



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Integración de Robot Manipulador en un Sistema Distribuido con ROS

Autor

Juan Antonio Martínez Sánchez

Director

Francisco Barranco Expósito



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

—
Granada, Noviembre de 2024

Integración de Robot Manipulador en un Sistema Distribuido con ROS

Autor

Juan Antonio Martínez Sánchez

Director

Francisco Barranco Expósito



DEPARTAMENTO DE INGENIERÍA DE COMPUTADORES, AUTOMÁTICA Y ROBÓTICA

—
Granada, Noviembre de 2024

Integración de Robot Manipulador en un Sistema Distribuido con ROS

Juan Antonio Martínez Sánchez

Palabras clave: ROS (Robot Operating System), Manipulador robótico, Simulación en Gazebo, Control remoto, Planificación de movimientos, Robótica colaborativa, Sistema distribuido, Automatización, Validación experimental, Entorno de manipulación remota.

Resumen

Este trabajo se centra en la integración de un robot manipulador con el sistema de operación robótica Robot Operating System (ROS), diseñado para permitir su operación en un entorno de manipulación remota. La robótica colaborativa y la manipulación a distancia son áreas clave en el desarrollo de tecnologías avanzadas, y este proyecto busca contribuir a este campo mediante la creación de un sistema robusto y accesible para el control remoto de robots manipuladores.

El objetivo principal es diseñar, simular y validar experimentalmente un sistema de control que permita manipular el robot de forma remota a través de ROS, garantizando precisión, seguridad y adaptabilidad en el manejo de tareas complejas. Para lograrlo, se utilizará un simulador 3D avanzado (como Gazebo) en la etapa inicial, lo cual permitirá modelar el entorno y el comportamiento del robot, así como probar los algoritmos de planificación de movimiento y control en condiciones seguras. La simulación ofrecerá un entorno controlado donde se pueden anticipar posibles desafíos técnicos, probar distintos enfoques de control y optimizar la planificación de movimientos sin riesgos para el hardware.

Una vez que el sistema haya sido probado y optimizado en simulación, se llevará a cabo la implementación y validación en un entorno real utilizando el robot manipulador físico. En esta fase, se integrarán las configuraciones y algoritmos validados previamente, adaptándolos al hardware y evaluando su efectividad en condiciones prácticas. El sistema permitirá la ejecución de comandos de manera remota y en tiempo real, lo cual resulta de especial interés para aplicaciones en entornos de difícil acceso o que requieren un control preciso sin la intervención directa del operador.

Este proyecto tiene el potencial de aportar soluciones innovadoras para la manipulación remota, contribuyendo al desarrollo de entornos automatizados y colaborativos donde los robots puedan operar de manera efectiva en diversas aplicaciones industriales, científicas y médicas. Al emplear ROS, se asegura la flexibilidad y escalabilidad del sistema, permitiendo futuras expansiones y adaptaciones para distintos tipos de robots y aplicaciones.

Integration of a Manipulator Robot in a Distributed System with ROS

Juan Antonio, Martínez Sánchez

Keywords: Robot Operating System (ROS), Manipulator robot, Remote operation, Collaborative robotics, Remote manipulation, 3D simulation, Gazebo, Motion planning, Control system, Real-time command execution, Automation, Scalability, Industrial applications, Scientific applications, Medical applications

Abstract

This project focuses on the integration of a manipulator robot with the Robot Operating System (ROS) to enable remote operation in a robotic manipulation environment. Collaborative robotics and remote manipulation are key areas in advanced technology development, and this project aims to contribute to this field by creating a robust and accessible system for remote control of manipulator robots.

The primary objective is to design, simulate, and experimentally validate a control system that allows for the remote manipulation of the robot through ROS, ensuring precision, safety, and adaptability for complex tasks. To achieve this, a 3D simulator (such as Gazebo) will be used in the initial stages to model the robot's environment and behavior, as well as to test motion planning and control algorithms in a safe environment. Simulation provides a controlled setting in which technical challenges can be anticipated, different control approaches can be tested, and motion planning can be optimized without risk to hardware.

