Universidad de Málaga

Escuela de ingenierías industriales

Departamento de Ingeniería de Sistemas y Automática

Trabajo Fin de Grado

Desarrollo de un modelo de simulación para un manipulador móvil con base robótica omnidireccional

Grado en ingeniería electrónica, robótica y mecatrónica

Autor: Mario Alber Gil

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

21 de febrero de 2025

Declaración de Originalidad del Trabajo Fin de Grado

D./Dña. Mario Alber Gil
DNI/Pasaporte: 79043992F. Correo electrónico: 0610671528@uma.com
Titulación: Grado en ingeniería electrónica, robótica y mecatrónica
Título del Proyecto/Trabajo: Desarrollo de un modelo de simulación para un manipulador móvil con base robótica omnidireccional

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 21 de febrero de 2025

Fdo.: Mario Alber Gil

Resumen

Desarrollo de un modelo de simulación para un manipulador móvil con base robótica omnidireccional

Autor: Mario Alber Gil

Tutor: Juan Manuel Gandarias Palacios

Cotutor: Jesús Manuel Gómez de Gabriel

Departamento: Departamento de Ingeniería de Sistemas y Automática

Titulación: Grado en ingeniería electrónica, robótica y mecatrónica

Palabras clave Robótica; Manipulador móvil; Simulación; Gazebo

El presente proyecto se centra en el desarrollo e integración de un modelo de simulación para la plataforma robótica RAFI, previamente diseñada por el Departamento de Ingeniería de Sistemas y Automática. Dicha plataforma combina una base móvil omnidireccional con un brazo manipulador robótico de siete grados de libertad.

Para su implementación, se emplean Gazebo, un simulador de robótica de código abierto, y ROS (Robot Operating System), un marco de trabajo diseñado para el desarrollo de sistemas robóticos. El enfoque principal del proyecto es la creación de un modelo de simulación de la base móvil del robot, seguido de su integración con el modelo preexistente del brazo robótico.

Las simulaciones realizadas han validado el funcionamiento del modelo completo desarrollado, logrando simular con éxito el movimiento omnidireccional de la plataforma, así como su interacción con un entorno simulado.

Abstract

Development of a simulation model for a mobile manipulator with an omnidirectional robotic base

Author: Mario Alber Gil

Supervisor: Juan Manuel Gandarias Palacios

Cosupervisor: Jesús Manuel Gómez de Gabriel

Departamemt: Departamento de Ingeniería de Sistemas y Automática

Degree: Grado en ingeniería electrónica, robótica y mecatrónica

Keywords: Robótica; Manipulador móvil; Simulación; Gazebo

This project entails the development and integration of a simulation model for the RAFI robotic platform, which was previously designed by the Department of Systems and Automatics Engineering. The platform integrates an omnidirectional mobile base with a robotic manipulator arm featuring seven degrees of freedom.

To achieve this, the project employs Gazebo, an open-source robotics simulator, and ROS (Robot Operating System), a framework designed for robotic system development. The primary objective is the creation of a simulation model for the mobile base of the robot, followed by its integration with the existing robotic arm model.

The simulations conducted validate the functionality of the fully integrated robotic system. Specifically, they demonstrate the omnidirectional movement capabilities of the platform and its interaction with the simulated environment.

A la memoria de
mis abuelos Antonio y María

Agradecimientos

En primer lugar, agradecer a mis tutores, Juanma y Jesús, por permitirme realizar este trabajo de fin de grado, que me ha ayudado a expandir mis conocimientos y a desarrollarme académicamente. Gracias también por la ayuda que me han brindado, sin la que no habría podido llevar a cabo el proyecto. Gracias a mi madre y hermana, por su apoyo emocional cuando más lo necesitaba, y a mi padre, por siempre intentar ayudarme en el proyecto. Gracias también por inculcarme los valores del trabajo y esfuerzo desde pequeño, sin los que no habría sido posible llegar tan lejos. Finalmente, gracias a todos los amigos que me han acompañado durante toda mi etapa universitaria.

Índice

Índice de Figuras

Figura **Página**

CAPÍTULO 1

Introducción

Contenido

En este capítulo se presenta la motivación que ha llevado a la realización de este proyecto, así como el estado del arte relativo a los modelos de simulación de distintas plataformas y manipuladores móviles. Además, se detallan los objetivos que se desean conseguir a lo largo del proyecto y, finalmente, se detalla la estructura de esta memoria.

1.1. Motivación

En los últimos años, se ha producido una gran evolución en los robots inteligentes capaces de operar en distintos entornos [2]. En este sentido, los robots manipuladores móviles han cobrado una gran importancia en distintas aplicaciones, desde industriales hasta domésticas. Un robot manipulador móvil se compone, generalmente, de una plataforma móvil integrada con uno o varios brazos robóticos, además de otros subsistemas dedicados a sensores y diversas herramientas [3]. Este tipo de robots ha ganado popularidad debido a su versatilidad para operar en una amplia gama de entornos, que abarcan desde aplicaciones industriales [4] o domésticas [5] hasta entornos agrícolas [6].

Un desafío común en el desarrollo de robots es que no siempre es la imposibilidad de realizar pruebas directamente en ciertos entornos, por lo que la simulación se ha vuelto una herramienta imprescindible a la hora de desarrollar e investigar estas áreas de la robótica [7]. Los modelos de simulación desempeñan un papel esencial en el desarrollo de robots al ofrecer una herramienta que agiliza el proceso de diseño, reduce los costos asociados y permite verificar el funcionamiento del sistema en entornos virtuales antes de su implementación física [2]. Además, estos modelos permiten probar y optimizar algoritmos complejos, evaluar el comportamiento del sistema bajo diferentes escenarios y facilitar la colaboración entre equipos de desarrollo multidisciplinarios [8].

En este contexto, el número de simuladores ha estado creciendo rápidamente. Entre los más usados se encuentran Webots [9], MORSE [10], CoppeliaSim [11] y Gazebo [12], siendo los dos últimos los que dan un mejor rendimiento [13]. La elección del simulador adecuado depende de las necesidades específicas del proyecto, como la complejidad del entorno simulado, la interacción con sensores, la precisión de las dinámicas físicas y la integración con otros componentes de software.

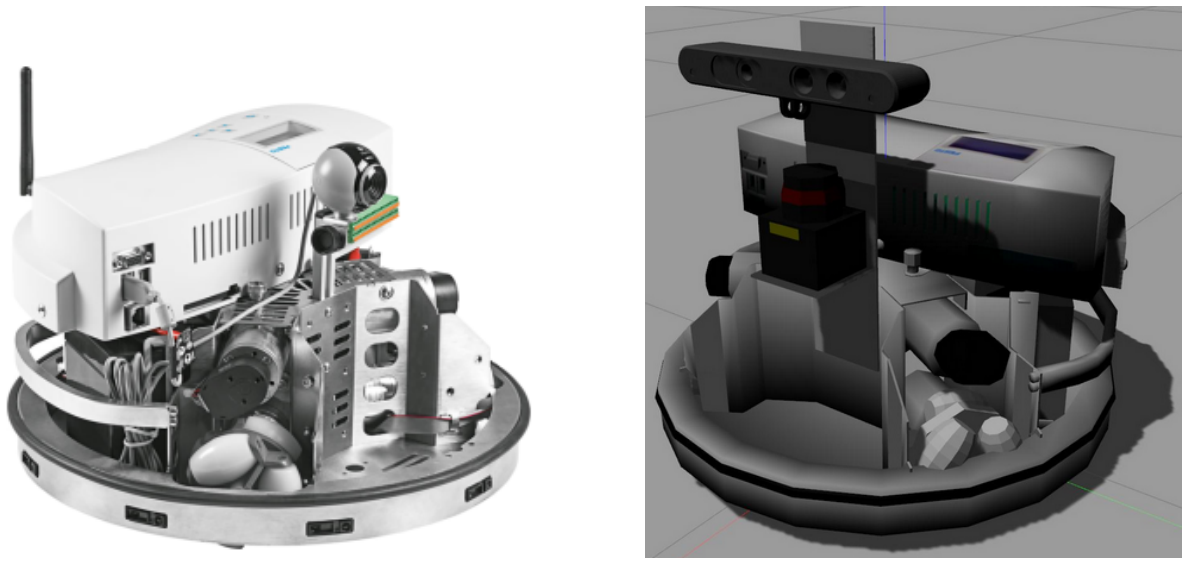
Cada uno de los simuladores antes mencionados usa un formato de descripción distinto. Estos formatos de descripción representan distintas maneras de describir un robot, ya sea visualmente o expresando su cinemática [14]. Estos formatos de descripción permiten una representación detallada de los robots y sus componentes, facilitando la integración y el control en entornos virtuales cada vez más sofisticados.

En el contexto de este trabajo específico, se aborda la creación de un modelo de simulación completo y exhaustivo para un robot manipulador móvil, compuesto por un brazo robótico comercial y una base omnidireccional desarrollada y fabricada en el departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga. Esto implica la fusión e integración de ambos modelos para obtener una representación completa y funcional del robot en un entorno de simulación. Así, con el modelo completo, se podrá en un futuro analizar y evaluar el comportamiento del robot en una variedad de situaciones y escenarios simulados, lo que permitirá no solo comprender su funcionalidad y eficacia, sino también identificar posibles áreas de mejora y optimización en su diseño y programación.

1.2. Estado del arte

Como se ha comentado anteriormente, la simulación robótica está en auge, con cada vez más modelos de simulación de robots, por lo que no es difícil encontrar diferentes modelos de simulación. En el contexto de este trabajo son de especial interés robots móviles omnidireccionales y manipuladores. A continuación se describen algunos de los modelos más relevantes.

Robotino es un robot diseñado por Festo Didactic, que incluye sensores, actuadores y software de calidad para ser un robot orientado al ámbito educativo. Por esto mismo es un robot popular [15].



(a) Fuente:[1]. (b) Fuente:[2].

Figura 1.1: Imágenes del robot Robotino de Festo Didactic real (a) y su modelo de simulación en Gazebo (b).

Este robot posee un modelo de simulación completo y funcional, que es accesible en su github [3]. Este modelo está especialmente pensado para su simulación en gazebo, aunque es posible su uso en otros entornos de simulación, siempre que se tenga instalado el paquete *rto_core*. En este paquete se tiene todo lo necesario tanto para la simulación como para su uso real, como puede ser la descripción del robot o programas para su localización y navegación.

La estructura del metapaquete es sencilla, con un paquete dedicado a los archivos de lanzamiento del robot y otro paquete para los distintos mundos programados para pruebas.

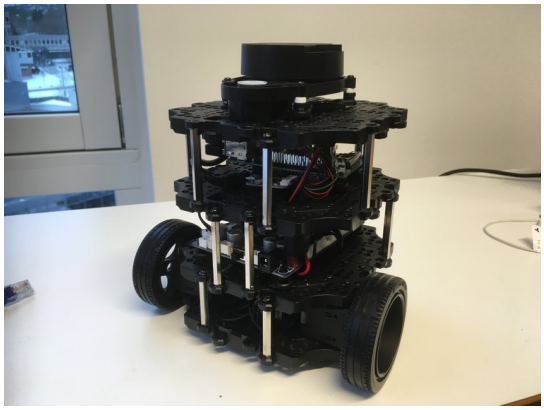
TurtleBot es un kit robótico de bajo coste desarrollado en 2010. El modelo más reciente es el TurtleBot4 que supone un avance respecto a los anteriores reduciendo su tamaño sin comprometer

[1] Wiki ROS, *Robotino*, URL: http://wiki.ros.org/Robots/Robotino. Fecha de acceso: 11 de septiembre 2024.

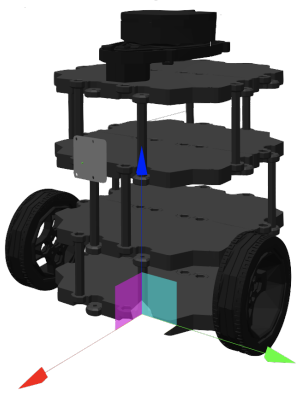
[2] GitHub, *Robotino*, URL: https://github.com/dietriro/rto_simulation?tab=readme-ov-file. Fecha de acceso: 11 de septiembre 2024.

[3] El metapaquete de simulación del robotino se encuentra en el siguiente enlace: https://github.com/dietriro/rto_simulation?tab=readme-ov-file.

su capacidad. Este robot móvil está pensado para aquellas personas que desean aprender robótica a un precio reducido pero con una potencia adecuada para ello. Es *open-source*, por lo que los modelos de simulación se pueden encontrar en su github [4].



(a) Fuente:[5].

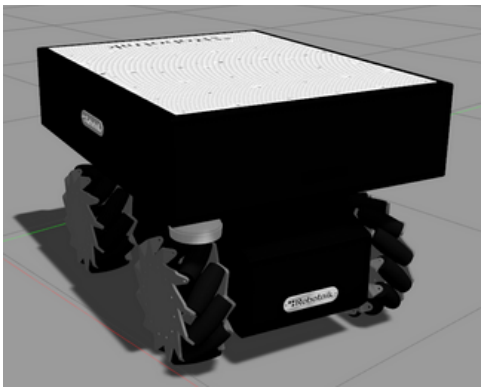


(b) Fuente:[6].

Figura 1.2: Imágenes del robot Turtlebot3, en una de sus configuraciones, real (a) y su modelo de simulación en ROSDS (b).

Otro robot reseñable es el Summit-XL-STEEL desarrollado por Robotnik. Este modelo de summit es más reseñable en relación a nuestro proyecto debido a sus ruedas mecanum y su omnidireccionalidad, cualidades que comparte con el robot objeto de este trabajo. El summit está especialmente diseñado para soportar grandes pesos, de hasta 250 kilogramos [16].

(a) Fuente:[7].



(b) Fuente:[8].

Figura 1.3: Imágenes del robot Summit-XL-Steel real (a) y su modelo de simulación en Gazebo (b).

[4] El modelo se simulación del TurtleBot3 se encuentra en este enlace: https://github.com/ROBOTIS-GIT/turtlebot3_simulations.

[5] Researchgate, *Turtlebot 3*, URL: https://www.researchgate.net/publication/330422953_Classifying_Human_and_Robot_Movement_at_Home_and_Implementing_Robot_Movement_Using_the_Slow_In_Slow_Out_Animation_Principle. Fecha de acceso: 24 de noviembre 2024.

[6] The construct, *Turtlebot 3*, URL: https://www.theconstruct.ai/turtlebot3/. Fecha de acceso: 7 de noviembre 2024.

El modelo de simulación es también completamente funcional, aunque comparte paquetes con su versión estándar, el Summit-XL. El modelo de simulación está repartido entre dos repositorios, uno principal, llamado summit_xl_sim, dedicado al lanzamiento de la simulación y al control principalmente, y otro que alberga las descripciones y otros elementos comunes de los robots de la familia summit_xl, llamado summit_xl_common [9].

