debe pulsar la combinación ctrl+c en la terminal donde se ejecuta este proceso. Esta método garantiza que, en caso de que se apague el PC integrado o se cierre la terminal accidentalmente, el manipulador se bloquee.

4.2. Experimento 2. Teleoperación del controlador de la base

El objetivo de este experimento es comprobar el funcionamiento del controlador de la base mediante su teleoperación. Se espera que la plataforma se mueva conforme los comandos de entrada del mando.

Para la puesta en marcha del experimento se ha utilizado el archivo de lanzamiento relacionado con la teleoperación de la base contenido dentro del paquete `rafi_launch_files`:

```
1          roslaunch rafi_launch_files rafi_teleop_base.launch
```

La figura 4.2 muestra la teleoperación del controlador de la base mediante el mando. Los fotogramas 1-2 muestran el movimiento en la dirección Y. Los fotogramas 2-4 el movimiento angular. Por último, los fotogramas 4-6 muestran el movimiento en la dirección X. La plataforma se mueve según le indica el operador mediante el mando, luego, el resultado de este experimento es satisfactorio.

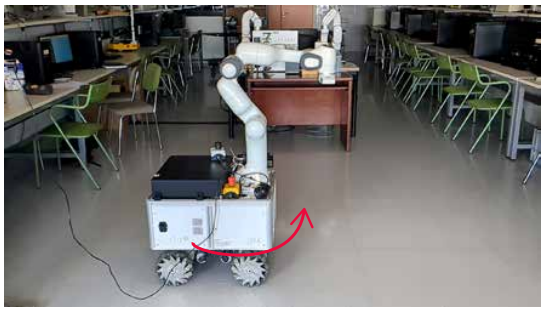
La figura 4.3 muestra otro experimento realizado durante la teleoperación de la base cuyo objetivo es demostrar el funcionamiento de la modificación de la escala de velocidad. Se muestra la relación entre los comandos del mando y la velocidad de la base. Durante este experimento, la base se ha movido en el eje X positivo. Se observa la capacidad del operador de modificar la escala de velocidad de la base mediante la pulsación simultánea del botón B y los botones + o -, fase representada por un sombreado verde y rojo respectivamente. La información de esta gráfica ha sido obtenida gracias a *Plotjuggler*, una herramienta de ROS que permite analizar el contenido de un mensaje dentro de un topic. Se ha simplificado la leyenda de las gráficas para facilitar su comprensión. Para un lector más familiarizado con los mensajes y topics de ROS se especifica que:

- Botón B corresponde con el contenido del topic `/joy/buttons[1]`
- Botón + corresponde con el contenido del topic `/joy/buttons[6]`
- Botón - corresponde con el contenido del topic `/joy/buttons[7]`
- Eje JRy corresponde con el contenido del topic `/joy/axes[1]`
- $\dot{x}_B$ corresponde con el contenido del topic `/cmd_base_vel/linear/x`

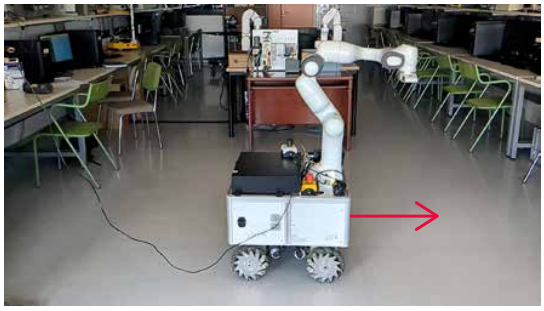
(a) Fotograma 1

(b) Fotograma 2

(c) Fotograma 3



(d) Fotograma 4



(e) Fotograma 5

(f) Fotograma 6

Figura 4.2: Secuencia de fotogramas que muestra la teleoperación del controlador de la base

Nótese que los valores de los ejes y botones no constan de unidades debido a que son comandos del mando. Se puede afirmar que la modificación de la escala de velocidad es satisfactoria.

4.3. Experimento 3. Teleoperación del controlador de impedancia del manipulador

El objetivo de este experimento es verificar el correcto funcionamiento del nodo de teleoperación del controlador de impedancia del manipulador. Para su puesta en marcha, se debe ejecutar:

```
1    roslaunch rafi_launch_files rafi_teleop_impedance_franka.launch
```