Once the system has been tested and optimized in simulation, implementation and validation will take place in a real-world setting using the physical manipulator robot. In this phase, the previously validated configurations and algorithms will be integrated, adapted to the hardware, and evaluated for effectiveness under practical conditions. The system will enable remote, real-time command execution, which is particularly valuable for applications in hard-to-access environments or those requiring precise control without direct operator intervention.

This project has the potential to provide innovative solutions for remote manipulation, contributing to the development of automated and collaborative environments where robots can effectively operate across various industrial, scientific, and medical applications. By using ROS, the system ensures flexibility and scalability, allowing for future expansions and adaptations to different types of robots and applications.

Yo, **Juan Antonio Martínez Sánchez**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 7776796H, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Antonio Martínez Sánchez

Granada a 12 de Noviembre de 2024.

D. **Francisco Barranco Expósito**, Profesor del Departamento de Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Integración de Robot Manipulador en un Sistema Distribuido con ROS*, ha sido realizado bajo su supervisión por **Juan Antonio Martínez Sánchez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 12 de Noviembre de 2024.

El director:

Francisco Barranco Expósito

Agradecimientos

Quiero expresar mi más sincera gratitud a todas las personas que me han acompañado durante el desarrollo de este trabajo y a lo largo de esta importante etapa de mi vida.

En primer lugar, quiero agradecer a mi tutor, Francisco Barranco Expósito, por su constante apoyo, dedicación, paciencia, y por su amabilidad y disponibilidad a lo largo de todo el proceso.

A mi mentor Elio Valenzuela Segura, quien me apoyó a lo largo del proyecto, especialmente en las pruebas de laboratorio, cuya ayuda fue fundamental para superar los desafíos prácticos que se presentaron.

No quiero olvidarme de los amigos que he hecho durante esta etapa, cuya compañía ha hecho este proceso mucho más llevadero y lleno de momentos inolvidables. Gracias por estar siempre ahí.

Agradezco profundamente al personal docente de la ETSIIT por ser una parte esencial en mi formación, por el conocimiento invaluable que me han transmitido, y por demostrar siempre su profesionalidad y dedicación.

Por último, quiero expresar mi más sincero agradecimiento a mi familia por su apoyo incondicional y por confiar en mí desde el primer día. Sin vuestra comprensión, paciencia y amor, nada de esto habría sido posible.

Índice general

| | |
|--|-----------|
| 1. Introducción | 1 |
| 1.1. Descripción del Problema | 1 |
| 1.2. Relevancia y Contribución del Proyecto | 3 |
| 1.3. Motivación | 3 |
| 1.4. Objetivos | 4 |
| 1.5. Planificación | 4 |
| 2. Estado del Arte | 9 |
| 2.1. Introducción al Estado del Arte | 9 |
| 2.2. Contexto y Problema General | 10 |
| 2.3. Control de Brazos Robóticos Industriales | 12 |
| 2.3.1. Métodos de Control | 12 |
| 2.3.2. Cinemática Inversa (IK) | 13 |
| 2.3.3. Importancia de la Cinemática Inversa en el Control Robótico | 13 |
| 2.4. Stack de Software y Hardware | 14 |
| 2.4.1. Computación Distribuida | 14 |
| 2.4.2. Alternativas en Computación Distribuida | 14 |
| 2.4.3. Computación Distribuida Seleccionada | 15 |
| 2.4.4. Simulación | 20 |
| 2.4.5. Alternativas en Simulación | 20 |
| 2.4.6. Simulador Seleccionado | 22 |
| 2.4.7. Planificadores | 24 |
| 2.4.8. Alternativas en Planificación de Movimientos | 24 |
| 2.4.9. Planificador Seleccionado | 25 |
| 2.4.10. Brazo Robótico Staubli TX2-90 | 27 |
| 3. Metodología de Trabajo | 31 |
| 3.1. Configuración de ROS | 31 |
| 3.2. Configuración y Ejecución de la Simulación en Gazebo | 34 |
| 3.3. Configuración de MoveIt y Ejecución en Gazebo | 35 |
| 4. Implementación | 41 |
| 4.1. Soluciones Software Desarrolladas | 41 |
| 4.1.1. Preparación del Espacio de Trabajo | 41 |
| 4.1.2. Solución para Simulación | 41 |
| 4.1.3. Solución para el Caso Real | 43 |
| 4.2. Arquitectura de la Solución | 45 |
| 4.2.1. Estructura del software | 45 |
| 4.2.2. Estructura de Nodos y Tópicos en la Simulación de ROS | 47 |
| 4.2.3. Servicios Disponibles en la Simulación | 49 |