El modelo más relevante para este proyecto es el RB-Kairos, también desarrollado por Robotnik. Se trata de un robot manipulador colaborativo móvil pensado para aplicaciones industriales. Posee como base al anteriormente mencionado Summit-XL-Steel, aunque este posee además un brazo robot de la empresa *Universal Robots*. Al igual que el Summit, es capaz de soportar cargas de 250 kilogramos y posee 4 ruedas mecanum que le brindan la capacidad de movimiento omnidireccional [17].

Figura 1.4: Foto del robot RB-Kairos de Robotnik. Fuente:[10].

[7] ROS components, Summit-XL-Steel, URL: https://robots.ros.org/summit-xl-steel/. Fecha de acceso: 16 de noviembre 2024.

[8] Github, *Summit-XL-Steel*, URL: https://github.com/RobotnikAutomation/summit_xl_common. Fecha de acceso: 11 de septiembre 2024.

[9] Los paquetes de simulación del Summit-XL-Steel se encuentra en el siguiente enlace: https://github.com/RobotnikAutomation/summit_xl_sim, mientras que el paquete common se encuentra en el siguiente enlace: https://github.com/RobotnikAutomation/summit_xl_common.

En la simulación se realiza la misma separación que hemos comentado en el Summit. En el repositorio común (rbkairos_common[11]) se encuentran las descripciones y la configuración, mientras que en otro repositorio (rbkairos_sim[12]) se encuentran los archivos de lanzamiento de la simulación.

Todos estos modelos de simulación están descritos en formato URDF (siglas en inglés de *Unified Robot Description Format*, traducido como formato unificado de descripción de robots), estándar de ROS, cuya estructura se detalla en el capitulo 3. URDF permite describir la cinemática, dinámica, colisiones y la visualización de los robots, convirtiéndolo en un formato versátil. Además, es compatible con distintos simuladores. Aún así, no permite la descripción de distintos robots en un mismo archivo. A pesar de esto es uno de los formatos más usados en la robótica [14].

Además de URDF, existen una gran cantidad de formatos de descripción de robots. SDF (siglas en inglés de *Simulation Description Format*, traducido como formato de descripción de simulación) es el formato desarrollado junto con el simulador Gazebo, por lo que está principalmente diseñado para la descripción de objetos y entornos. Este es un formato ampliamente usado, pero solo permite la simulación con Gazebo, ya que posee ciertos parámetros exclusivos de este simulador [14].

SRDF (siglas en inglés de *Semantic Robot Description Format*, traducido como formato de descripción semántica de robots) es otro formato de descripción, enfocado en representar información que no existe en URDF pero útil en ciertas aplicaciones. Empezó como parte del proyecto de MoveIt!, un software para resolver trayectorias de robots. Además, existe una herramienta, llamada *MoveIt Setup Assistant*, que ofrece una interfaz de diálogo con la que poder generar este archivo de descripción [14].

Estos formatos están basados en XML (siglas en inglés de *Extensible Markup Language*, traducido como lenguaje de marcado extensible), lo que permite su creación y edición manual, aunque existen otros formatos basados en otro tipo de programación. Por ejemplo, SMURF (siglas en inglés de *Semantically Marked Up Record Format*, traducido como formato de registro marcado semánticamente) es un formato de descripción basado en YAML, diseñado para ser prácticamente una extensión de URDF [14].

1.3. Objetivos y contribución

El objetivo principal de este proyecto es el desarrollo e integración de un modelo de simulación. Específicamente, el modelo a desarrollar es el de RAFI(Robot de Asistencia Física Inteligente), un robot manipulador colaborativo móvil con base omnidireccional, diseñado por la Universidad de Málaga, que se muestra en la figura 1.5. El simulador seleccionado para el proyecto es Gazebo. Se ha elegido este simulador debido a su integración con ROS y su amplia comunidad. Estas dos cualidades facilitan el desarrollo de modelos de simulación gracias a la cantidad de soluciones

[10] ROS components, RB-Kairos, URL: https://www.roscomponents.com/es/manipuladores-moviles/rb-kairos-plus. Fecha de acceso: 16 de noviembre 2024.

[11] El paquete común se encuentra en el siguiente enlace: https://github.com/RobotnikAutomation/rbkairos_common.

[12] El paquete de simulación se encuentra en el siguiente enlace: https://github.com/RobotnikAutomation/rbkairos_sim/tree/melodic-devel.

distintas disponibles en la red. Además, según [13], Gazebo es el segundo mejor simulador disponible después de CoppeliaSim.

Figura 1.5: Foto del Robot de Asistencia Física Inteligente (RAFI). Fuente:[13].

Los objetivos parciales de este proyecto son:

1. Analizar de la estructura del modelo de simulación del brazo robótico Franka Panda. Este brazo es el usado por RAFI, por lo que es importante entender el control de la simulación de este.

2. Simular el brazo robótico Franka Panda para comprobar el correcto funcionamiento del modelo de simulación y analizar los distintos controladores disponibles para el brazo.

3. Adaptar los modelos tridimensionales de RAFI. Se modificará el formato de los modelos para que sean compatibles con Gazebo y se simplificarán las estructuras para asegurar una simulación fluida.

4. Se creará el paquete de simulación, que contendrá archivos de descripción, de inicialización y demás archivos necesarios. El paquete de simulación se añadirá también al repositorio de Github del proyecto, facilitando su accesibilidad.

5. Crear el modelo descriptivo de la base de RAFI. Se usará el formato URDF junto con los modelos tridimensionales mencionados anteriormente.

6. Integrar en un modelo descriptivo único la base de RAFI junto con el brazo robótico Franka Panda.

[13]Github, RAFI, URL: https://github.com/TaISLab/Rafi. Fecha de acceso: 18 de noviembre 2024.

7. Crear los archivos de inicialización. Estos servirán para iniciar la simulación de una forma sencilla y rápida.

8. Implementar un *plugin* para simular el movimiento omnidireccional de la base de RAFI. Gracias a la amplia comunidad de Gazebo existe una gran variedad de *plugins* disponibles.

9. Realizar simulaciones de prueba en entornos simulados que permitirán validar el funcionamiento del modelo completo.

1.4. Estructura de la memoria

La memoria esta estructurada de la siguiente forma: En el capítulo 2 se encuentra una descripción del contexto del proyecto, incluyendo la descripción de RAFI, del brazo Franka Panda y del software utilizado. En el capítulo 3 se detalla lo realizado en el trabajo para la consecución de los objetivos del mismo, como la creación tanto del paquete como del modelo descriptivo de RAFI y su implementación en Gazebo. En el capítulo 4 se describen las distintas pruebas realizadas para comprobar la funcionalidad del modelo y, para terminar, en el capítulo 5 se realiza una síntesis de lo conseguido en el proyecto, así como se describen las posibles líneas futuras de investigación del proyecto.

CAPÍTULO 2

Contexto y Marco Teórico

Contenido

En este capítulo se describe la plataforma móvil RAFI, junto con sus componentes. Después se explica la estructura en ROS del manipulador Franka Emika Panda. Además, se detalla el funcionamiento de ROS, así como distintas herramientas de software utilizadas en el proyecto, como puede ser el propio simulador.

2.1. Descripción de RAFI

El RAFI (Robot de Asistencia Física Inteligente) es un robot colaborativo desarrollado en el marco de diversos proyectos de investigación, en el que han colaborado varios Trabajos Fin de Grado de estudiantes de la Universidad de Málaga (UMA). En la figura 2.1 se muestra la base omnidireccional del robot, desarrollada en el trabajo de fin de grado de Alonso Serrano [18]. El RAFI está diseñado específicamente para su uso en entornos domésticos, actuando como un robot de asistencia. En este sentido, el futuro desarrollo de nuevas funcionalidades se debe llevar a cabo en consonancia con este concepto de uso doméstico y asistencia personal.

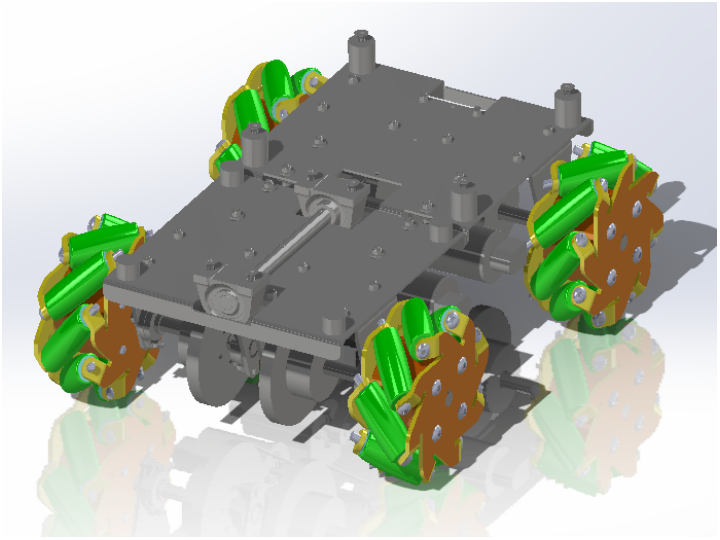


Figura 2.1: Diseño en Solidworks de la plataforma móvil de RAFI.

Un avance destacado en el desarrollo del RAFI fue realizado por Francisco Carbonero en su trabajo de fin de grado [1]. En este proyecto, se llevó a cabo el diseño y montaje del chasis en su totalidad. Además, se implementaron varios programas que permiten controlar el movimiento del robot mediante un joystick. El último avance fue realizado por Rodrigo Castro en su trabajo de fin de grado, en el que se desarrolló una interfaz software para el control de RAFI [19].

El diseño del chasis ha sido concebido específicamente para albergar las baterías en su interior junto con el sistema de gestión de las mismas. Este chasis está compuesto por perfiles de aluminio de 30x30mm, lo que permite una posible reconfiguración de los paneles laterales en caso de ser necesario. Adicionalmente, los paneles están fabricados con un material comercial conocido como Alucubond, que consiste en una estructura tipo sándwich de aluminio y polietileno, lo que contribuye a que sea más ligero y fácil de mecanizar [1].

En el diseño actual, destaca el tipo de ruedas utilizadas, que son del tipo *Mecanum*, cuyo diseño se puede examinar más detenidamente en la figura 2.2. Este tipo de rueda no es convencional, debido a su estructura, y se emplea ampliamente en el ámbito de la robótica, ya que permite un movimiento omnidireccional. La capacidad de movimiento omnidireccional es crucial en este ámbito debido a la

alta maniobrabilidad que ofrece. La maniobrabilidad es una cualidad crucial en entornos domésticos con espacios reducidos. Como se ilustra en la figura 2.2, estas ruedas están compuestas por una serie de rodillos dispuestos en un ángulo de 45 grados. Cuando las ruedas giran en direcciones opuestas, se generan fuerzas laterales que permiten al robot alcanzar velocidades en direcciones angulares [1]. Este principio de movilidad se ilustra en la figura 2.3.

Figura 2.2: Ejemplo de rueda del tipo mecanum. Fuente: [1].

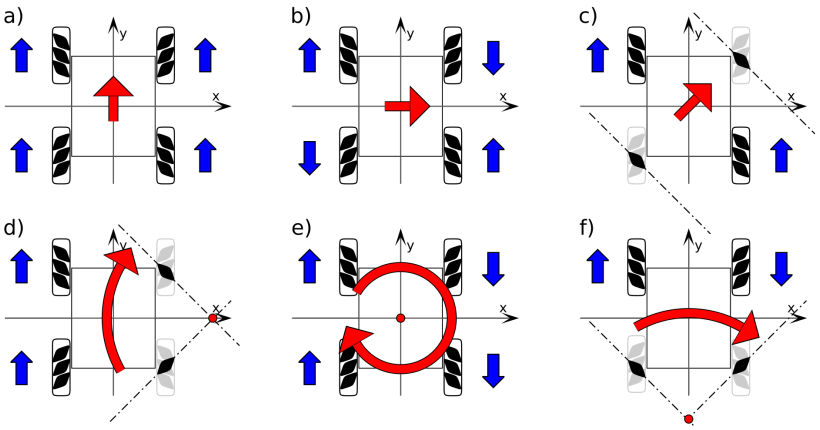


Figura 2.3: Esquema de control de ruedas mecanum. Fuente: [1].

2.2. Estructura de ROS del brazo Franka Panda

Como se ha mencionado previamente, el brazo utilizado por el RAFI es un modelo comercial de la empresa Franka Emika. Este brazo es ampliamente reconocido en el ámbito de la robótica debido a su simplicidad y versatilidad. En el contexto de nuestro proyecto, no es necesario profundizar en las características físicas del robot; sin embargo, sí es relevante analizar la base de ROS asociada a este.

[1]Wikipedia, Mecanum wheel, URL: https://en.wikipedia.org/wiki/Mecanum_wheel. Fecha de acceso: 17 de noviembre de 2024.

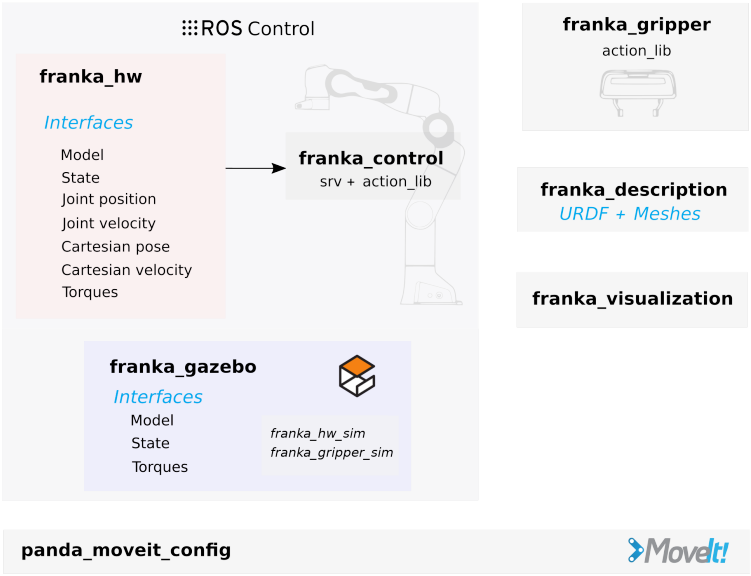


Figura 2.4: Descripción general esquemática de los paquetes existentes en Franka_ROS. Fuente: [2].

En el metapaquete Franka_ROS existen una gran cantidad de paquetes dirigidos al control del robot real, como vemos en la figura 2.4. Sin embargo, nos vamos a centrar en los paquetes dedicados a la simulación del Franka, que incluyen franka_gazebo, franka_description, franka_visualization y panda_moveit_config [3].