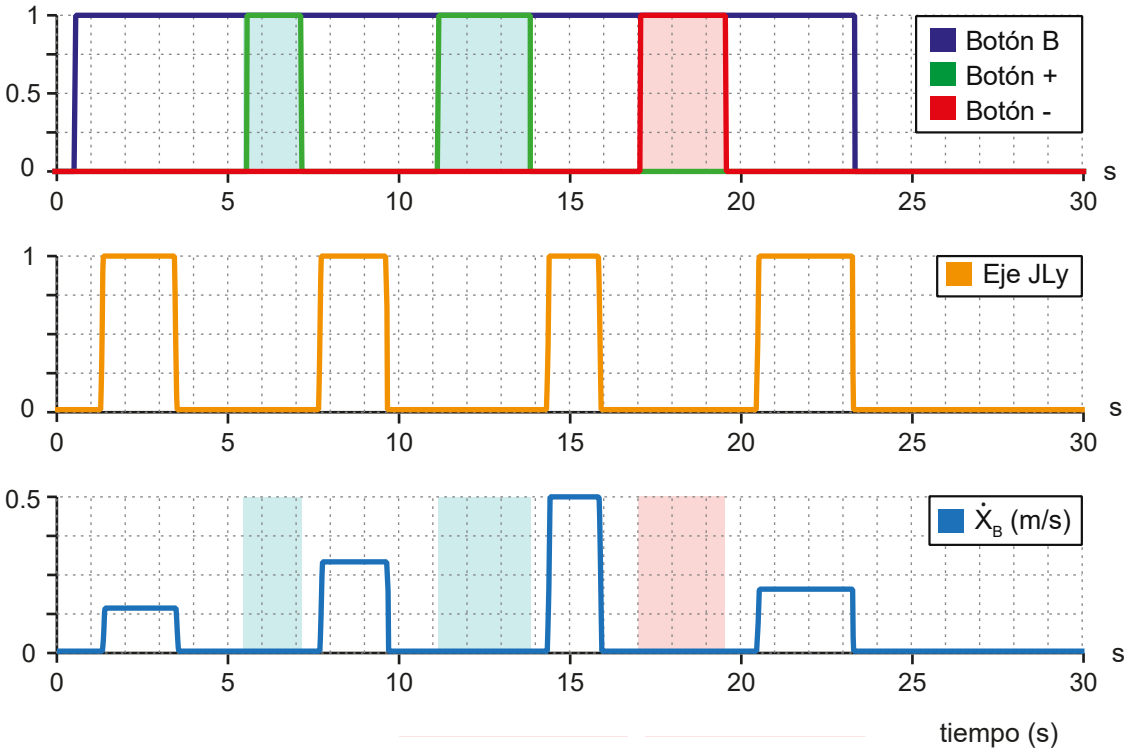


Figura 4.3: Relación de los comandos de los botones y ejes y la respuesta de velocidad de la base durante la teleoperación en el eje X positivo, mostrando la modificación de la escala de velocidad

La figura 4.4 muestra la teleoperación del controlador de impedancia del manipulador con el mando. Los fotogramas 1-2 muestran el movimiento en el eje X. El fotograma 3 corresponde al movimiento en el eje Y. Los fotogramas 4-5 corresponden con el movimiento en el eje Z. Por último, los fotogramas 6-9 muestran diferentes orientaciones del efector final. El manipulador se puede teleoperar mediante el mando correctamente, luego el experimento es satisfactorio.

4.4. Experimento 4. Teleoperación del controlador de velocidad del manipulador

El objetivo de este experimento es ejecutar la teleoperación del controlador de velocidad cartesiana del manipulador. El controlador requiere que el manipulador esté desbloqueado y en una determinada configuración articular. Para ello, se ejecuta el alias:

```
1    $ franka_ros_start_pos = 'roslaunch franka_example_controllers
        move_to_start.launch robot_ip:=172.16.0.2 load_gripper:=false
        robot:=panda'
```

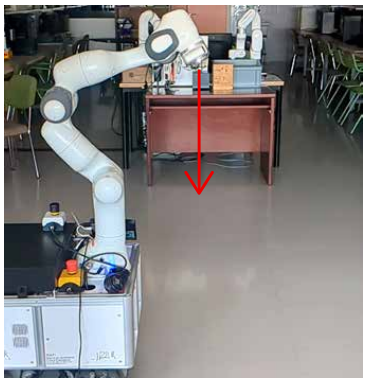
La figura 4.5 muestra como el manipulador cambia de pose tras ejecutar el comando.

Para la ejecución de la teleoperación se debe ejecutar el siguiente comando:

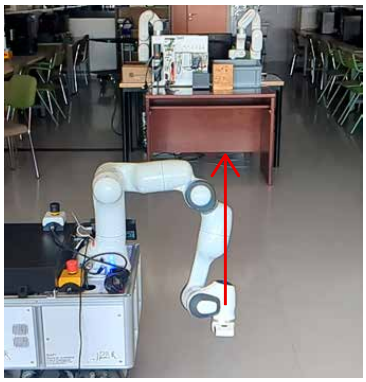
(a) Fotograma 1

(b) Fotograma 2

(c) Fotograma 3



(d) Fotograma 4



(e) Fotograma 5

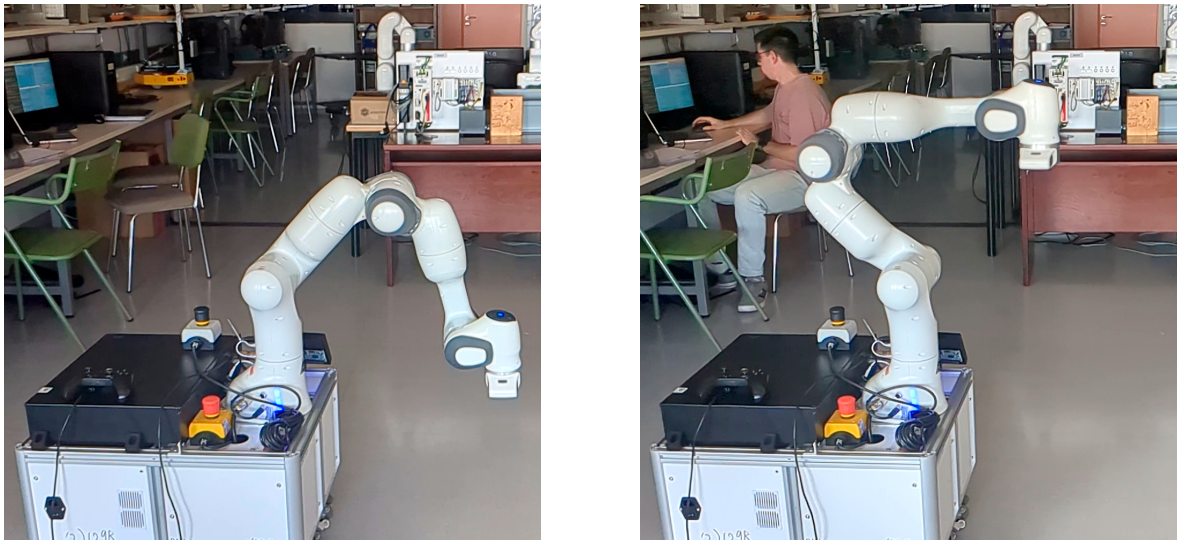
(f) Fotograma 6

(g) Fotograma 7

(h) Fotograma 8

(i) Fotograma 9

Figura 4.4: Secuencia de fotogramas que muestra la teleoperación del control de impedancia del manipulador



(a) Fotograma 1 (b) Fotograma 2

Figura 4.5: Secuencia de fotogramas que muestra la puesta en marcha de los requerimientos previos del controlador de velocidad cartesiana

```
roslaunch rafi_launch_files rafi_teleop_velocity_franka.launch
```

El resultado lo muestra la figura 4.6, dónde se muestra la teleoperación del controlador de velocidad cartesiano del manipulador con el mando. Los fotogramas 1-2 muestran el efector final moviéndose con velocidad lineal en el eje X. El fotograma 3 muestra el manipulador moviéndose con velocidad lineal en la dirección Y. Los fotogramas 4-5 corresponde a una velocidad lineal en el eje Z. Por último, los fotogramas 7-9 representan una variación de la velocidad angular del efector final respecto al fotograma 6.