| | |
|--|-----------|
| 5. Pruebas y Validaciones | 51 |
| 5.1. Prueba en Simulación | 51 |
| 5.2. Prueba con Brazo Robótico Físico en un Entorno Realista | 58 |
| 5.3. Prueba de Precisión | 61 |
| 5.4. Comparación de Algoritmos | 67 |
| 5.4.1. Cálculo de la Longitud de la Trayectoria | 67 |
| 5.4.2. RRTConnect | 69 |
| 5.4.3. RRTStar | 70 |
| 5.4.4. LazyPRM | 70 |
| 5.4.5. BKPIECE | 71 |
| 5.4.6. SPARS | 72 |
| 5.4.7. Análisis y Comparativa de Resultados | 73 |
| 6. Conclusiones | 76 |
| 6.1. Revisión de Objetivos | 76 |
| 6.2. Documentación de la Solución en Repositorio Git | 77 |
| 6.3. Trabajos Futuros | 77 |
| 6.4. Conclusión Final | 78 |
| Bibliografía | 81 |
| A. Anexo de Imágenes | 82 |

Índice de figuras

| | | |
|-------|--|----|
| 1.1. | Logo del proyecto IFMIF-DONES | 1 |
| 1.2. | Esquema simplificado del funcionamiento de la instalación IFMIF-DONES [1] | 2 |
| 1.3. | Instalación IFMIF-DONES | 2 |
| 2.1. | Robot utilizado para limpiar escombros en la planta nuclear de Fukushima tras el desastre nuclear de 2011. [2] | 10 |
| 2.2. | Brazos robóticos en una línea de producción automotriz, utilizados para ensamblar piezas con precisión y eficiencia en procesos automatizados. | 11 |
| 2.3. | Sistema robótico da Vinci | 11 |
| 2.4. | ERA en acción en la Estación Espacial Internacional. [3] | 12 |
| 2.5. | Parámetros del brazo robótico para la cinemática inversa | 13 |
| 2.6. | Modelo del Staubli TX-90 en RViz | 19 |
| 2.7. | Gazebo | 22 |
| 2.8. | Brazo Robótico Staubli TX2-90. | 27 |
| 3.1. | Matriz de Colisiones en el Setup Assistant de MoveIt. | 36 |
| 3.2. | Definición de Planning Group en el Setup Assistant de MoveIt. | 37 |
| 3.3. | Configuración de Controladores en el Setup Assistant de MoveIt. | 38 |
| 3.4. | Simulación del robot en Gazebo. | 38 |
| 3.5. | Interfaz de MoveIt en RViz para la planificación de movimientos. | 39 |
| 4.1. | Estructura de carpetas del proyecto | 46 |
| 4.2. | Estructura de nodos en la simulación de ROS | 47 |
| 4.3. | Estructura de nodos y tópicos en la simulación de ROS | 48 |
| 5.1. | Posición inicial del brazo robótico. | 57 |
| 5.2. | Posición final del brazo robótico y mensaje de éxito en la terminal. | 57 |
| 5.3. | Teach pendant SP2 del controlador CS9 mostrando el estado de conexión y ejecución del robot Staubli con ROS. | 59 |
| 5.4. | Estructura de nodos y tópicos en la prueba en entorno real de ROS | 60 |
| 5.5. | Posición del brazo robótico Staubli TX2-90 y el entorno de la prueba | 61 |
| 5.6. | Efector final utilizado. | 62 |
| 5.7. | Resultado Test 1 | 63 |
| 5.8. | Resultado Test 2 | 64 |
| 5.9. | Estado final de la ejecución de la prueba en la terminal y en RViz | 65 |
| 5.10. | | 66 |
| 5.11. | | 66 |
| 5.12. | Resultado final y comprobación de medidas del cuadrado. | 66 |
| A.1. | Datasheet del brazo robótico Staubli TX2-90 (parte 1). | 83 |
| A.2. | Datasheet del brazo robótico Staubli TX2-90 (parte 2). | 84 |