Franka_gazebo En este paquete se encuentran los archivos necesarios para lanzar la simulación de gazebo, como distintos archivos de *launch* o incluso distintos mundos de prueba. Añade, además, ciertos *plugins* para simular correctamente el robot, como son *franka_hw_sim*, que facilita la simulación de los *joints* que tienen una interfaz hardware de franka, y *franka_gripper_sim*, que simula el nodo de la pinza del brazo en gazebo, para mandar mensajes a este y poder accionarla.

Franka_description En este paquete se encuentran los archivos de descripción del robot y del efector, en función de la cinemática de estos, sus limites de movimiento y su espacio de colisión. Estas descripciones se encuentran en formato URDF, que se describen más adelante en el capítulo 3.

Franka_visualization Este paquete contiene los nodos que se encargan de publicar el estado del robot para poder visualizarlo mediante RViz.

[2] Github, Franka Control Interface, URL: https://frankaemika.github.io/docs/franka_ros.html. Fecha de acceso: 17 de noviembre de 2024.

[3] Más información sobre los demás paquetes de franka_ros y las características físicas del brazo se encuentra en https://frankaemika.github.io/docs/.

Panda_moveit_config Incluye la configuración necesaria para la integración con Move it! del robot, un software diseñado específicamente para la planificación y simulación de trayectorias de brazos robóticos, lo que permite su control y planificación de movimiento desde una interfaz gráfica como RViz.

2.3. Descripción de ROS

ROS (Robot Operating System) es un *framework* de código abierto diseñado para facilitar el desarrollo y operación de sistemas robóticos. ROS provee de una serie de herramientas y librerías con el fin de integrar elementos robóticos como sensores o actuadores.

ROS se basa en un sistema *peer-to-peer*, en el que son importantes los conceptos de master y nodo. Un nodo es la unidad básica de ROS, es el equivalente a un programa o aplicación, encargados de realizar tareas específicas en el sistema de ROS. Estos nodos se comunican entre ellos y el master a través de topics o de servicios.

Un master es el componente principal de un sistema ROS, ya que actúa como núcleo del mismo. Es de vital importancia en un sistema de ROS, ya que sin un master no es posible la comunicación entre nodos debido a que el master facilita la localización y conexión entre nodos.

La comunicación entre nodos de ROS se puede dar por dos formas distintas, por medio de topics o servicios.

Un topic es el canal de comunicación entre nodos. Los nodos publican la información en el topic correspondiente y así otros nodos se suscriben a este topic y pueden leer la información publicada. La comunicación a través de topics es unidireccional, es decir, los nodos que publican información en el topic no se pueden suscribir a él y, por lo tanto, no pueden leer la información que se ha publicado.

Por otra parte, un nodo puede ofrecer un servicio, ejerciendo una tarea determinada. Otros nodos se pueden comunicar con este server de manera bidireccional, a través de una solicitud del servicio. Esta manera de comunicación es bastante útil para aquellas operaciones que requieran confirmación.

Otro concepto que es importante para el desarrollo de nuestro proyecto es el de las transformaciones. En general, es importante para los sistemas robóticos trackear relaciones espaciales, bien para poder localizarse en el mundo o bien para realizar cierto control. ROS provee del paquete *tf* para este fin. El sistema *tf* construye un árbol dinámico de transformaciones que relaciona todos los marcos de coordenadas del sistema. Según se distribuye la información desde los distintos subsistemas del robot, el sistema *tf* calcula las transformaciones necesarias entre los nodos deseados [20].

ROS permite además lanzar varias veces los mismos nodos pero para distintos procesos mediante el uso de *namespaces*. Esto es realmente útil para simulaciones con dos robots del mismo tipo o para la operación de robots humanoides que usan dos brazos iguales [20].

2.4. Gazebo

Gazebo es un simulador de robótica tridimensional de código abierto que permite recrear entornos para probar el desarrollo de robots en tiempo real. Este simulador surge de la necesidad de simular robots en diversos entornos, dado que simuladores anteriores, como Webots o OpenSim, carecían de la capacidad para personalizar los escenarios en los que se llevaban a cabo las simulaciones. Gazebo también reproduce la física del mundo real, lo que permite simular fenómenos como la gravedad, colisiones y otras interacciones del robot con los objetos del entorno simulado [12]. Además, Gazebo integra una variedad de motores de física, como ODE, DART, Bullet y Simbody. Una ventaja significativa de este simulador es su activa comunidad de usuarios, que desarrollan robots y entornos para Gazebo, que pueden utilizarse como ejemplos [13].

Para tener una mejor simulación de los distintos tipos de robots existen distintos *plugins*, que son módulos adicionales que modifican o añaden funcionalidades a la simulación en sí, como pueden ser la simulación de sensores o motores. Estos *plugins* pueden ser proporcionados por el entorno de gazebo o desarrollados de cero para mayor personalización, lo que permite su adaptación a tareas específicas.

2.4.1. Plugins de Gazebo

Un *plugin* es un conjunto de código, generalmente escrito en C++, que permite realizar modificaciones en las físicas o añadir nuevas funcionalidades [21], como la incorporación de sensores. Dependiendo de a que tipo de elemento se aplica se puede decir que existen seis tipos de *plugin* diferenciados.

- Los *plugins* de sistema se especifican desde el terminal de comandos y se cargan durante la puesta en marcha [21].
- Los *plugins* de mundo (*world plugins*) modifican y controlan un entorno simulado.
- Los *plugins* de sensor implementan sensores en la simulación.
- Los *plugins* de modelo (*model plugins*) se encargan de añadir funcionalidades a los modelos de los robots simulados.
- Los *plugins* visuales modifican aspectos visuales en la simulacion.
- Los *plugins* de GUI(siglas en inglés de *Graphic User Interface*, traducido como Interfaz gráfica de usuario) se usan para mostrar más información en pantalla a través de interfaces.

Para el modelo de simulación desarrollado en este proyecto, será necesario contar con un *plugin* de modelo que facilite la simulación del movimiento omnidireccional del RAFI. En este sentido existen distintas soluciones que se pueden implementar.

Una opción inicial lógica sería utilizar el *plugin* de Gazebo denominado *planar move*. Sin embargo, este *plugin* interfiere con las físicas del entorno de Gazebo, lo que provoca que el movimiento del brazo sea lento y que la gravedad parezca tener un efecto mínimo en el modelo. Esto se puede deber a que el *plugin* actualiza la velocidad del modelo en el motor de físicas, lo que puede causar conflictos con el propio motor. Este problema es conocido por la comunidad de Gazebo, pero en la fecha de realización de este proyecto no existe una solución definitiva. Por lo tanto, esta opción se descarta en nuestra implementación.

Por esta razón, finalmente se ha optado por el *plugin force based move*, desarrollado por la Universidad de Darmstadt, aunque con ligeras modificaciones para facilitar la resolución de errores. Este *plugin* simula el movimiento aplicando fuerzas directamente sobre el modelo en Gazebo.

Su implementación es sencilla; dentro del archivo URDF, se debe añadir la etiqueta correspondiente a Gazebo y, a continuación, incluir una etiqueta de *plugin*. En esta última, se especificará primero el nombre del *plugin*, el cual puede no coincidir con el nombre del archivo, y se indicará el archivo .so del *plugin*. Una vez dentro de la etiqueta de *plugin*, es posible modificar los diferentes parámetros del mismo. El siguiente código es un ejemplo de implementación de un *plugin* cualquiera.

```
<gazebo>
<plugin name='(Nombre del plugin)' file='(Archivo .so del plugin)'>
(Parametros del plugin)
</plugin>
</gazebo>
```

2.5. RViz

RViz (*ROS Visualization*) es una herramienta de visualización tridimensional integrada en ROS que permite la representación en tiempo real de robots, así como de una amplia variedad de datos relacionados con el estado y el entorno de dichos robots. Su uso es común en el ámbito de la robótica debido a su capacidad para visualizar datos de sensores, planificar trayectorias y controlar interactivamente robots [20].

En el contexto de nuestro estudio, RViz adquiere una importancia particular debido a su integración con MoveIt! [22]. Esta integración nos permite utilizar RViz como interfaz gráfica de movimiento, lo que facilita de manera intuitiva el movimiento del brazo robótico.

2.6. Inicialización de la simulación

Para poder simular el brazo en movimiento correctamente, es necesario pasar ciertos datos a gazebo y activar ciertos nodos, lo cual puede ser engorroso. Es por esto que es una buena práctica realizar un archivo launch, para poder iniciar la simulación completa con solo una línea de código.

El archivo launch es una herramienta que permite inicializar varios nodos y configuraciones en una sola línea de comando [23]. En este archivo se establecen por lo tanto los nodos a iniciar en ros y el orden de inicialización, además de establecer los parámetros necesarios para ciertos nodos. Para iniciar un archivo launch se utiliza el siguiente comando de ROS.

```
$ roslaunch <Nombre del paquete> <Nombre del archivo launch>
```

Ahora bien, si se desea iniciar un nodo cualquiera, fuera del archivo launch, se utiliza el siguiente comando.

```
$ rosrun <Nombre del paquete> <Nombre del nodo>
```

En el archivo launch que se ha creado para el proyecto se ha especificado la inicialización de los siguientes nodos:

- Gazebo: Este nodo constituye el núcleo de la simulación. Su función es iniciar Gazebo junto con su maestro y definir el entorno deseado. Adicionalmente, se configura para que la simulación comience en estado de pausa, lo cual permite realizar configuraciones iniciales si fuera necesario. A diferencia de otros nodos, este se lanza mediante un proceso ligeramente distinto, ya que el paquete gazebo_ros incluye un archivo de lanzamiento predefinido para un mundo vacío. Por lo tanto, se incorpora este archivo de lanzamiento en el propio mediante una etiqueta de inclusión (*include*).

- Model Spawner: Este nodo es responsable de generar el modelo a partir del archivo URDF y se encuentra en el paquete gazebo_ros. Para su correcto funcionamiento, es necesario proporcionarle ciertos argumentos, entre ellos, la descripción del robot en formato URDF y las posiciones iniciales de las articulaciones (*joints*) del brazo robótico. Además, se incluye un argumento denominado *unpause* que permite que, una vez inicializado el modelo, se inicie automáticamente la simulación.

- Gripper spawner: Este nodo, perteneciente al paquete controller_spawner, permite la creación del controlador específico para la mano del brazo robótico. Su ejecución está condicionada a la especificación del uso de la mano, la cual se puede activar desde la línea de comandos. Como parámetros, se define el controlador a emplear, en este caso el propio del sistema Franka, y se establece el valor de *respawn* en *false*, lo que indica que, en caso de fallo del nodo, este no se reiniciará automáticamente.

- Controller spawner: Este nodo es similar al anterior; sin embargo, en este caso se utiliza para cargar el controlador del brazo robótico. Franka proporciona varios controladores para el brazo, aunque en este proyecto se emplea el controlador basado en impedancias cartesianas, el cual facilita su manipulación posterior mediante MoveIt! y RViz. Además, este nodo requiere esperar

a que el nodo franka_state_controller esté activo, condición que se establece explícitamente como argumento en el código.

- Robot state publisher y joint state publisher: Estos dos nodos tienen la función de recopilar, calcular y distribuir la información de posición, velocidad y esfuerzo de las articulaciones del robot. En particular, el nodo robot_state publica la posición y orientación de las articulaciones en formato TF, mientras que el nodo joint_state publica la posición, velocidad y esfuerzo de las articulaciones en un topic específico, joint_states. Ambos nodos son esenciales para la simulación, ya que permiten la actualización en tiempo real del estado del robot. Para el nodo joint_state, se configura adicionalmente para leer los estados del brazo y la pinza de los controladores definidos previamente, y se ajusta su frecuencia de actualización.

- Interactive marker: Este nodo crea en marcador interactivo que usaremos en Rviz para poder manejar libremente el brazo robot. Este nodo es parte de la integración con Move it! que posee el Franka.

- Rviz: Por último, este nodo lanza RViz, una herramienta de visualización utilizada para la integración con MoveIt!. Para un proyecto nuevo es necesario abrir Rviz y configurar los distintos parámetros de la visualización. Esta configuración se emplea como argumento de inicialización del nodo RViz.

Con esto quedaría especificar los parámetros de los nodos y la simulación. Es conveniente recoger todos los argumentos dentro del launch, ya que podemos establecer un valor por defecto, lo que haría incluso más sencilla la inicialización de la simulación.

CAPÍTULO 3

Metodología

Contenido

En este capítulo, se detallan los desarrollos técnicos realizados en este proyecto. En primer lugar, se aborda la creación del paquete de ROS que contiene el modelo de simulación, así como la configuración del mismo junto con sus dependencias. Seguidamente, se describe el contenido de un modelo de descripción de un robot cualquiera. Después, se detalla la estructura del modelo descriptivo del brazo Franka Panda. Además, se describe el desarrollo realizado para la creación del modelo descriptivo de la base móvil de RAFI. Para terminar, se detalla el modelo descriptivo de cuerpo completo de RAFI, analizando los esquemas de control que permiten la simulación del mismo.

3.1. Creación del paquete de ROS

Para la realización del proyecto, se ha desarrollado un paquete de ROS, accesible desde su repositorio de GitHub [1], en el que se encuentra el modelo de cuerpo completo de RAFI. La creación de un paquete de ROS cualquiera se lleva a cabo mediante la ejecución del siguiente comando en la terminal del sistema:

```
$ catkin_create_pkg <nombre del paquete> [dependencia 1] [dependencia2]
```

La especificación de la lista de dependencias del paquete es, en un principio, opcional; sin embargo, en este caso resulta esencial definirlas adecuadamente. Existen diferentes tipos de dependencias, dependiendo de si son necesarias durante la fase de construcción o en la ejecución del paquete. Entre las dependencias más relevantes para el paquete se destacan *rviz*, *xacro*, *joint_state_publisher* o *robot_state_publisher*.

Para recrear las simulaciones presentadas o realizar nuevas simulaciones utilizando el modelo, no es necesario crear un paquete desde cero como se ha descrito anteriormente. En su lugar, es suficiente con clonar el repositorio de GitHub en el directorio deseado utilizando el siguiente comando:

```
$ git clone <URL del repositorio>
```

La estructura del paquete, que se detalla en la figura x, se divide en distintas carpetas dedicadas a distintos ficheros.

- En la carpeta *description* se encuentran los modelos descriptivos tanto de la base de RAFI como del modelo completo, los cuales se detallan en las siguientes secciones.
- La carpeta *worlds* contiene distintos entornos de simulación diseñados para cada uno de los modelos.
- En la carpeta *launch* se localizan los archivos de inicialización de las simulaciones de los diferentes modelos.
- En la carpeta *meshes* se almacenan los modelos tridimensionales necesarios para la correcta visualización de los modelos simulados.
- En la carpeta *rviz* se encuentran los archivos de configuración para la visualización en RViz.
- En la carpeta *franka_robot* se incluyen tanto los modelos descriptivos como los modelos tridimensionales del brazo robótico Franka Panda.