El resultado del experimento es satisfactorio en términos de comandar velocidades desde el mando. Sin embargo, este controlador es inestable y tiende a sufrir errores de tipo reflejo relacionados con una falta de suavidad de la señal de entrada o por el alcance de los límites cartesianos. El largo tiempo empleado durante la aceleración y deceleración es provocado por el limitador de tasa, influyendo negativamente en la capacidad de teleoperación del efector final por parte del operador. El punto positivo de este controlador es la exactitud entre el comando del mando y el ejecutado por el efector final, logrando trayectorias más suaves que con el controlador de impedancia. Se puede observar que si se comanda una velocidad lineal en el eje X, el efector final no modifica su orientación, como si ocurre con el control de impedancia.

(a) Fotograma 1

(b) Fotograma 2

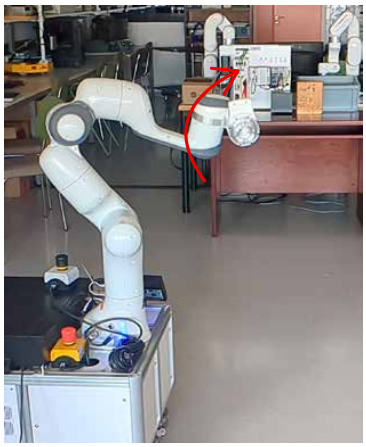
(c) Fotograma 3

(d) Fotograma 4

(e) Fotograma 5

(f) Fotograma 6

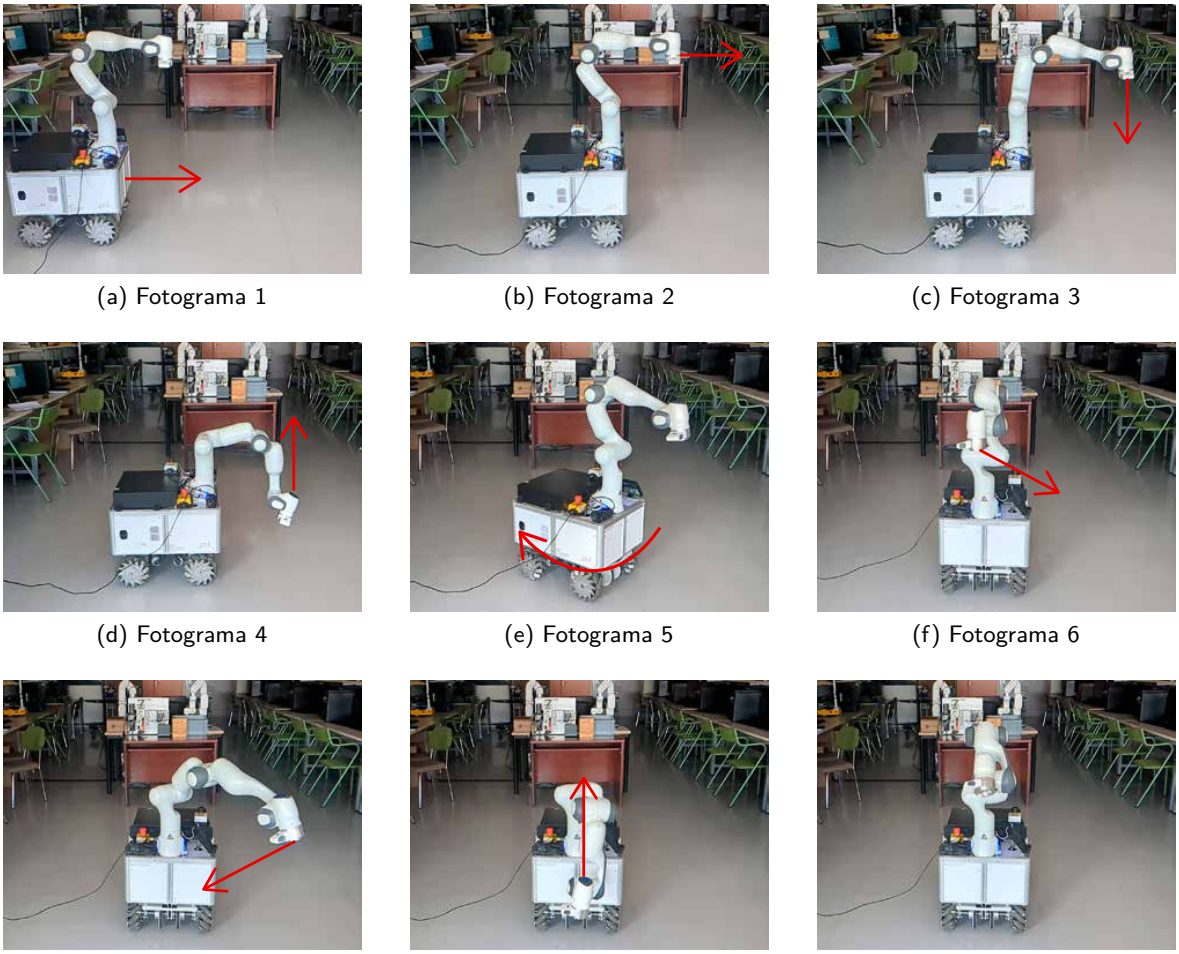
(g) Fotograma 7



(h) Fotograma 8

(i) Fotograma 9

Figura 4.6: Secuencia de fotogramas que muestra la teleoperación del control de velocidad cartesiana del manipulador Franka



(a) Fotograma 1 (b) Fotograma 2 (c) Fotograma 3

(d) Fotograma 4 (e) Fotograma 5 (f) Fotograma 6

(g) Fotograma 7 (h) Fotograma 8 (i) Fotograma 9

Figura 4.7: Secuencia de fotogramas que muestra la teleoperación del esquema A de RAFI

4.5. Experimento 5. Teleoperación del esquema A

El objetivo de este experimento es demostrar la teleoperación del esquema A. Este esquema consiste en un controlador de impedancia cartesiana para el manipulador junto con el controlador de la base. Para su puesta en marcha se debe ejecutar el siguiente comando:

```
1    roslaunch rafi_launch_files rafi_impedance.launch
```

La figura 4.7 muestra la teleoperación del esquema A de RAFI. El resultado de este experimento es satisfactorio porque permite mandar comandos de velocidad para la base y comandos de pose para el manipulador de manera simultánea desde el mando. Además, se destaca la sencillez para la ejecución de todos los nodos relacionados con los controladores, la teleoperación y el mando.