Índice de cuadros

| | |
|---|----|
| 1.1. Planificación del proyecto. | 5 |
| 1.2. Resumen de costos del proyecto. | 7 |
| 5.1. Resultados de RRTConnect | 69 |
| 5.2. Resultados de RRTStar | 70 |
| 5.3. Resultados de LazyPRM | 71 |
| 5.4. Resultados de BKPIECE | 72 |
| 5.5. Resultados de SPARS | 73 |
| 5.6. Comparativa de Resultados Promedio de los Algoritmos | 73 |

Introducción

1.1. Descripción del Problema

Este proyecto se enmarca dentro del contexto del proyecto IFMIF-DONES (*International Fusion Materials Irradiation Facility - DEMO Oriented Neutron Source*), una iniciativa de investigación internacional que tiene como objetivo desarrollar una fuente avanzada de neutrones orientada a la investigación de materiales que puedan soportar las condiciones extremas de los reactores de fusión nuclear de próxima generación. IFMIF-DONES es una infraestructura experimental clave para la comunidad científica, ya que permitirá estudiar el comportamiento de materiales en condiciones de radiación intensa y temperaturas elevadas, similares a las que se esperan en futuros reactores de fusión como ITER y DEMO [4].



Figura 1.1: Logo del proyecto IFMIF-DONES

La misión de IFMIF-DONES es proporcionar un flujo de neutrones de alta intensidad, lo cual es fundamental para analizar la resistencia, durabilidad y fiabilidad de materiales avanzados. Entre los materiales de interés se encuentran metales, aleaciones y compuestos especialmente diseñados para soportar entornos nucleares hostiles, donde la radiación y el calor presentan desafíos significativos para la estabilidad estructural y funcional de estos materiales [5].