[1] Disponible en el siguiente enlace https://github.com/TaISLab/rafi_sim.

3.2. Modelo descriptivo de un robot

La parte más crucial de un modelo de simulación de un robot es el archivo URDF (*Unified Robot Description Format*). Este archivo, basado en XML, se utiliza para describir las diversas partes del robot, especificando su geometría y la interacción entre sus componentes. Esto permite que Gazebo identifique y simule de manera efectiva los elementos del robot

En un archivo URDF se pueden identificar dos tipos de elementos diferenciables: los *links* y los *joints*.

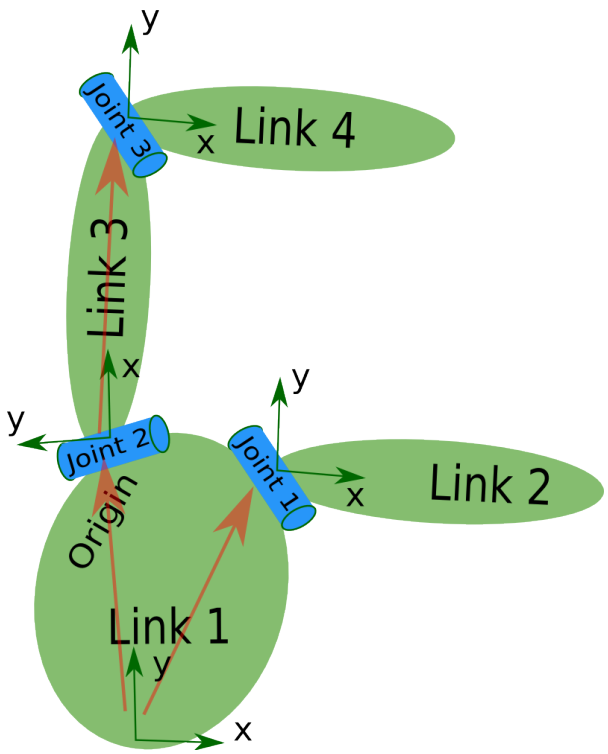


Figura 3.1: Ejemplo de estructura de un robot cualquiera definido con *links* y *joints*. Fuente: [2].

Los *links* definen componentes específicos del robot, como un brazo o una rueda. Para cada *link*, se especificarán sus características físicas, que pueden incluir la forma, la masa y el material, así como aspectos como el color y la textura. Dentro de un *link*, se establecen tres etiquetas distintas: *visual*, *collision* e *inertial*.

Dentro de la etiqueta *visual*, se definen las características visuales del componente mediante distintas etiquetas. La etiqueta *geometry* se utiliza para establecer la geometría del componente, la cual puede ser cúbica (*box*), cilíndrica (*cylinder*) o esférica (*sphere*). Además, si se desea modificar el color o la textura del componente, esto se realiza a través de la etiqueta *material*. A través de la etiqueta *collision*, se define la forma que tendrá el componente en términos de colisión. En la mayoría

[2]Wiki ROS, URDF, URL: http://wiki.ros.org/urdf/Tutorials/Create your own urdf file. Fecha de acceso: 18 de noviembre de 2024.

de los casos, esta forma coincidirá con la del componente; sin embargo, para componentes con geometrías complejas, puede ser más eficiente utilizar una forma de colisión equivalente más simple, con el fin de optimizar el uso de recursos en la simulación. En la etiqueta *inertial*, se especifican la masa del componente y su matriz de inercia. Esta matriz no resulta difícil de determinar, dado que para geometrías sencillas su cálculo es bien conocido.

En estas tres etiquetas, es posible definir el origen de coordenadas correspondiente a cada componente. Esta definición es de vital importancia, ya que establece las relaciones entre los distintos componentes del robot.

Los *joints* establecen el tipo de conexión entre dos componentes. Esta conexión puede ser de diferentes tipos: fija (*fixed*), que no permite movimiento entre los *links*; continua (*continuous*), que permite rotación indefinida; revolutiva (*revolute*), que permite rotación limitada; y prismática (*prismatic*), que permite movimiento lineal limitado. Para que un *joint* esté correctamente definido es necesario especificar tanto el *link* padre como el *link* hijo, así como el origen de esta conexión.

3.3. Modelo descriptivo de Franka Panda

Como se ha mencionado anteriormente, los modelos del brazo Franka Panda se encuentran en el paquete *franka_description*. Este paquete se organiza en dos carpetas principales: *meshes* y *robots*.

- En la carpeta *meshes* se almacenan los modelos 3D del brazo, los cuales se dividen según su uso para visualización o colisión. Los modelos destinados a la visualización en Gazebo están en formato COLLADA, lo que permite incluir la textura del modelo. Por otro lado, los modelos utilizados para colisión se encuentran en el formato estándar STL.

- En la carpeta *robots* se encuentran los archivos URDFs de los distintos modelos de brazos de Franka, organizados en subcarpetas. La carpeta *common* contiene URDFs básicos que definen el brazo, la pinza y el conjunto completo, los cuales pueden ser modificados según el modelo de Franka que se requiera. En las carpetas *fr3* y *panda* se encuentran los URDFs de estos robots, que son versiones modificadas de los modelos básicos utilizando Xacro. También existe una carpeta denominada *dual_panda* que incluye un URDF de ejemplo en el que se describen dos brazos Panda de manera simultánea.

Para una mejor comprensión de la forma en la que interactúan estas carpetas, se ha creado un esquema, visible en la figura 3.2. En este esquema se puede apreciar de una forma visual como se organizan los modelos de franka_ros.

Para este proyecto se ha incluido esta carpeta en el repositorio de simulación de RAFI, a fin de evitar en la medida de lo posible problemas de dependencias y simplificar la descarga de los paquetes.

La estructura del modelo descriptivo de Franka Panda presenta una configuración relativamente sencilla. En concreto, su URDF está compuesto por una secuencia de *links* conectados mediante diferentes *joints* dispuestas de manera secuencial. La mayoría de estos *joints* son de tipo revolutivo,

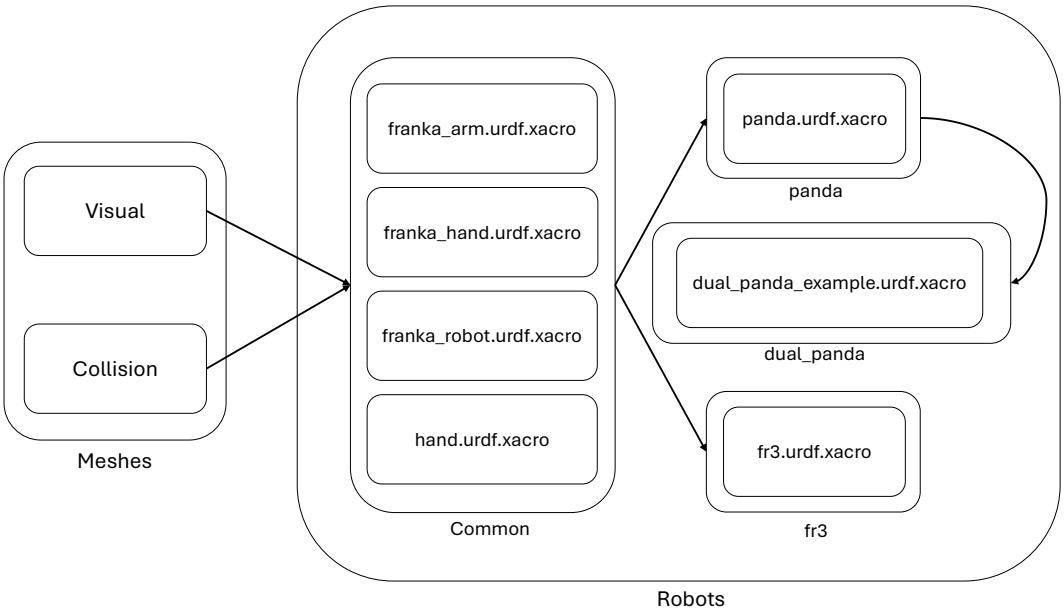


Figura 3.2: Esquema de organización de los modelos descriptivos y meshes de franka_ros.

lo que permite el movimiento del brazo en varios ejes. Además, el modelo incluye un *joint* de tipo fijo en el extremo del brazo y dos *joints* prismáticos en la pinza del mismo.

Para el control del brazo Franka Panda existen distintos controladores, cada uno con un esquema de control diferenciado. El controlador recomendado para el control interactivo del brazo es el *cartesian_impedance_example_controller*, por lo que será el usado en la simulación de cuerpo completo de RAFI. Aún así, es de especial interés mencionar los demás controladores de prueba.

- *cartesian_pose_example_controller*: basa su control en la pose del efector final, aplicando un movimiento cartesiano, sin tener en cuenta fuerzas externas ni aplicando compensación dinámica.

- *cartesian_velocity_example_controller*: regula la velocidad cartesiana del efector final, aplicando velocidades en los ejes cartesianos correspondientes.

- *elbow_example_controller*: se encarga de controlar únicamente la posición del codo durante el movimiento del efector final.

- *force_example_controller*: regula directamente los pares de las articulaciones, permitiendo la gestión de fuerzas externas.

- *joint_impedance_example_controller*: regula los pares aplicados en las articulaciones ajustados por la rigidez y amortiguación de las mismas.

- *joint_position_example_controller*: ajusta la posición de las articulaciones del brazo a partir de una posición inicial definida.
- *joint_velocity_example_controller*: regula la velocidad de las articulaciones siguiendo un perfil sinusoidal para evitar cambios bruscos de velocidad.
- *model_example_controller*: este controlador simplemente recoge información importante del modelo para mostrarla en la consola.
- *teleop_joint_pd_example_controller*: diseñado para el control de dos brazos robóticos, en una configuración de líder-seguidor. En esta configuración, uno de los robots se mueve libremente, mientras que el otro copia sus movimientos. La regulación de este controlador se lleva a cabo mediante un regulador proporcional derivativo (PD), que ajusta tanto la posición como la velocidad de las articulaciones del brazo.

Además de estos controladores, existe una versión de doble brazo del *cartesian_impedance _example_controller*. El *dual_arm_cartesian_impedance_example_controller* regula simultáneamente dos brazos robóticos con el mismo esquema de control.

3.4. Modelo descriptivo de la base de RAFI

Para el modelo descriptivo de la base primeramente se ha desarrollado un archivo URDF *rafi.urdf* que modela de manera simplificada la base móvil del RAFI. Este archivo sirve como referencia y ejemplo para la creación de otros modelos básicos. A continuación, se presenta un ejemplo de un *link*, en este caso el correspondiente al chasis, que permite ilustrar la estructura general de un *link* en el contexto del diseño de robots.

```
<link name="Chasis" >
    <visual>
        <geometry>
            <box size="0.65 0.5 0.27"/>
        </geometry>
    </visual>
    <collision>
        <geometry>
            <box size="0.65 0.5 0.27"/>
        </geometry>
    </collision>
    <inertial>
        <mass value="1.0"/>
        <origin xyz="0 0 0"/>
```

```
        <inertia ixx="0.1" ixy="0.0" ixz="0.0" iyy="0.1" iyz="0.0" izz="0.1"/>
    </inertial>
    <origin xyz="0 0 0.375" rpy="0 0 0"/>
</link>
```

Dado que el chasis del RAFI presenta una forma rectangular, se ha definido tanto en el elemento visual como en el de colisión como una geometría de tipo *box*, especificando sus dimensiones en metros. Adicionalmente, se han establecido los parámetros correspondientes a la inercia y la masa de este componente.

Asimismo, es necesario definir los *joints* que conectarán las ruedas con el chasis, los cuales corresponderán al siguiente tipo:

```
<joint name="Rueda1Chasis" type="continuous">
    <parent link="Chasis" />
    <child link="Rueda1" />
    <origin xyz="0.225 -0.21 -0.29" rpy="0 0 0" />
    <axis xyz="0 1 0"/>
</joint>
```

Este *joint* se caracteriza por su simplicidad. Se define como un *joint* de tipo *continuous*, lo que permite el libre giro de la rueda. Además, se especifica la relación entre la rueda y el chasis. Este mismo tipo de *joint* se aplicará a las cuatro ruedas, variando únicamente el origen para cada una de ellas.

El *tf_tree* del modelo puede visualizarse utilizando la herramienta de ROS *urdf_to_graphiz*, la cual permite verificar visualmente las relaciones establecidas en el modelo y comprobar su corrección. Esto se ilustra en la figura 3.3.

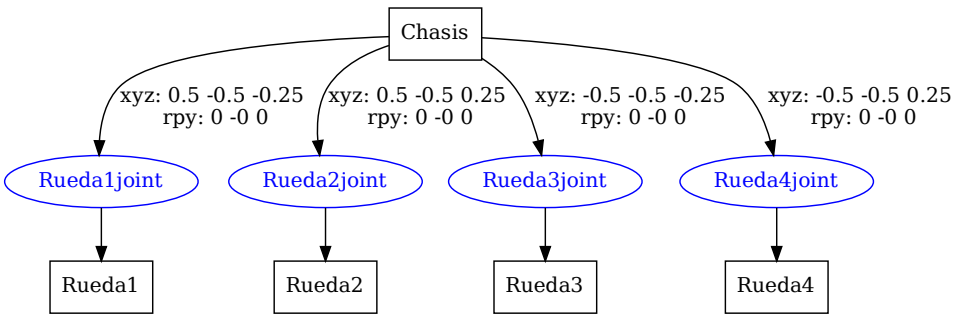


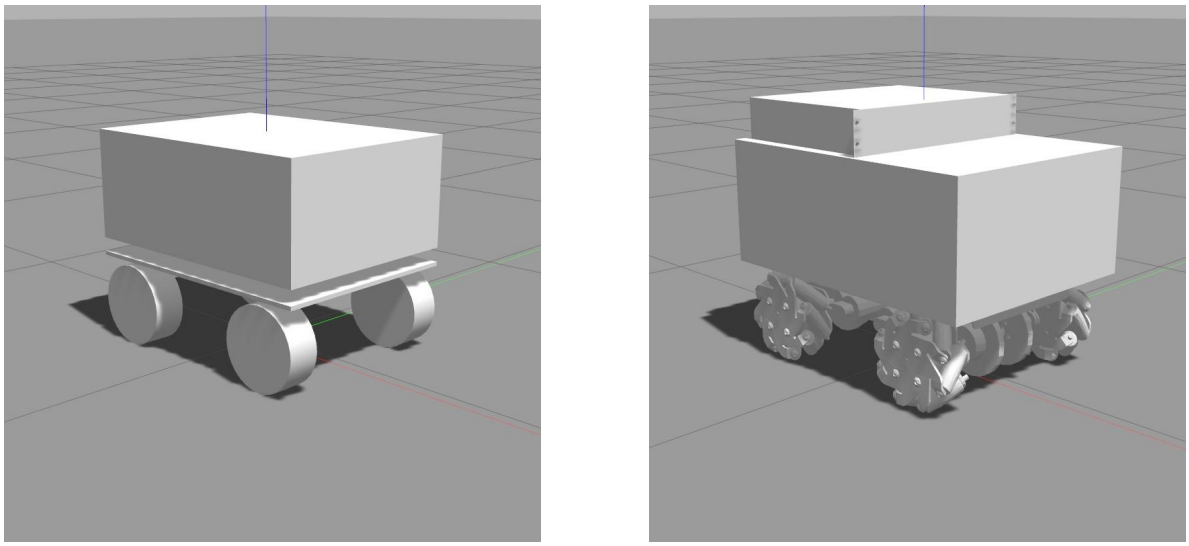
Figura 3.3: *TF Tree* del modelo simplificado del RAFI.