A continuación, se procede a realizar otro experimento cuyo objetivo es demostrar la relación entre los comandos del mando y la respuesta del sistema. Durante este experimento, tanto la base

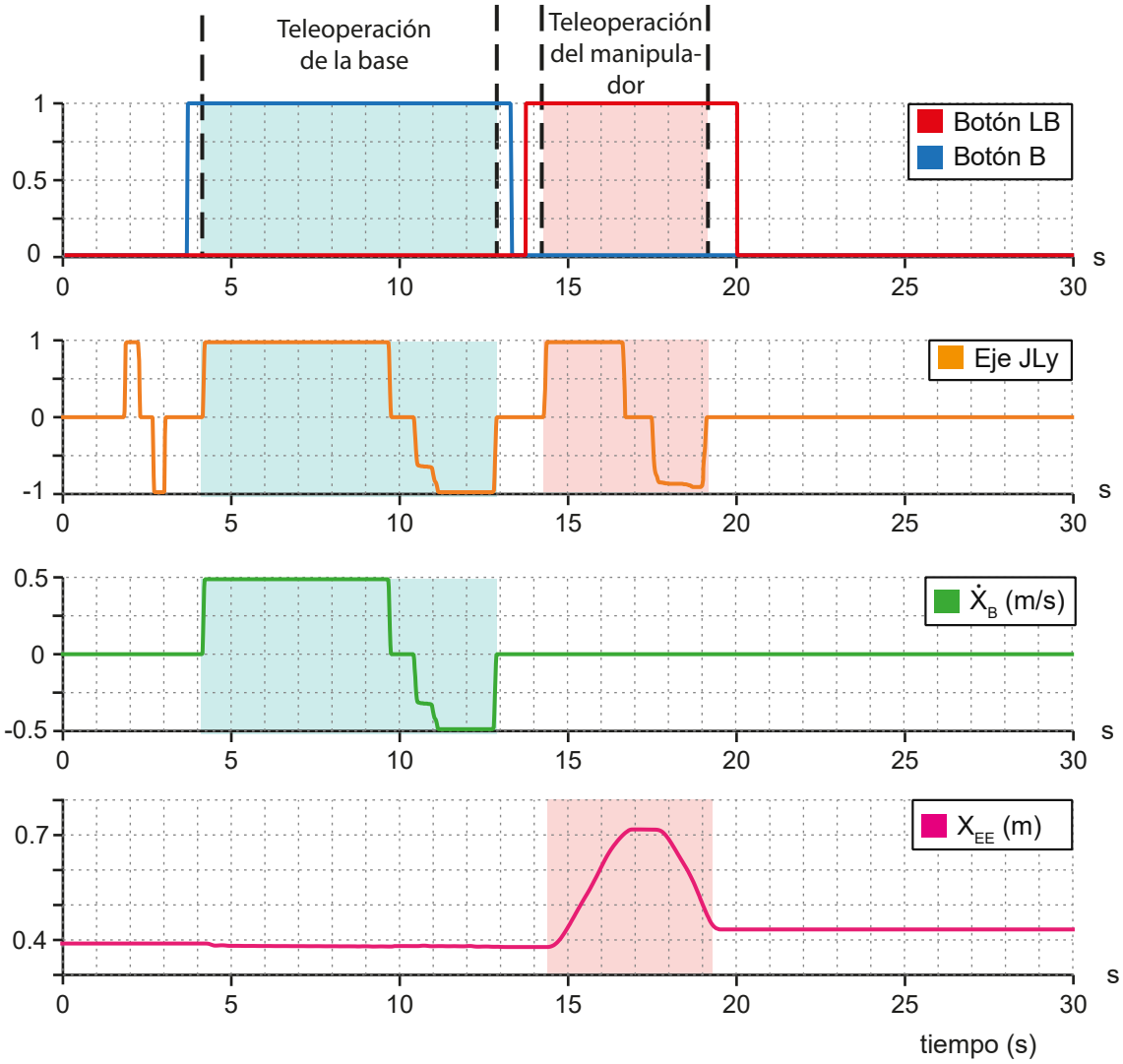


Figura 4.8: Relación entre los comandos del mando y la respuesta del sistema durante la teleoperación del Esquema A de RAFI. Durante este experimento, la plataforma móvil y el manipulador se han teleoperado a lo largo del eje X.

como el manipulador se van a mover a lo largo del eje X.

La figura 4.8 muestra la relación entre los comandos del mando, ejes y botones, y la velocidad de la base y la posición del efector final. Se observa que es necesario pulsar alguno de los botones de activación de movimiento para comandar movimiento mediante el joystick. El accionamiento simultáneo de B y JLy comanda velocidad lineal a la base, fase representada en un sombreado verde. Entre los segundos 4 y 9,8 la base se mueve en el sentido positivo de X. Entre los segundos 10,5 y 12 la base se mueve en el sentido negativo de X.

Respecto al manipulador, el accionamiento simultáneo de LB y JLy comanda una nueva pose, fase representada en un sombreado rojo. Entre los segundos 14,5 y 17 el efector final del manipulador se

desplaza en el sentido positivo del eje X. Entre los segundos 17,8 y 19,5 el efector final se desplaza en el sentido negativo del eje X.