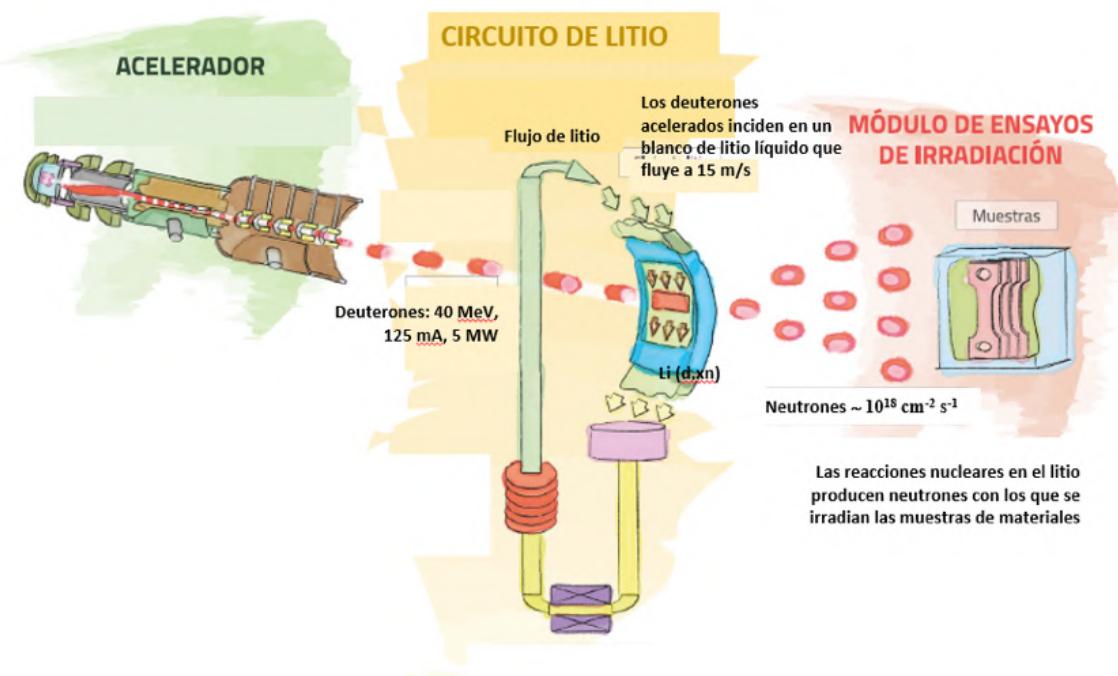


Figura 1.2: Esquema simplificado del funcionamiento de la instalación IFMIF-DONES [1]

Uno de los principales retos en este ámbito de investigación es la manipulación y control de materiales en entornos donde las condiciones de radiación y temperatura imposibilitan la intervención humana directa. Para enfrentar esta dificultad, la automatización y el uso de sistemas robóticos avanzados son esenciales, ya que permiten el manejo y operación de materiales de forma remota y segura. Este proyecto, por tanto, se centra en el desarrollo de sistemas robóticos capaces de operar en condiciones adversas, contribuyendo a la investigación y desarrollo de materiales en el marco de IFMIF-DONES.



Figura 1.3: Instalación IFMIF-DONES

1.2. Relevancia y Contribución del Proyecto

La robótica industrial es fundamental para permitir la manipulación segura de componentes en entornos de radiación, como los que se encuentran en el marco del proyecto IFMIF-DONES. Este proyecto contribuye a IFMIF-DONES mediante el desarrollo y control de sistemas robóticos que pueden ser empleados en la manipulación y control de materiales en condiciones adversas. Para simular y preparar estos sistemas en un entorno seguro, se emplea una plataforma de simulación y un entorno de desarrollo especializado en robótica, ampliamente utilizados en la investigación y desarrollo de tecnologías avanzadas.

Un brazo robótico ha sido seleccionado para este proyecto debido a su precisión y robustez, características indispensables en entornos industriales complejos y de alta exigencia. La estrategia de trabajo incluye una fase de simulación detallada en un entorno virtual y una fase de pruebas en laboratorio, lo cual permite una transición controlada entre el entorno simulado y el entorno real. Esta aproximación es esencial para garantizar tanto la seguridad como la funcionalidad del sistema robótico en las condiciones específicas del proyecto IFMIF-DONES.

Este proyecto busca no solo contribuir al desarrollo de sistemas de manipulación robótica avanzada, sino también establecer un marco de trabajo para la simulación y el control de robots en entornos de alta radiación. Con esto se pretende aportar soluciones de automatización en el campo de la energía nuclear, donde la manipulación precisa y la seguridad son factores críticos para el éxito de la investigación.

1.3. Motivación

El desarrollo de fuentes de energía sostenibles y de alta capacidad es uno de los mayores desafíos del siglo XXI. La fusión nuclear es una de las opciones más prometedoras para satisfacer la demanda energética global de forma limpia y segura, ya que, a diferencia de la fisión nuclear, produce menos residuos radiactivos y no utiliza materiales de alto riesgo. Sin embargo, la tecnología de fusión plantea retos significativos en cuanto a materiales y sistemas que puedan soportar las condiciones extremas de un reactor de fusión.

En este contexto, IFMIF-DONES representa un paso crucial hacia la creación de infraestructuras de investigación que permitan avanzar en el desarrollo de tecnologías para la energía de fusión. La automatización avanzada y la robótica desempeñan un papel esencial en esta investigación, ya que permiten la manipulación remota de materiales y componentes bajo condiciones que imposibilitan la intervención humana directa debido a la radiación y las altas temperaturas.

La motivación principal de este proyecto es contribuir a la investigación en automatización robótica en el ámbito nuclear. La capacidad de simular y controlar con precisión sistemas robóticos y de implementar un sistema de planificación de movimientos eficiente es fundamental para optimizar las operaciones y asegurar la seguridad en proyectos de alta tecnología como IFMIF-DONES. Este proyecto proporciona una plataforma de investigación para el desarrollo de soluciones avanzadas en robótica, esenciales para el progreso de la ciencia y la tecnología de fusión nuclear.