Aprovechando los modelos 3D de la base, es posible mejorar el aspecto visual de la simulación mediante la implementación de algunos ajustes menores. En primer lugar, es necesario que los

modelos 3D estén en un formato compatible, como el archivo con extensión *.stl*. Estos modelos deben ubicarse en una carpeta dentro del paquete, denominada *meshes*, para facilitar su acceso.

En el archivo URDF, se sustituirá la forma definida en la etiqueta *visual* por un *mesh*, especificando el archivo correspondiente. De esta manera, se obtendrá un modelo 3D más complejo en la simulación. En cuanto a la colisión, no es necesario realizar cambios en principio, ya que una forma de colisión más simple será suficiente. Un ejemplo de como añadir un *mesh* al modelo se presenta a continuación:

```
<visual>
  <geometry>
    <mesh filename="package://(Ruta del archivo)/(Nombre del modelo 3d).stl"/>
  </geometry>
</visual>
```



(a) (b)

Figura 3.4: Diferencia entre un modelo básico (a) y un modelo usando *meshes* (b).

Con el modelo detallado ya realizado, como se detalla en la figura 3.4, se ha graficado también su *tf_tree*, que se puede observar en la figura 3.5. Al añadir los modelos 3D de la base de RAFI, el archivo de descripción se ha simplificado considerablemente.

El control de la base de RAFI se realiza directamente con el *plugin* de movimiento. El *plugin* usado, como se ha comentado anteriormente, es el *force_based_move* desarrollado por la universidad de Darmstadt. El funcionamiento de el control mediante el *plugin* se detalla más adelante.

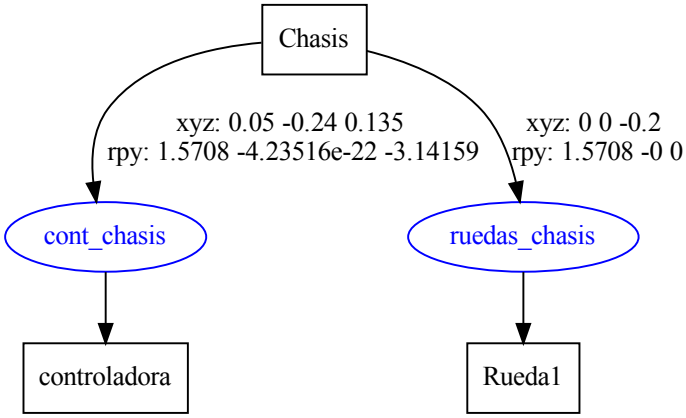


Figura 3.5: *TF Tree* del modelo de la base de RAFI.

3.5. Modelo descriptivo de cuerpo completo de RAFI

Gracias al uso de Xacro, la integración del modelo descriptivo completo del RAFI se simplifica considerablemente. Xacro permite la utilización de macros en archivos XML, lo que facilita la construcción y manipulación de modelos descriptivos complejos. En este caso, el modelo descriptivo del brazo Franka Panda ya está expresado como una macro, al igual que el modelo de la base, que también ha sido diseñado como tal. De este modo, el modelo completo del RAFI se compone únicamente de dos macros, lo que facilita su implementación y gestión.

Para definir una macro en XML se debe usar la siguiente estructura de código:

```
<xacro:macro name="(Nombre de la macro)">
(Definición de la macro)
</xacro:macro>
```

Ahora bien, para usar una macro en un URDF cualquiera, se deben añadir las siguientes líneas de código:

```
<xacro:include filename="(Ubicación del archivo donde se define la macro)">
<xacro:(Nombre de la macro)>
```

Además, para cualquier XML en el que se use xacro, además de cambiar la extensión de *.urdf* a *.urdf.xacro*, se debe modificar la definición del robot, que es la primera línea del urdf.

```
<robot name="(Nombre del robot)" xmlns:xacro="http://wiki.ros.org/xacro">
```

El control del modelo de cuerpo completo de RAFI se realiza por separado, con cada parte del modelo operando bajo un esquema de control independiente.

El control de la base se realiza directamente mediante un *plugin* que actúa como controlador. Este *plugin* recibe comandos de velocidades lineales y angulares, los cuales se traducen en fuerzas de empuje aplicadas al modelo de la base. Los comandos de velocidad se publican en el tópico *cmd_vel*, ya sea directamente a través del uso del comando de ROS *rostopic pub* en un terminal o mediante un programa externo. En este caso, se ha empleado un paquete de teleoperación, cuyo funcionamiento se detallará en el capítulo 4. El esquema de control se observa en la figura 3.6.

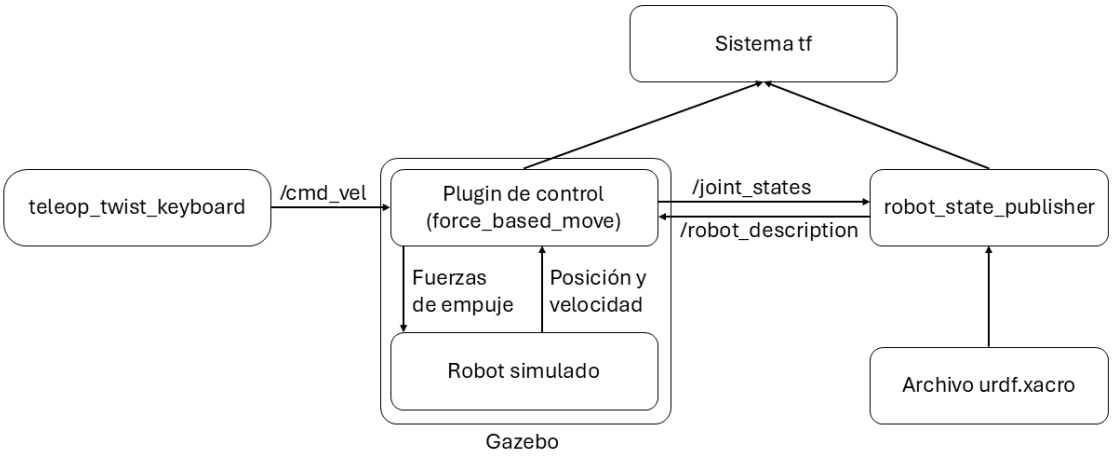


Figura 3.6: Esquema simplificado del control de la base de RAFI.

Este esquema de control de la base difiere del implementado la realidad, debido al funcionamiento particular del *plugin*. En un entorno real, el control se lleva a cabo mediante controladores en las ruedas, los cuales traducen comandos de velocidad en rotación de las ruedas [3].

Por otro lado, el control del brazo Franka Panda se lleva a cabo con distintos controladores, cada uno con características específicas. En este caso, se utiliza el controlador *cartesian_impedance_example_controller*, que implementa un esquema de control basado en impedancias cartesianas. Este enfoque se fundamenta en un modelo de sistema masa-resorte-amortiguador para ajustar el comportamiento del brazo frente a fuerzas externas, controlando así la posición y orientación del efector final. El control cartesiano se realiza con un controlador proporcional-derivativo (PD), que ajusta el par de las articulaciones en función del error cartesiano. También se implementa un control del espacio nulo para mantener la configuración deseada una vez que el efector ha alcanzado su posición objetivo.

Para llevar a cabo el control, el controlador accede a las interfaces del modelo, el estado y los esfuerzos del robot con el fin de obtener su estado inicial. Además, se suscribe al topic *equilibrium_pose*,

[3] Más información sobre el control del robot real se encuentra en el trabajo de fin de grado de Francisco Carbonero [1] y en el trabajo de fin de grado de Rodrigo Castro [19].

donde se publica la posición objetivo a la que debe llegar el efector final. Un esquema simplificado del control de Franka se observa en la figura 3.7.

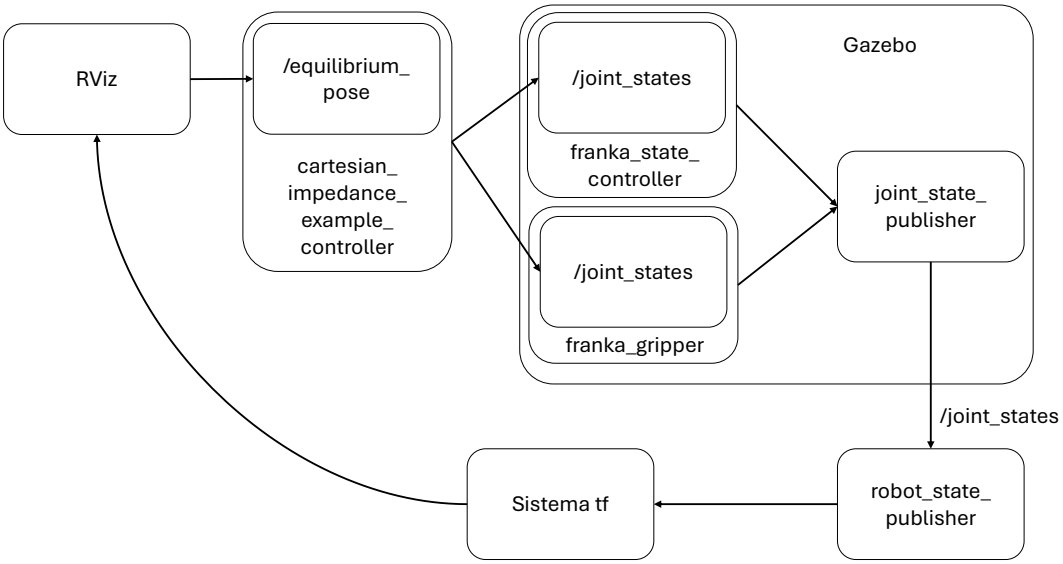


Figura 3.7: Esquema simplificado del control del brazo Franka Panda.

CAPÍTULO 4

Experimentos y Resultados

Contenido

En este capítulo se realiza un análisis del rendimiento de los modelos de simulación detallados en el capítulo anterior. Se incluyen los comandos necesarios para replicar las simulaciones realizadas. Para cada simulación, se ha realizado una grabación, disponibles en el Github del proyecto, accesible en el siguiente enlace: https://github.com/TaISLab/rafi_sim. Se utilizan los fotogramas más destacados para ilustrar las simulaciones en esta memoria.

4.1. Entorno experimental

Los experimentos que se detallan a continuación son simulaciones de cada una de las partes que conforman el modelo de cuerpo completo de RAFI. Para estas simulaciones se ha utilizado el simulador Gazebo en su versión 11. El sistema operativo que se ha usado es Ubuntu en su versión 20.04, con ROS en su versión *Noetic*.

4.2. Protocolo de experimentación

Para llevar a cabo estas simulaciones es necesario primero descargarlas desde GitHub. Todo lo relativo a la simulación de RAFI se encuentra en el repositorio de este proyecto, mientras que para la simulación del Franka se recomienda directamente descargar el repositorio del propio Franka. También es necesario instalar las dependencias de cada uno de los paquetes, generalmente especificadas en el repositorio de GitHub.

Es necesario crear una carpeta que funcione como *workspace*, para descargar los paquetes dentro de esta. Esto se realiza con los siguientes comandos:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Con el comando *catkin_make* se compilan los paquetes dentro del *workspace*, por lo que es necesario ejecutarlo cada vez que se realice un cambio en alguno de los paquetes. Es importante además ejecutar el siguiente comando en cada nuevo terminal:

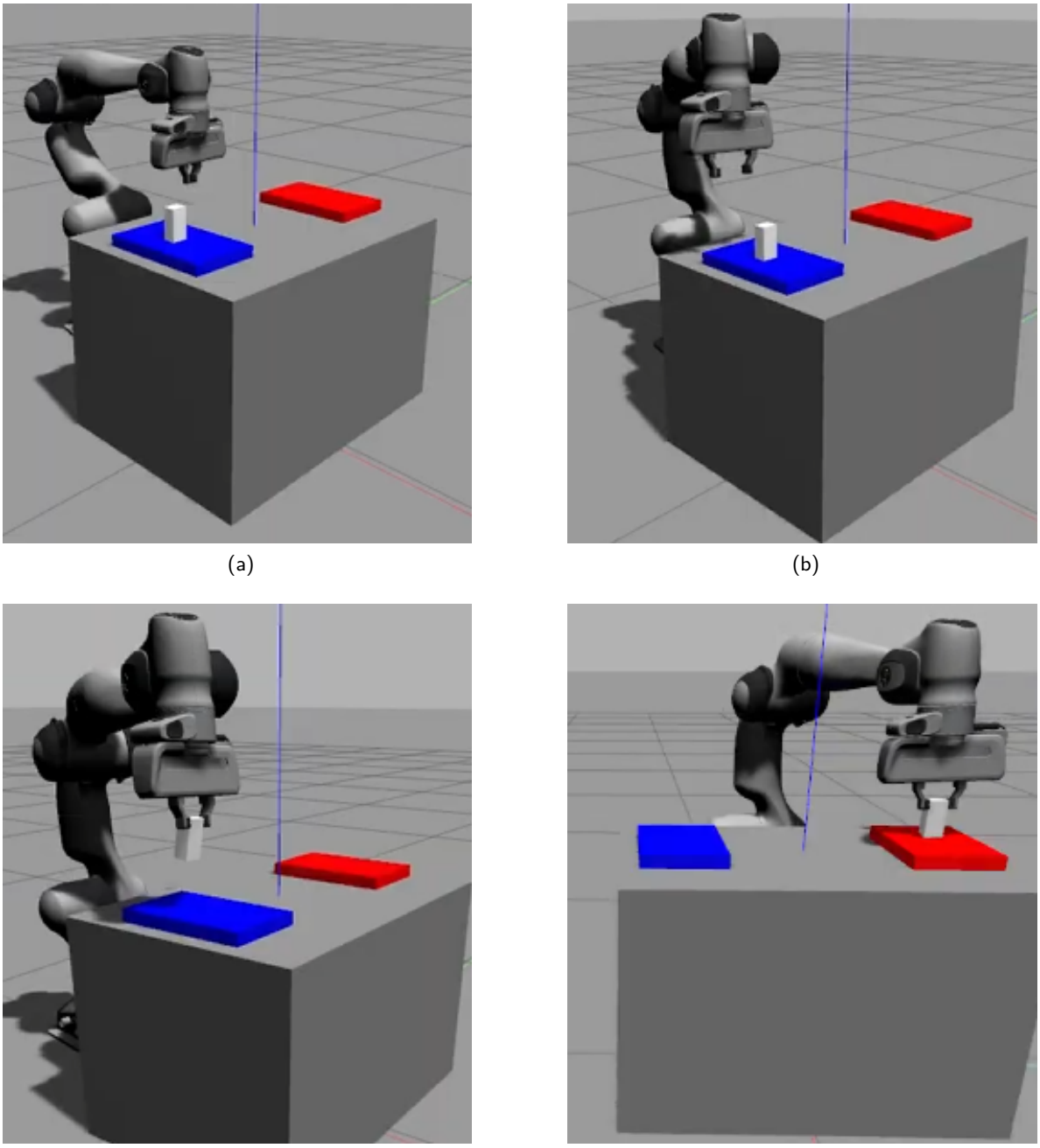
```
$ source devel/setup.bash
```

Así, el entorno estaría correctamente configurado para llevar a cabo las simulaciones que se detallan a continuación.