La información de esta figura ha sido extraída gracias a Plotjuggler. La relación entre la leyenda y los mensajes contenidos en los topics es la siguiente:

- Botón B es el estado del botón. Corresponde con el contenido del topic `/joy/buttons[1]`
- Botón LB es el estado del botón. Corresponde con el contenido del topic `/joy/buttons[4]`
- Eje JRy es el estado del joystick JR en su dirección Y. Corresponde con el contenido del topic `/joy/axes[1]`
- $\dot{x}_B$ se trata de la velocidad lineal de la base en el eje X. Corresponde con el contenido del topic `/cmd_base_vel/linear/x`
- x_{EE} se trata de la posición cartesiana del efector final (End-Effector, EE). Corresponde con el contenido del topic `/franka_state_controller/franka_states/O_T_EE[12]`

Se procede a analizar el segundo 4,3 de la curva X_{EE}. Se observa como el inicio del movimiento de la base modifica ligeramente la posición en X del efector final. Esto es provocado por una rigidez baja. Aumentar el valor de la rigidez traslacional corregiría el problema. Finalmente, se afirma que la teleoperación del esquema A es satisfactoria.

4.6. Experimento 6. Teleoperación del esquema B

El objetivo de este experimento es comprobar el funcionamiento de la teleoperación del esquema B de RAFI. Este esquema consiste en un controlador de velocidad cartesiana para el manipulador y el controlador de la base. Para el correcto inicio del experimento, es necesario que el manipulador se encuentre en una determinada configuración articular. Para ello se utiliza el comando:

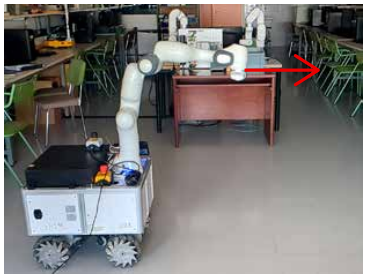
```
1    $ franka_ros_start_pos
```

A continuación, se debe ejecutar el siguiente comando:

```
1    roslaunch rafi_launch_files rafi_velocity.launch
```

La figura 4.9 muestra la ejecución de la teleoperación del esquema B de RAFI. El experimento es satisfactorio porque se logra comandar velocidades cartesianas tanto a la base como al manipulador de forma simultánea. Además, la pose del codo del manipulador no se mueve durante el movimiento de la base, a diferencia del comportamiento durante la teleoperación del Esquema A. Este esquema permite mantener la configuración articular exacta del manipulador aunque la base se encuentre en movimiento.

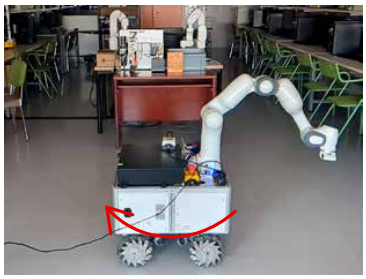
(a) Fotograma 1



(b) Fotograma 2

(c) Fotograma 3

(d) Fotograma 4



(e) Fotograma 5

(f) Fotograma 6

(g) Fotograma 7

(h) Fotograma 8

(i) Fotograma 9

Figura 4.9: Secuencia de fotogramas que muestra la teleoperación del esquema B de RAFI

A continuación, se procede a analizar el comportamiento de la teleoperación del Esquema B mediante otro experimento. Por simplicidad, solo se va a modificar la velocidad lineal en X de la base y del efector final.

La figura 4.10 muestra la relación entre los comandos del mando, ejes y botones implicados, y la respuesta en velocidad de la base y del manipulador. Para teleoperar la base, se debe accionar simultáneamente el botón B y el eje JLy, fase representada por un sombreado verde. Para teleoperar el efector final del manipulador se debe accionar el botón LB y el eje JLy, fase representada en un sombreado rojo. La curva $\dot{X}_{EE}$ muestra la velocidad lineal en el eje X del efector final. Entre los segundos 10 y 12,2 se observa la aceleración del efector final con velocidad positiva. Cuando se deja de accionar el botón LR, el efector final decelera. Entre los segundos 17,2 y 20,1 vuelve a acelerar con velocidad negativa, es decir, se mueve en el sentido negativo del eje X. Posteriormente, decelera entre los segundos 20,1 y 23,6. La fase de deceleración comienza cuando no se pulsa el botón LR o el eje JLy de manera simultánea. La deceleración es suave como consecuencia del limitador de ratio.