1.4. Objetivos

Objetivo General

Desarrollar y validar un sistema de control robótico capaz de operar de manera segura y eficiente en entornos simulados y reales, adecuado para la manipulación de materiales en entornos de alta radiación en el contexto del proyecto IFMIF-DONES.

Objetivos Específicos

- **OE1: Establecer la comunicación con el brazo robótico mediante ROS:** Configurar el sistema de comunicación mediante ROS para enviar comandos al brazo robótico, permitiendo su control remoto y la interacción con otros sistemas en el entorno de trabajo.
- **OE2: Integrar y controlar un brazo robótico en un entorno simulado:** Configurar y ejecutar la simulación del brazo robótico en un entorno virtual, permitiendo observar y analizar sus movimientos antes de su implementación física.
- **OE3: Implementar y optimizar la planificación de movimientos:** Utilizar un sistema de planificación para diseñar trayectorias de movimiento seguras y eficientes, asegurando que el brazo pueda realizar tareas específicas evitando colisiones.
- **OE4: Validar el sistema con el brazo robótico físico en un entorno realista:** Realizar pruebas en condiciones realistas con el brazo robótico físico, comparando los resultados obtenidos en la simulación con los del entorno físico para asegurar la precisión y funcionalidad en entornos industriales.
- **OE5: Liberar el código en un repositorio de GitHub:** Publicar el código desarrollado en un repositorio de GitHub, documentándolo adecuadamente. Esta práctica, poco común en el campo de la robótica, es crucial para facilitar la colaboración con la comunidad científica y permitir que otros investigadores y desarrolladores lo utilicen y mejoren.

1.5. Planificación

La planificación de este proyecto se desarrolló utilizando una metodología en cascada [6], con un enfoque lineal en el que cada fase dependía del éxito de la anterior. Este enfoque resultó adecuado debido a la naturaleza secuencial del proyecto, donde cada paso requería una base establecida en la etapa anterior para poder avanzar. El tiempo total estimado fue de 300 horas, correspondientes a 12 créditos ECTS. Sin embargo, este cálculo es una aproximación, ya que la disponibilidad de tiempo varió a lo largo del proyecto debido a la coincidencia con vacaciones y a la dificultad de compaginar con mis responsabilidades laborales. A continuación, se detalla cada fase de trabajo, junto con los desafíos que surgieron durante el desarrollo.

| Fase | Duración | May | Jun | Jul | Ago | Sep | Oct | Nov |
|---|-----------|-----|-----|-----|-----|-----|-----|-----|
| Configuración del Entorno e Integración de Simulación | 2 meses | | | | | | | |
| Investigación de Planificadores | 1 mes | | | | | | | |
| Configuración e Integración de MoveIt! | 2 meses | | | | | | | |
| Establecimiento de Comunicación y Pruebas de Validación en Simulación | 1 mes | | | | | | | |
| Pruebas y Validación en el Laboratorio | 1 mes | | | | | | | |
| Documentación | 2 semanas | | | | | | | |

Cuadro 1.1: Planificación del proyecto.

Detalle de las Fases

1. Configuración del Entorno de Trabajo e Integración de la Simulación (Mayo-Junio)

La primera fase del proyecto abarcó la configuración del entorno de trabajo en ROS y la integración del simulador Gazebo con ROS-Industrial. Esta etapa comenzó en mayo con la introducción a ROS, donde aprendí sobre su estructura, funcionamiento y componentes principales. A continuación, integré ROS-Industrial, lo que permitió utilizar modelos de robots industriales, como el brazo Staubli TX-90, en simulación. Durante junio, trabajé en la implementación de estos modelos en Gazebo, configurando los parámetros necesarios para realizar pruebas virtuales en un entorno seguro. A pesar de algunos problemas de compatibilidad y ajustes específicos en los paquetes, logré establecer un entorno de simulación funcional que permitió el avance hacia las siguientes fases.