4.3. Simulación del brazo Franka Panda

La simulación del brazo Franka Panda se realiza en el entorno de prueba que proporciona el paquete. En este entorno encontramos una piedra, que se debe llevar de una plataforma a la otra. Para lanzar la simulación se utiliza el siguiente comando:

```
roslaunch franka_gazebo panda.launch x:=-0.5 \
world:=$(rospack find franka_gazebo)/world/stone.sdf \
controller:=cartesian_impedance_example_controller \
rviz:=true
```



(a) (b)

(c) (d)

Figura 4.1: Secuencia de simulación del brazo Franka Panda.

Se utiliza el controlador de impedancia cartesiana ya que permite la manipulación del brazo robot con RViz. Para abrir la pinza, se publica la acción de mover mediante el siguiente comando:

```
rostopic pub --once /franka_gripper/move/goal \
franka_gripper/MoveActionGoal "goal: { width: 0.08, speed: 0.1 }"
```

Una vez el brazo se coloca sobre la piedra, se puede cerrar la pinza publicando esta vez un mensaje de *grasp*. Es necesario ajustar la fuerza de cierre de la pinza para asegurarse de que la piedra no se caiga. El mensaje se publica de la siguiente manera:

```
rostopic pub --once /franka_gripper/grasp/goal \
franka_gripper/GraspActionGoal \
"goal: { width: 0.03, epsilon:{ inner: 0.005, outer: 0.005 } ,\
speed: 0.1, force: 5.0}"
```

Si la acción de *grasp* ha funcionado bien, se puede entonces mover el brazo hasta la otra plataforma, donde se deja la piedra. Para parar el agarre se vuelve a enviar un mensaje esta vez de *stop*, lo que para el agarre de la pinza o otra acción que se esté ejecutando. El comando sería el siguiente:

```
rostopic pub --once /franka_gripper/stop/goal franka_gripper/StopActionGoal {}
```

Como se ha comentado anteriormente, el movimiento del brazo Franka Panda se puede realizar con RViz. Esto es gracias a la integración del brazo con el programa MoveIt!, que también se ha detallado anteriormente. Como se puede apreciar en la figura 4.2, la interfaz de MoveIt! es bastante sencillo, con posibilidad de mover el efector final en tres dimensiones y rotar sobre los tres ejes. Los movimientos que se realicen con la interfaz se escriben en el topic *equilibrium_pose*. Se han recogido los cambios que se producen en el topic en la gráfica x. No se han representado los giros de la orientación ya que no son necesarios en la prueba, aunque sería posible. La adquisición de estos datos se ha realiza con la herramienta de ROS *rqt_multiplot* y la representación con MatLab.

Figura 4.2: Interfaz de movimiento en RViz del brazo Franka Panda.

Estos cambios en la posición del punto de equilibrio pasan al controlador, el cual calcula las fuerzas que deben ejercer los actuadores. Esto se puede observar en la figura 4.4, en la que se detallan

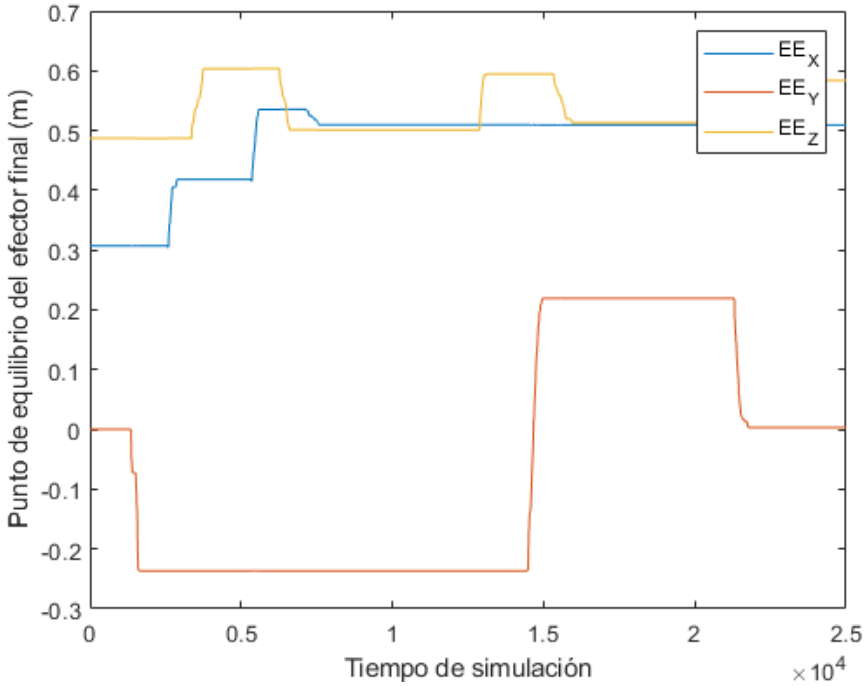


Figura 4.3: Gráfica de posición del punto de equilibrio del brazo Franka Panda en simulación.

los nodos y topics activos en la simulación. Para dibujar este grafo se usa la herramienta de ros *rqt_graph*.

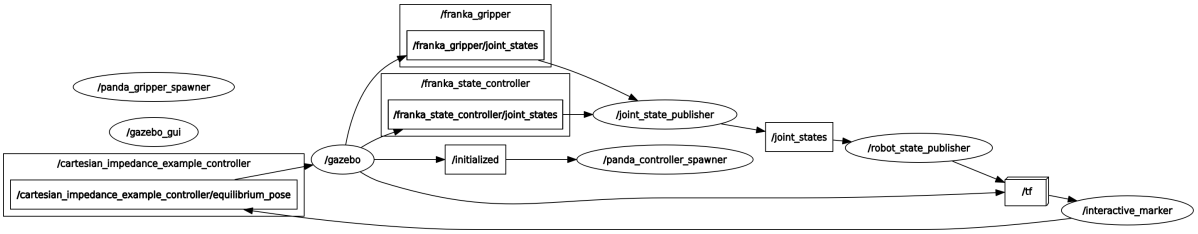


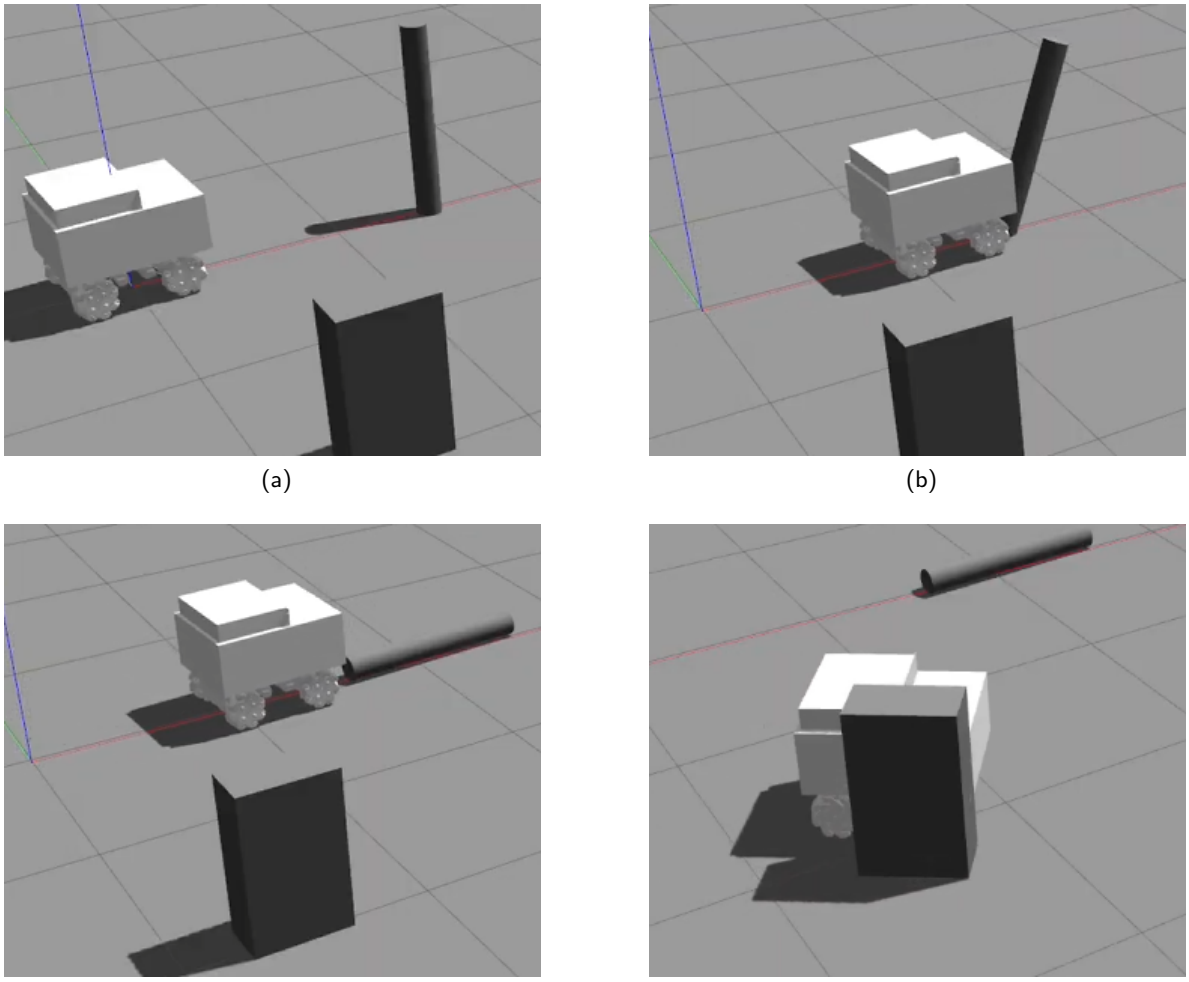
Figura 4.4: Nodos y topics de la simulación del brazo Franka Panda.

4.4. Simulación de la base de RAFI

Para la simulación de la base de RAFI se ha creado un mundo sencillo, con dos elementos, uno en forma cúbica y el otro en forma cilíndrica, con los que probar la correcta colisión del modelo.

Para lanzar la simulación se utiliza el siguiente comando:

```
roslaunch rafi_gazebo display_base.launch \
world:=$(rospack find rafi_gazebo)/worlds/boxes.sdf
```



(a) (b)

(c) (d)

Figura 4.5: Secuencia de simulación de la base móvil de RAFI.

Para mover la base de RAFI existen dos opciones. Se puede, al igual que con la pinza del brazo, escribir directamente en el topic *cmd_vel*, con el comando *rostopic pub*. Este método, aunque correcto, puede resultar poco intuitivo. Por esto se ha optado por usar un paquete externo, llamado *teleop_twist_keyboard*, que proporciona una interfaz que permite generar estos comandos de velocidad con el teclado. Estos comandos pueden ser lineales o angulares según se necesita, pudiendo simular correctamente robots omnidireccionales. El comando para lanzar el programa es el siguiente:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Con esta interfaz existen dos modos de movimiento. En el modo estándar el movimiento que se produce es el de un robot no omnidireccional, lo que permite rotar el robot, mientras que si se pulsa la tecla *shift*, se produce un movimiento omnidireccional. Con la tecla *k* se para al robot y con las

teclas de alrededor se dirige el movimiento. También se puede aumentar o reducir la velocidad a voluntad.

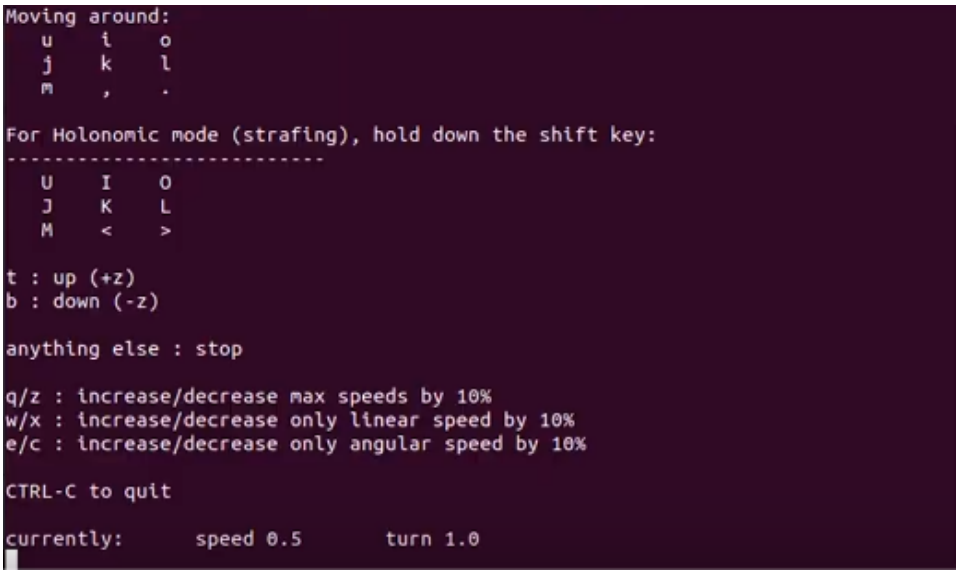


Figura 4.6: Interfaz de teleoperación para la simulación de la base de RAFI.

En la simulación, el objetivo es chocar con el cilindro y el tetraedro para mostrar las físicas de colisión. Como se observa en la figura 4.5, la base de RAFI es capaz de moverse e interactuar con el entorno. Aunque es posible realizar pruebas más complejas, con esta sencilla simulación basta para demostrar el correcto funcionamiento de la base.

Para ello y con la ayuda del programa de teleoperación se envían comandos de velocidad lineales y angulares. Estos comandos se recogen en la figura 4.7. Para esta gráfica se utiliza la herramienta de ROS *rqt_plot*.