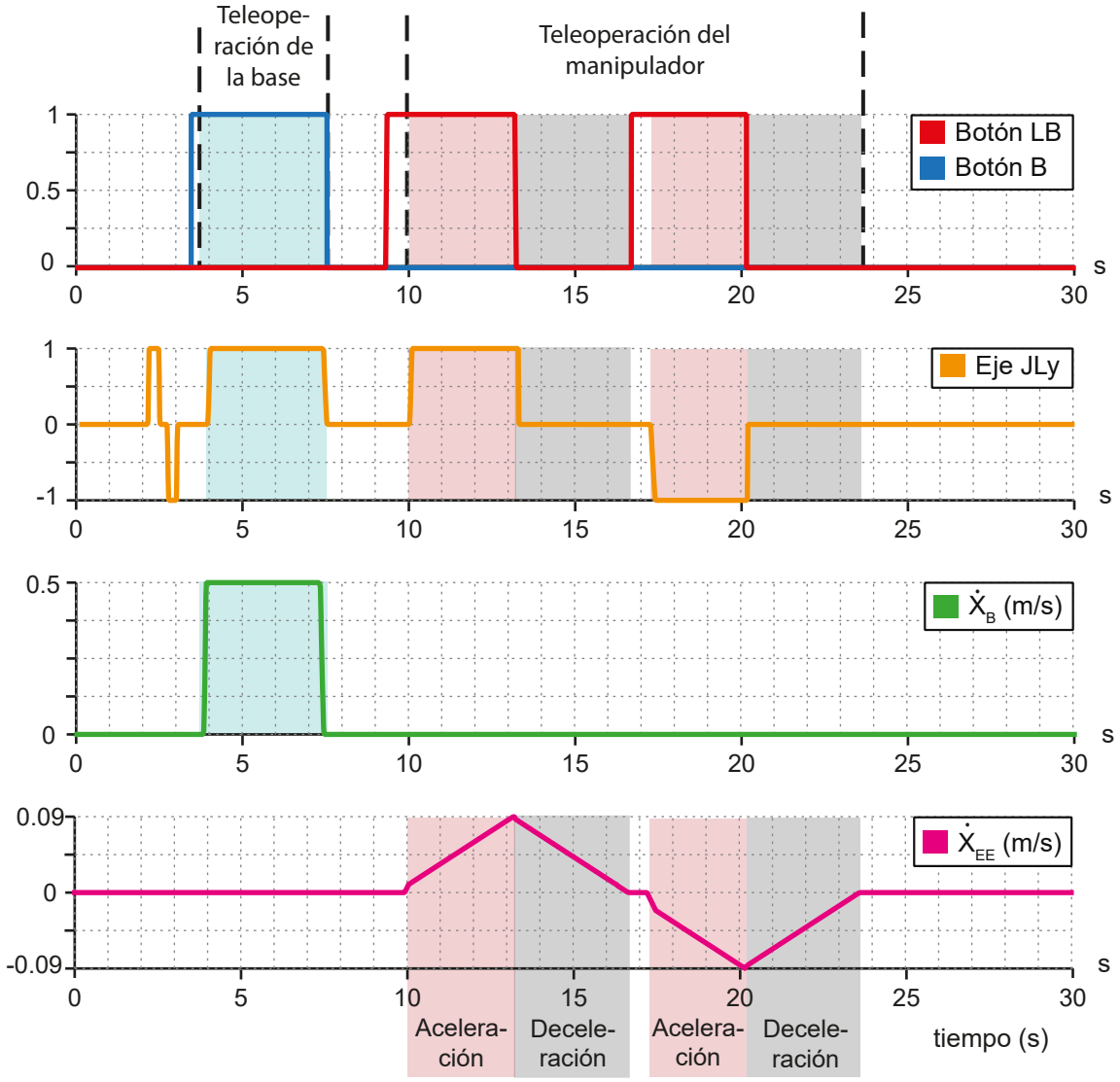


Figura 4.10: Relación entre los comandos del mando y la respuesta del sistema durante la teleoperación del Esquema B de RAFI. Durante este experimento, la plataforma móvil y el manipulador se han teleoperado a lo largo del eje X.

La información de esta figura ha sido extraída gracias a Plotjuggler. La relación entre la leyenda y los mensajes contenidos en los topics es la siguiente:

- Botón B es el estado del botón. Corresponde con el contenido del topic `/joy/buttons[1]`
- Botón LB es el estado del botón. Corresponde con el contenido del topic `/joy/buttons[4]`
- Eje JRy es el estado del joystick JR en su dirección Y. Corresponde con el contenido del topic `/joy/axes[1]`
- $\dot{x}_B$ se trata de la velocidad lineal de la base en el eje X. Corresponde con el contenido del topic `/cmd_base_vel/linear/x`
- $\dot{x}_{EE}$ se trata de la velocidad cartesiana del efector final. Corresponde con el contenido del topic `/franka_state_controller/franka_states/O_dP_EE_d[0]`

Se afirma que la teleoperación del Esquema B funciona correctamente aunque con limitaciones en los tiempos de aceleración y deceleración del efector final debido a la aplicación del limitador de ratio, requisito indispensable para evitar errores en el control del manipulador.

CAPÍTULO 5

Conclusiones

En este capítulo se analiza la calidad de las soluciones adoptadas para cumplir con los objetivos recogidos en el apartado 1.3, así como la descripción de los principales desafíos surgidos durante el desarrollo del proyecto. Además, se propondrán posibles líneas de trabajo futuras para la mejora de RAFI.

Entre las contribuciones de este trabajo se encuentra una mejora en la interconexión y comunicación entre los distintos agentes involucrados en el funcionamiento de RAFI: manipulador, base móvil y el operador humano. Se ha aportado una interfaz de software para el control y teleoperación del robot en su totalidad. Se ha integrado el control de nuevos elementos como el brazo manipulador. Se aportan dos controladores para el manipulador, en impedancia cartesiana y velocidad cartesiana. Se ha modificado la implementación anterior del controlador de la base móvil. Además, se ha documentado el funcionamiento detrás de cada uno de los controladores mediante la aportación del modelo matemático y su diagrama de entradas y salidas. Se ha aportado un paquete de ROS encargado de la teleoperación de cada uno de los controladores mediante un mando accionado por el operador humano. El código fuente de estos paquetes sigue una estructura homogénea que facilita su uso por personas no familiarizadas con el proyecto. Este proyecto ha continuado la senda establecida por los anteriores trabajos mediante la aportación de nuevas funcionalidades al robot RAFI.

A continuación se va analiza el cumplimiento de los objetivos iniciales del proyecto, cómo se han llevado a cabo y las principales dificultades surgidas en el proceso.

El objetivo 1 era establecer una comunicación efectiva entre todos los componentes del sistema a través de un sistema distribuido. Para ello se ha utilizado ROS, dónde cada nodo se encarga de ejecutar una función específica. El uso de ROS ha permitido la creación de un sistema escalable y distribuido. La escalabilidad se ha demostrado durante el desarrollo dónde se han ido incorporando funcionalidades de forma progresiva, al principio se trabajó en la funcionalidad de la base y

posteriormente se trabajó en el control del manipulador. La capacidad de distribución del sistema, es decir, que los distintos nodos estén interconectados con independencia del lugar de ejecución ha permitido una mayor flexibilidad durante el desarrollo. Nodos como ”rqt_graph” han permitido analizar el comportamiento de los nodos y topics del sistema desde el PC de desarrollo. Además, se ha utilizado el protocolo SSH para comunicar ambos PCs, lo que ha permitido trabajar con el PC integrado sin acceso físico a él.

El objetivo 2 era adaptar el software desarrollado en anteriores trabajos para la plataforma RAFI para controlar y teleoperar la base. El software desarrollado con anterioridad consiste en un paquete de ROS encargado del control y teleoperación de los motores de la base mediante un mando y el uso de un sensor LiDAR para evitar obstáculos. El sensor LiDAR no se ha utilizado en el presente proyecto. El primer paso fue analizar y comprender el paquete. En el proceso se identificaron posibles mejoras en la implementación relacionadas con la modificación de la escala de velocidad. La versión previa utilizada dos escalas de velocidad, una a velocidad normal y otra a mayor velocidad, lo que limitaba la teleoperación de la base. Se sustituyó por un método encargado de incrementar y decrementar la escala de manera progresiva mediante el uso de dos botones. Esta modificación ha mejorado la capacidad de teleoperación de la base y ha resultado en un código más limpio y comprensible para personas que no están familiarizadas con él.

El objetivo 3 era implementar herramientas para la gestión del manipulador mediante terminal. Este objetivo era fundamental para comenzar a trabajar con el manipulador porque su gestión se realiza desde el navegador web del PC integrado. El uso del protocolo SSH para abrir una ventana gráfica del navegador se descartó debido a su rendimiento deficiente. Como alternativa, se ha utilizado un paquete ROS ya desarrollado que actúa como cliente web del IDE de Franka. Esto ha permitido el desbloqueo del manipulador y activación del modo FCI desde la terminal de manera satisfactoria.

El objetivo 4 era adaptar el controlador de impedancia cartesiana en el manipulador. Se ha implementado el controlador de ejemplo desarrollado por el fabricante y se ha modificado. Este controlador utiliza una visualización en tiempo real del manipulador que permite la modificación de la pose del efector final desde una ventana gráfica de RViz. Se sustituyó la subscripción al simulador por una subscripción a un topic que actúa como entrada del comando de pose deseada. De esta forma, el nodo encargado de su teleoperación, o el propio operador desde la terminal, podría publicar una nueva pose para el manipulador. El principal problema durante este proceso fue que no se había eliminado la subscripción del controlador al simulador, de forma que el manipulador obtenía un comando de pose tanto del nodo de teleoperación como del nodo de RVIz simultáneamente. Esto provocaba un comportamiento errático del manipulador, que se reflejaba en que un temblor del efector final durante la teleoperación. Finalmente, la eliminación de la subscripción a RViz eliminó el temblor aunque obligó a modificar el nodo de teleoperación para obtener la pose inicial y evitar que se desplomase durante el inicio del nodo de teleoperación.

El objetivo 5 era adaptar el controlador de velocidad cartesiana en el manipulador. Al igual que el

control de impedancia, el controlador de ejemplo es proporcionado por el fabricante. El controlador original está diseñado para ejecutar una secuencia de movimiento en bucle, sin interacción alguna con el operador. Se modificó este controlador para eliminar la secuencia y añadir una subscripción a un topic como entrada de comandos de velocidad deseada. Entre los requisitos para la correcta ejecución de este controlador se encuentran que el manipulador debe estar en una determinada configuración articular al inicio del control y que la señal a comandar debe ser suficientemente suave y continua. Una violación del segundo requerimiento provoca un error en el controlador que resulta en su detención. La documentación se refiere a estos errores como discontinuidades o reflejos (reflex). Se probaron distintas soluciones como un filtro de paso bajo, un filtro de primer orden o un limitador de tasa. Finalmente, se optó por usar el limitador de tasa incluido en la propia librería del manipulador como ”limitRate”. Este limitador de tasa tiene como entrada los valores máximos de velocidad, aceleración y tirón, los valores pasados y la velocidad deseada. Se ha optado por disminuir en un orden de magnitud los valores máximos del manipulador para disminuir la aparición de reflejos. Estas medidas han disminuido la aparición de reflejos aunque todavía es frecuente la aparición de alguno. El mayor inconveniente del uso del limitador de tasa es el tiempo que tarda en acelerar y decelerar el efector final, lo que limita su uso en aplicaciones reales.

El objetivo 6 era el desarrollo de la interfaz de software encargada del control de ambos robots y el objetivo 7 era el desarrollo de una interfaz de teleoperación simultánea para ambos robots. Esto es el resultado de la correcta implementación del sistema distribuido y de la implementación y ejecución de los controladores y sus nodos de teleoperación de la base y del manipulador de manera simultanea. El resultado ha sido satisfactorio, siendo comprobable en las secciones encargadas de la teleoperación de los sistemas de control de RAFI del capítulo 4; donde se aportan imágenes tomadas durante la teleoperación de los robots de manera simultánea.

RAFI tiene un gran potencial de mejora en distintos aspectos. Se proponen nuevas lineas de trabajo relacionadas con el aumento de capacidades, la mejora de la interfaz de software actual o la mejora de su relación con el entorno.

Respecto al control del robot, se propone la creación de un controlador de cuerpo completo para el robot, considerando la cadena cinemática desde las ruedas de la base hasta el efector final del manipulador. Esto sería la base para la creación de un controlador de impedancia para RAFI que permita mantener el efector final en una posición en el espacio aunque se mueva la base. El robot intentaría compensar los esfuerzos que soporta la base mediante el movimiento del efector final. Una de las limitaciones que tiene la implantación actual es que el efector final puede colisionar con la plataforma. Este problema se podría abordar con un controlador de cuerpo completo.

En cuanto a la interfaz de teleoperación, se podrían abordar nuevas técnicas de retroalimentación como el uso de respuesta háptica cuando el robot alcance un límite cartesiano o cuando se aumente la escala de velocidad. También se podría añadir una funcionalidad real a la pantalla táctil para la gestión del robot como activación de frenos del manipulador, inicio de un controlador determinado o modificación de parámetros de impedancia en tiempo real. También se podría utilizar para aportar

retroalimentación visual del estado real del robot.

En relación a nuevas aplicaciones del robot, es fundamental que el brazo manipulador conste de una herramienta en el efector final para que pueda interactuar con el entorno. Se propone el acoplamiento de una pinza o una ventosa y su integración en la interfaz de control y teleoperación. El uso de diferentes herramientas aumenta el abanico de aplicaciones del robot.