En esta gráfica, en el eje x se encuentra el tiempo de simulación en segundos, mientras que en el eje y se encuentra la velocidad en m/s. Es necesario notar que los datos no comienzan en tiempo cero, ya que el inicio de la simulación es anterior a la recogida de datos con *rqt_plot*.

En la figura 4.8 se observan los nodos y topics de la simulación. En este caso hay muchos menos nodos que en la simulación del franka. Esto es debido a que el control no se realiza mediante un controlador dedicado, si no desde el *plugin* dentro del modelo. El *plugin* por lo tanto se encuentra en el nodo /gazebo, que recibe la información desde el topic /cmd_vel, como se ha detallado anteriormente.

4.5. Simulación del modelo de cuerpo completo de RAFI

Para probar el modelo de cuerpo completo de RAFI se ha diseñado un mundo basado en el usado en la simulación del brazo Franka. En este mundo se encuentran dos mesas separadas, en el que la idea es mover la pieza de una mesa a otra, haciendo uso así de las funcionalidades del modelo

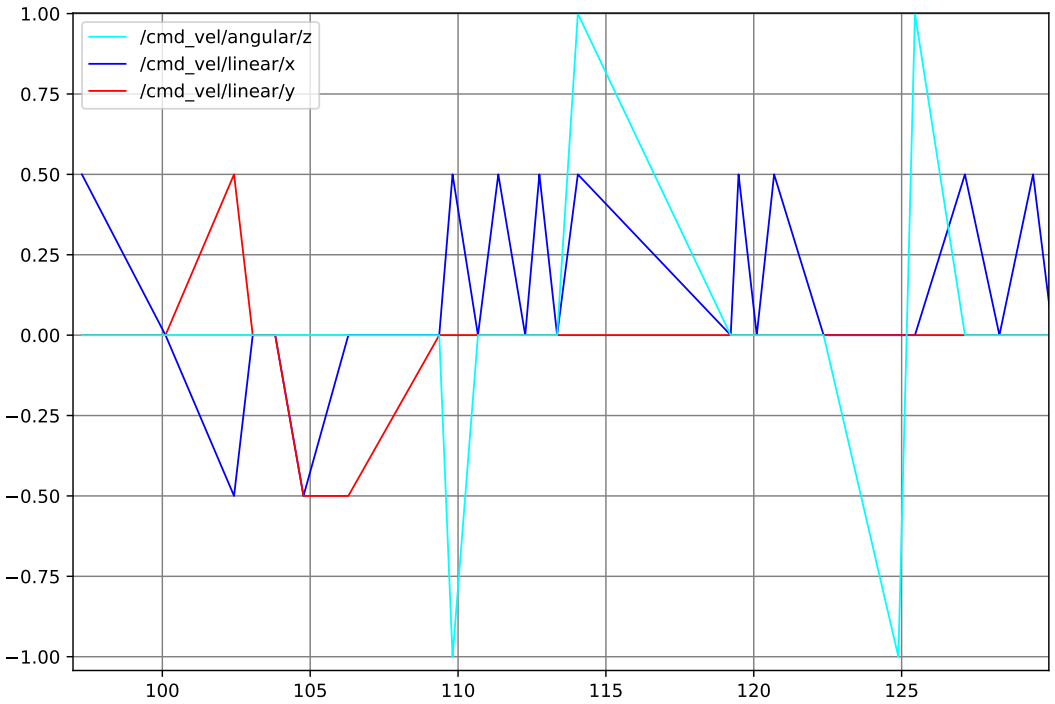


Figura 4.7: Evolución en el tiempo de los comandos de velocidad lineal y angular de la base de RAFI. En el eje X se representa el tiempo de simulación en segundos, mientras que en el eje Y se representa la velocidad lineal, en m/s, y angular, en rad/s.

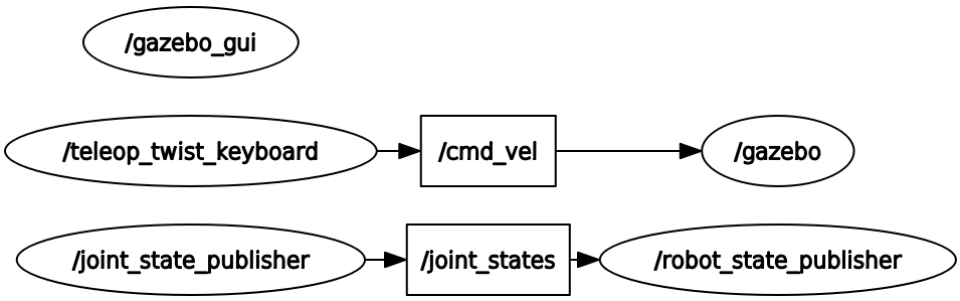


Figura 4.8: Nodos y topics de la simulación de la base de RAFI.

completo, como se observa en la figura 4.9. Además se ha colocado un cilindro similar al de la simulación de la base para probar la colisión del modelo completo. Para lanzar la simulación se utiliza el siguiente comando:

```
roslaunch rafi_gazebo display_v2.launch \
world:=$(rospack find rafi_gazebo)/worlds/stone_v2.sdf
```

Como se ha detallado anteriormente, el control del modelo de cuerpo completo de RAFI se realiza por separado. El brazo Franka Panda se controla mediante los marcardores interactivos en

RViz, mientras que la base se controla con el programa de teleoperación. Así, esta simulación es mecánicamente una mezcla de las dos anteriores.

Es importante, una vez iniciada la simulación, seleccionar un marco de referencia válido en RViz. Este marco de referencia debe ser un *link* del modelo del RAFI. Para esta simulación se ha utilizado el *link Chasis*. Esto implica que la posición del punto de equilibrio estará en función al chasis del RAFI.

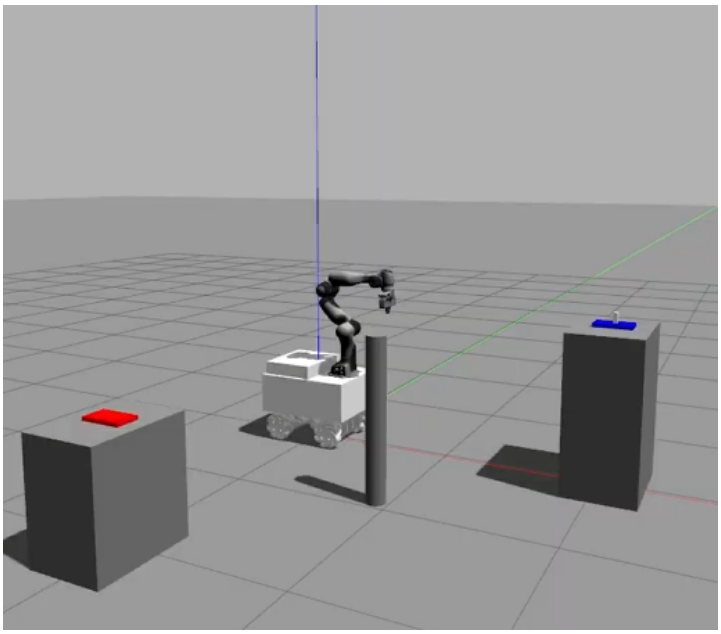


Figura 4.9: Entorno de simulación del modelo de cuerpo completo de RAFI.

Ya en la simulación, movemos a RAFI hacia la primera mesa, para poder coger la piedra. Una vez en posición, se abre la pinza y se agarra la piedra con los mismos comandos que se usan en la simulación del brazo.

Así, con la piedra agarrada por la pinza, se mueve el robot hacia la otra mesa donde dejar la piedra. Para soltar la piedra se manda un mensaje de stop de la misma manera que en la simulación del brazo. Una vez soltada la piedra podría concluir la simulación, aunque se ha probado la colisión del modelo con el cilindro, como se observa en la figura 4.10

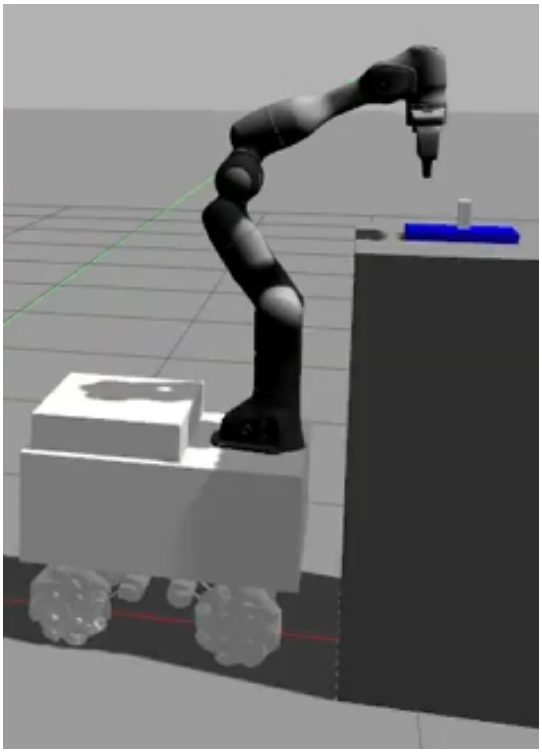
Con esto quedaría probado el funcionamiento del modelo de cuerpo completo de RAFI. Como en las simulaciones anteriores, se ha recogido en gráficas tanto los comandos de velocidad enviados a la base, en la figura 4.11 como la posición del punto de equilibrio del efector final enviado al control de Franka, en la figura 4.12.

En esta simulación no ha sido necesario girar el robot, por lo que no se han enviado comandos de velocidad angular en ningún momento, como se observa en la figura 4.11.

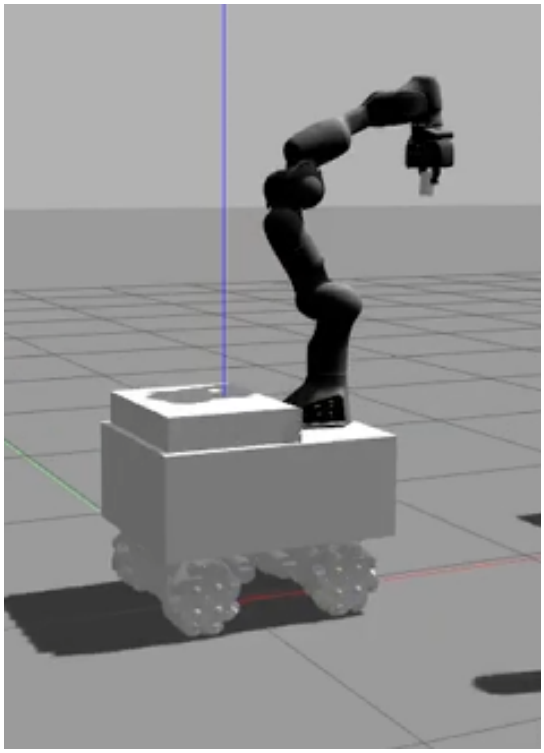
Como el marco de referencia del efector final se sitúa en el chasis, se observa en la figura 4.12 como el punto de equilibrio no se mueve sobre el eje Y, ya que no hace falta para esta prueba.

También se ha realizado el grafo con los nodos y topics activos. Como se puede observar en la figura 4.13, este grafo es una mezcla de las dos simulaciones anteriores.

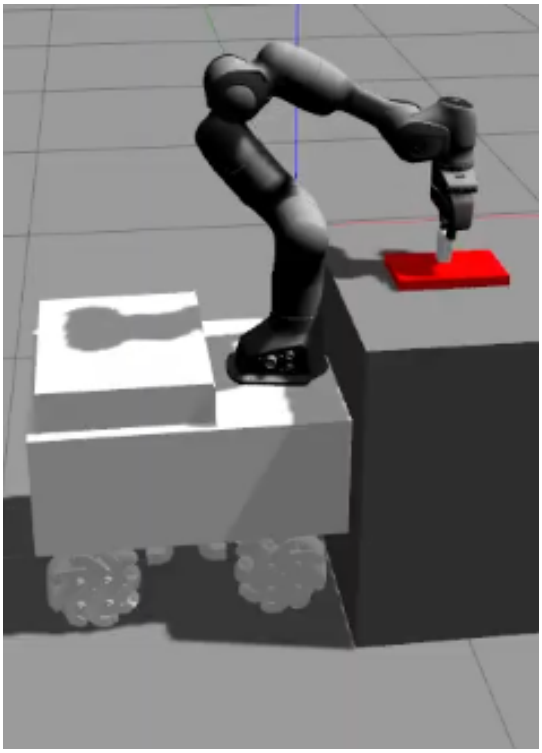
En concreto, se observa como el bucle de control de la simulación es bastante parecido al del franka, con el añadido del control de la base desde el nodo de teleoperación.



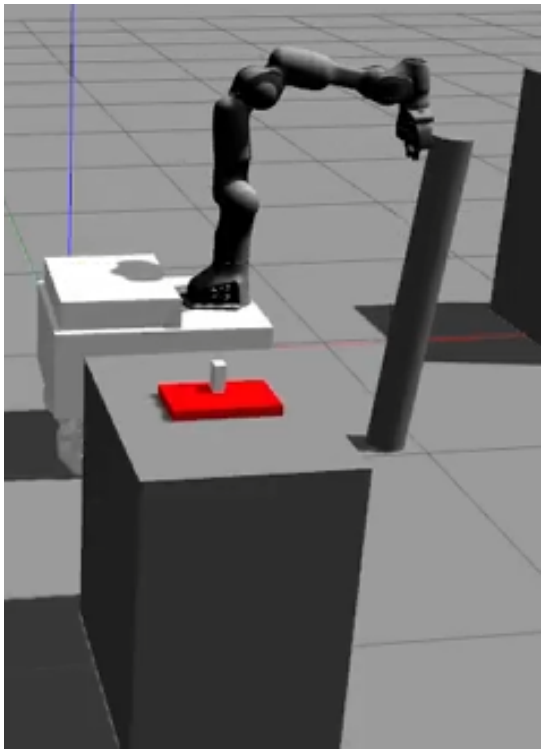
(a)



(b)



(c)



(d)

Figura 4.10: Secuencia de simulación del modelo de cuerpo completo de RAFI.