Bibliografía

[1] E. Garcia, M. A. Jimenez, P. G. De Santos y M. Armada, "The evolution of robotics research," *IEEE Robotics and Automation Magazine*, vol. 14, n.º 1, págs. 90-103, 2007. DOI: 10.1109/MRA.2007.339608.

[2] O. Khatib, "Mobile manipulation: The robotic assistant," *Robotics and Autonomous Systems*, vol. 26, n.º 2, págs. 175-183, 1999, Field and Service Robotics, ISSN: 0921-8890. DOI: https://doi.org/10.1016/S0921-8890(98)00067-0. dirección: https://www.sciencedirect.com/science/article/pii/S0921889098000670.

[3] T. B. Sheridan, "Human–Robot Interaction: Status and Challenges," *Human Factors*, vol. 58, n.º 4, págs. 525-532, 2016, PMID: 27098262. DOI: 10.1177/0018720816644364. eprint: https://doi.org/10.1177/0018720816644364. dirección: https://doi.org/10.1177/0018720816644364.

[4] E. Matheson, R. Minto, E. G. Zampieri, M. Faccio y G. Rosati, "Human–robot collaboration in manufacturing applications: A review," *Robotics*, vol. 8, n.º 4, pág. 100, 2019.

[5] F. Sherwani, M. M. Asad y B. Ibrahim, "Collaborative Robots and Industrial Revolution 4.0 (IR 4.0)," en *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, 2020, págs. 1-5. DOI: 10.1109/ICETST49965.2020.9080724.

[6] V. Villani, F. Pini, F. Leali y C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, págs. 248-266, 2018, ISSN: 0957-4158. DOI: https://doi.org/10.1016/j.mechatronics.2018.02.009. dirección: https://www.sciencedirect.com/science/article/pii/S0957415818300321.

[7] R. Sultanov, S. Sulaiman, H. Li, R. Meshcheryakov y E. Magid, "A Review on Collaborative Robots in Industrial and Service Sectors," en *2022 International Siberian Conference on Control and Communications (SIBCON)*, 2022, págs. 1-7. DOI: 10.1109/SIBCON56144.2022.10003014.

[8] Y. R. Stürz, L. M. Affolter y R. S. Smith, "Parameter Identification of the KUKA LBR iiwa Robot Including Constraints on Physical Feasibility," *IFAC-PapersOnLine*, vol. 50, n.º 1, págs. 6863-6868, 2017, 20th IFAC World Congress, ISSN: 2405-8963. DOI: https://doi.

org/10.1016/j.ifacol.2017.08.1208. dirección: https://www.sciencedirect.com/science/article/pii/S2405896317317147.

[9] D. Speck, C. Dornhege y W. Burgard, "Shakey 2016—How Much Does it Take to Redo Shakey the Robot?" *IEEE Robotics and Automation Letters*, vol. 2, n.º 2, págs. 1203-1209, 2017. DOI: 10.1109/LRA.2017.2665694.

[10] K. A. Wyrobek, E. H. Berger, H. M. Van der Loos y J. K. Salisbury, "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot," en *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, págs. 2165-2170.

[11] J. Salisbury y M. Srinivasan, "Phantom-based haptic interaction with virtual objects," *IEEE Computer Graphics and Applications*, vol. 17, n.º 5, págs. 6-10, 1997. DOI: 10.1109/MCG.1997.1626171.

[12] D. Van Krevelen y R. Poelman, "A survey of augmented reality technologies, applications and limitations," *International journal of virtual reality*, vol. 9, n.º 2, págs. 1-20, 2010.

[13] S. Cousins, "ROS on the PR2 [ROS Topics]," *IEEE Robotics and Automation Magazine*, vol. 17, n.º 3, págs. 23-25, 2010. DOI: 10.1109/MRA.2010.938502.

[14] J. Bohren, R. B. Rusu, E. Gil Jones et al., "Towards autonomous robotic butlers: Lessons learned with the PR2," en *2011 IEEE International Conference on Robotics and Automation*, 2011, págs. 5568-5575. DOI: 10.1109/ICRA.2011.5980058.

[15] N. Kashiri, L. Baccelliere, L. Muratore et al., "CENTAURO: A Hybrid Locomotion and High Power Resilient Manipulation Platform," *IEEE Robotics and Automation Letters*, vol. 4, n.º 2, págs. 1595-1602, 2019. DOI: 10.1109/LRA.2019.2896758.

[16] J. Y. C. Chen, E. C. Haas y M. J. Barnes, "Human Performance Issues and User Interface Design for Teleoperated Robots," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, n.º 6, págs. 1231-1245, 2007. DOI: 10.1109/TSMCC.2007.905819.

[17] W. Song, X. Guo, F. Jiang, S. Yang, G. Jiang e Y. Shi, "Teleoperation Humanoid Robot Control System Based on Kinect Sensor," en *2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 2, 2012, págs. 264-267. DOI: 10.1109/IHMSC.2012.159.

[18] J. Li, H. Liu, X. Luo, C. L. Philip Chen y C. Yang, "Gesture-Based Human-Robot Interaction Framework for Teleoperation Control of Agricultural Robot," en *2023 IEEE International Conference on Unmanned Systems (ICUS)*, 2023, págs. 1-6. DOI: 10.1109/ICUS58632.2023.10318494.

[19] J. Lee, H. Yoon y D. Lee, "A Stable Tele-operation of a Mobile Robot with the Haptic Feedback," *IFAC-PapersOnLine*, vol. 51, n.º 22, págs. 13-18, 2018, 12th IFAC Symposium on Robot Control SYROCO 2018, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.11.511. dirección: https://www.sciencedirect.com/science/article/pii/S2405896318332178.

[20] P. Stotko, S. Krumpen, M. Schwarz et al., "A VR System for Immersive Teleoperation and Live Exploration with a Mobile Robot," en *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, págs. 3630-3637. DOI: 10.1109/IROS40897.2019.8968598.

[21] F. de Paula Carbonero Navarro, "Funciones básicas de navegación ROS para un manipulador móvil omnidireccional," Trabajo Fin de Grado, Universidad de Málaga, Escuela de Ingenierías Industriales, Málaga, España, 2023.