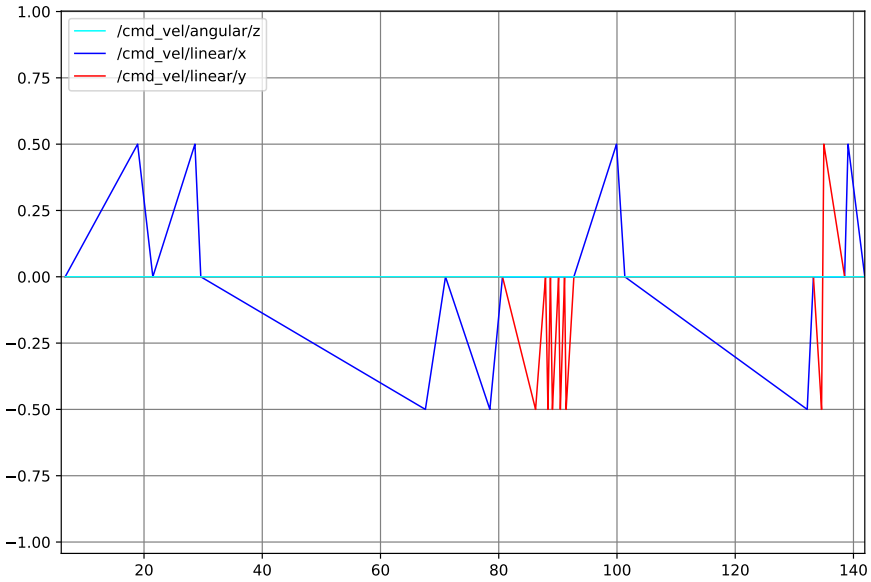


Figura 4.11: Evolución en el tiempo de los comandos de velocidad lineal y angular del modelo completo de RAFI. En el eje X se representa el tiempo de simulación en segundos, mientras que en el eje Y se representa la velocidad lineal, en m/s, y angular, en rad/s.

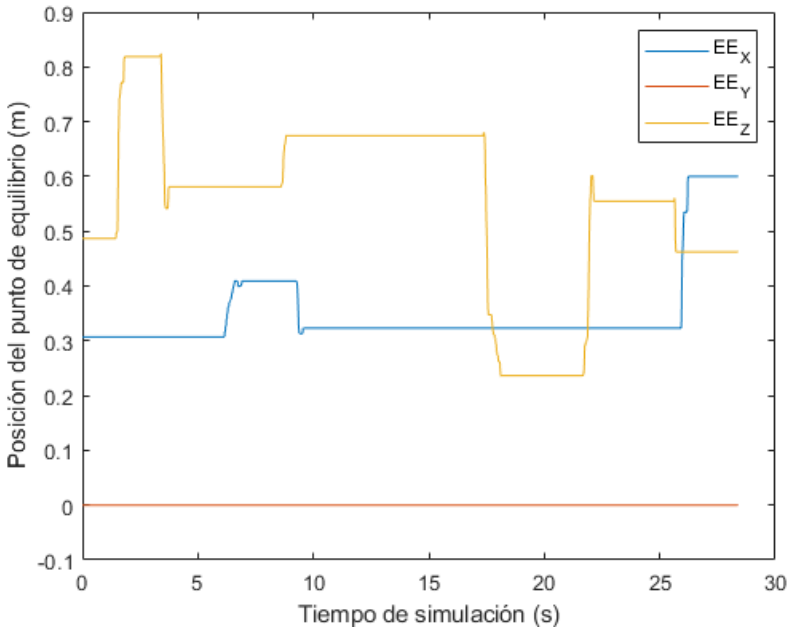


Figura 4.12: Gráfica de posición del punto de equilibrio del brazo Franka Panda en la simulación del modelo de cuerpo completo de RAFI.

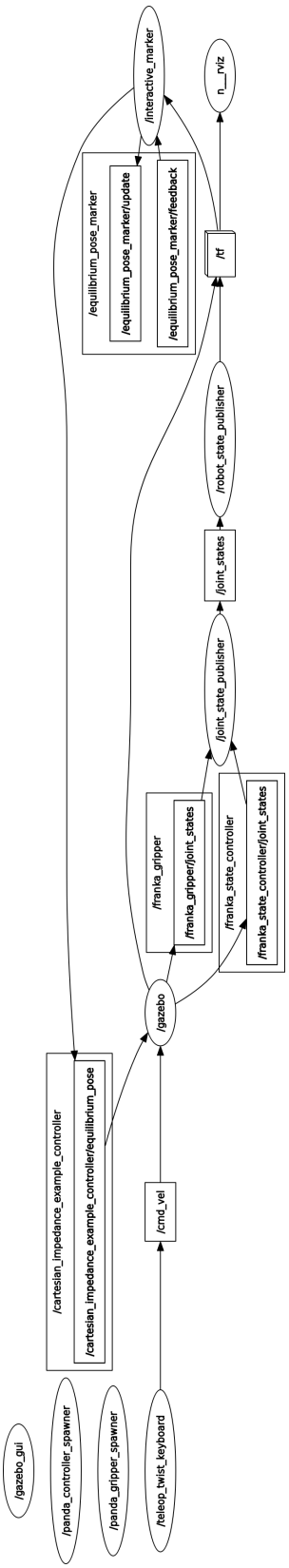


Figura 4.13: Nodos y topics de la simulación del modelo de cuerpo completo de RAFI.

CAPÍTULO 5

Conclusiones

La contribución de este proyecto es la consecución del objetivo de disponer de un modelo de simulación, para lo cual se ha realizado el modelado de cuerpo completo del sistema RAFI. Esta contribución lleva consigo un amplio número de aportaciones secundarias de gran utilidad para los trabajos que se vayan a realizar sorte esta plataforma. Se ha desarrollado el modelo descriptivo correspondiente a la base de RAFI y se ha integrado junto con el brazo robótico Franka Panda para conformar un modelo completo. Asimismo, se ha incorporado un plugin al modelo para dotar de movimiento a la base, lo que facilita el control de la simulación. Además, se ha creado un paquete de ROS específico para la simulación de RAFI y se han elaborado archivos de inicialización que simplifican el inicio de las simulaciones de los distintos modelos. Finalmente, se ha documentado el funcionamiento de los modelos de simulación desarrollados, explicando sus distintos esquemas de control. El código de los distintos archivos desarrollados dentro del paquete de simulación está estructurado de forma homogénea para facilitar el uso para personas no familiarizadas con este proyecto.

En el inicio de el proyecto, se establecieron una serie de objetivos parciales. Ahora se va a analizar el desarrollo de los mismos.

El primer objetivo era analizar la estructura del modelo de simulación del brazo Franka Panda, que se ha realizado en esta memoria. Para ello se ha recurrido a la documentación propia de Franka Emika sobre el brazo existente en la red, por lo que no ha resultado complicado conseguir este objetivo. En esta memoria se recoge este análisis, en el que se ha hecho especial énfasis en los distintos controladores del brazo. Además, el segundo objetivo era simular este brazo robótico, lo que ha facilitado este análisis. La simulación del modelo del brazo resulta también sencilla, gracias a que en la misma documentación existe un pequeño tutorial de inicialización de la simulación.

El tercer objetivo era la adaptación de los modelos tridimensionales existentes de RAFI. Estos

modelos estaban en el formato estándar de SolidWorks, que no es compatible con Gazebo. Para ello se usó el propio SolidWorks para exportarlos a un formato compatible. Además, estos modelos eran muy detallados, lo que hacía que la simulación se ralentizara. Para solventar esto se usó Blender para reducir los polígonos de los modelos, con el objetivo de agilizar la simulación.

El cuarto objetivo era la creación del paquete de ROS para el modelo de simulación. Gracias a las distintas herramientas que ofrece ROS no ha resultado un proceso complicado. Además, este objetivo comprendía añadir el paquete al repositorio de Github del proyecto. Una vez más, existen una gran variedad de herramientas distintas para este cometido, lo que facilita este proceso.

El quinto objetivo era el desarrollo del modelo descriptivo de la base de RAFI. Para ello se ha creado el archivo URDF que define la base móvil. Este proceso requería la medición del robot real, a fin de mantener unas proporciones realistas. Se han aprovechado los modelos tridimensionales adaptados para dotar al modelo de mayor realismo. El URDF se complementa con el octavo objetivo, ya que la implementación del *plugin* se realiza en el mismo URDF. El principal problema surge al elegir el *plugin* que usar. Para robots omnidireccionales existe el *plugin* de Gazebo llamado Planar Move, pero se descartó debido al pobre rendimiento en simulación. Al final se optó por el *plugin* Force Based Move desarrollado por la Universidad de Dresden, que permitía simular el movimiento omnidireccional. El *plugin* utilizado sirve como controlador de la base, lo que permite simplificar el desarrollo del modelo de simulación.

El sexto objetivo era la integración en un único modelo de descripción tanto la base móvil como el brazo Franka Panda, consiguiendo un modelo de simulación de cuerpo completo de RAFI. Este proceso es sencillo con la implementación de XACRO, lo que permite usar macros en los archivos URDF. Con esto simplemente se modificó el modelo descriptivo de la base, lo que permitió escribir el modelo descriptivo de cuerpo completo como dos macros. Una de estas macros define la base y la otra el brazo robótico.

El séptimo objetivo era la creación de archivos de inicialización. Estos archivos permiten una inicialización rápida y sencilla con un solo comando. Así, se establecieron los nodos necesarios para la correcta simulación en estos archivos. El problema principal de este proceso es identificar los nodos necesarios, pero no resultó un trabajo difícil.

El último objetivo era realizar las simulaciones de prueba para la validación de los modelos. Las simulaciones se han recogido en esta memoria. Estas simulaciones son el resultado del correcto desarrollo de los objetivos anteriores. Para estas simulaciones se han creado entornos personalizados con el fin de probar las distintas funcionalidades de los modelos. Los resultados de las simulaciones, recogidos en el capitulo 4, son satisfactorios.

De esta manera, se puede afirmar que el modelo de simulación cumple con los objetivos establecidos al inicio del proyecto. No obstante, el modelo de simulación aún presenta áreas de mejora que podrían abordarse en futuros proyectos.

Gracias a la base de ROS de este proyecto es posible realizar modificaciones al modelo de manera sencilla y eficiente. De este modo, en el futuro se podrían incorporar nuevas funcionalidades al

modelo, aprovechando las funciones de ROS. En este contexto, resultaría relevante la incorporación de sensores al modelo, lo cual permitiría programar la detección de obstáculos. Además, también sería beneficioso implementar el mapeo del entorno utilizando estos mismos sensores, con el objetivo de lograr un movimiento autónomo del robot.

Ahora bien, antes de añadir nuevas funcionalidades al modelo sería interesante optimizar el esquema de control del modelo completo. En la versión actual del modelo, el control de RAFI se gestiona de manera separada, empleando dos esquemas de control distintos. Aunque esta aproximación es funcional, se desvía de la realidad y complica el control de la simulación. Por ello, se sugiere implementar un esquema de control unificado que permita el control simultáneo tanto del brazo como del chasis, además de el control unificado del robot real y el simulado.

Asimismo, el control por simulación de RAFI permite la exploración de nuevas soluciones y algoritmos mediante el uso de la propia simulación, empleando una técnica conocida como *sim2real*. Esta metodología ofrece la posibilidad de desarrollar y probar diversos proyectos simultáneamente sobre RAFI, sin la necesidad de validar el funcionamiento directamente en el robot físico.

Finalmente, se plantea la inclusión del modelo cinemático de la base de RAFI en la simulación. Esta mejora permitiría analizar las relaciones cinemáticas entre las velocidades angulares de las ruedas y las velocidades lineales y angulares de la base, proporcionando una representación más precisa del comportamiento del sistema.

Bibliografía

[1] F. de Paula Carbonero, "Funciones básicas de navegación ros para un manipulador móvil omnidireccional," 2023.

[2] H. Choi, C. Crump, C. Duriez, A. Elmquist, G. Hager, D. Han, F. Hearl, J. Hodgins, A. Jain, F. Leve *et al.*, "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *Proceedings of the National Academy of Sciences*, vol. 118, no. 1, p. e1907856118, 2021.

[3] Y. You, Z. Fan, W. Chen, G. Zhu, B. Qiu, J. Xin, J. Chen, F. Deng, Y. Hou, W. Liang *et al.*, "Design and implementation of mobile manipulator system," in *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE, 2019, pp. 113–118.

[4] O. Madsen, S. Bøgh, C. Schou, R. S. Andersen, J. S. Damgaard, M. R. Pedersen, and V. Krüger, "Integration of mobile manipulators in an industrial production," *Industrial Robot: An International Journal*, vol. 42, no. 1, pp. 11–18, 2015.

[5] Z. Li, P. Moran, Q. Dong, R. J. Shaw, and K. Hauser, "Development of a tele-nursing mobile manipulator for remote care-giving in quarantine areas," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3581–3586.

[6] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, "Design, integration, and field evaluation of a robotic apple harvester," *Journal of Field Robotics*, vol. 34, no. 6, pp. 1140–1159, 2017.

[7] L. Žlajpah, "Simulation in robotics," *Mathematics and Computers in Simulation*, vol. 79, no. 4, pp. 879–897, 2008.

[8] C. Pepper, S. Balakirsky, and C. Scrapper, "Robot simulation physics validation," in *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, 2007, pp. 97–104.

[9] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[10] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *2011 ieee international conference on robotics and automation*. IEEE, 2011, pp. 46–51.

[11] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 1321–1326.

[12] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. Ieee, 2004, pp. 2149–2154.

[13] A. Farley, J. Wang, and J. A. Marshall, "How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion," *Simulation Modelling Practice and Theory*, vol. 120, p. 102629, 2022.

[14] M. Ivanou, S. Mikhel, and S. Savin, "Robot description formats and approaches," in *2021 International Conference"Nonlinearity, Information and Robotics"(NIR)*. IEEE, 2021, pp. 1–5.

[15] «Robotino». Accedido: 11 de septiembre de 2024. [Online]. Available: http://wiki.ros.org/Robots/Robotino

[16] «Summit-XL-STEEL». Accedido: 11 de septiembre de 2024. [Online]. Available: https://robots.ros.org/summit-xl-steel/

[17] «RB-Kairos». Accedido: 11 de septiembre de 2024. [Online]. Available: https://robots.ros.org/rb-kairos/

[18] A. Serrano Flores, "Control de una plataforma omnidireccional para un manipulador móvil," 2021.

[19] R. Castro Ochoa, "Desarrollo de una interfaz software para el control de un manipulador móvil con ruedas omnidireccionales," 2024.

[20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[21] E. Mingo Hoffman, S. Traversaro, A. Rocchi, M. Ferrati, A. Settimi, F. Romano, L. Natale, A. Bicchi, F. Nori, and N. G. Tsagarakis, "Yarp based plugins for gazebo simulator," in *Modelling and Simulation for Autonomous Systems: First International Workshop, MESAS 2014, Rome, Italy, May 5-6, 2014, Revised Selected Papers 1*. Springer, 2014, pp. 333–346.

[22] «Move it! tutorials». Accedido: 9 de septiembre de 2024. [Online]. Available: https://moveit.github.io/moveit_tutorials/

[23] S. Pietrzik and B. Chandrasekaran, "Setting up and using ros-kinetic and gazebo for educational robotic projects and learning," in *Journal of Physics: Conference Series*, vol. 1207, no. 1. IOP Publishing, 2019, p. 012019.