2. Investigación de Planificadores de Movimiento (Julio)

En julio, dediqué tiempo a la investigación de diferentes planificadores de movimiento disponibles en ROS, explorando alternativas y evaluando sus ventajas y desventajas. Esta fase fue crucial para seleccionar un planificador adecuado, ya que las características de cada uno varían en cuanto a precisión, capacidad de detección de colisiones y eficiencia en el cálculo de trayectorias. Sin embargo, esta etapa fue desafiante, ya que la documentación de algunos planificadores era limitada y fue necesario revisar múltiples fuentes de información para comprender las opciones disponibles. No se realizaron pruebas en esta fase, ya que se centró únicamente en el análisis teórico de las alternativas.

3. Configuración e Integración de MoveIt! (Agosto-Septiembre)

Una vez elegido el planificador adecuado, la siguiente fase consistió en configurar e integrar los paquetes de MoveIt!, un sistema de planificación de movimiento para ROS. Esta fase implicó ajustar los parámetros específicos del brazo robótico Staubli y adaptar el entorno de trabajo en MoveIt! para optimizar la planificación de trayectorias y asegurar la detección de colisiones. Esta fase presentó varios retos técnicos, especialmente en lo referente a la configuración de los paquetes de MoveIt! para el brazo Staubli, ya que hubo problemas de configuración que requerían ajustes continuos. A pesar de estos obstáculos, logré completar esta etapa y obtener una configuración funcional que permitió avanzar a

las pruebas.

4. Establecimiento de Comunicación y Pruebas de Validación en Simulación (Finales de Septiembre - Octubre)

Durante esta fase, me centré en establecer una comunicación efectiva entre el planificador y el simulador Gazebo, lo que permitió ejecutar trayectorias planificadas y verificar su comportamiento en el entorno simulado. Sin embargo, surgieron errores recurrentes debido a problemas en la comunicación entre los paquetes de ROS y MoveIt!, lo que requería ajustes constantes en los paquetes de configuración del brazo robótico para resolver incompatibilidades y asegurar una comunicación fluida. Esta fase fue crítica para garantizar que el planificador funcionara correctamente en simulación antes de avanzar a las pruebas en un entorno real.

5. Pruebas y Validación en el Laboratorio con el Brazo Real (Tercera Semana de Octubre - Primera Semana de Noviembre)

Tras completar las pruebas en simulación, procedí a realizar pruebas en el laboratorio con el brazo físico Staubli TX2-90. Esta fase incluyó la implementación de un sistema de comunicación y control directo, así como la validación de los movimientos planificados en un entorno real. Durante este periodo, se realizaron ajustes finos en los parámetros de control para asegurar que el brazo operara con precisión y seguridad en el laboratorio. Esta fase fue intensiva, ya que era esencial comprobar que el sistema respondiera adecuadamente en un entorno físico, y cualquier error en los movimientos o en la comunicación podía comprometer la seguridad del equipo. Las pruebas concluyeron exitosamente, logrando los objetivos de precisión y fiabilidad del sistema en el entorno de laboratorio.

6. Documentación (Última Semana de Octubre - Primera Semana de Noviembre)

La fase final del proyecto consistió en la elaboración de este documento, que documenta exhaustivamente todo el proceso de desarrollo. En este documento se detallan los aspectos técnicos, las metodologías empleadas, y las decisiones clave tomadas en cada fase del proyecto. Además, se presentan los problemas encontrados, las soluciones aplicadas y una evaluación crítica de los resultados obtenidos. Esta documentación tiene como objetivo ofrecer una visión completa del proyecto, estructurando la información de manera clara y accesible, y sirviendo como referencia para futuros trabajos o mejoras en proyectos similares.

Aunque el total estimado de horas para este proyecto fue de 300 horas, distribuidas en 12 créditos ECTS, la dedicación real varió considerablemente debido a periodos vacacionales y a la dificultad de compaginar con mis responsabilidades laborales. Hubo semanas en las que pude dedicar más tiempo al proyecto, mientras que en otros momentos el avance fue más lento. Las fases de investigación de planificadores y configuración de MoveIt! fueron las más demandantes, debido a los desafíos técnicos y los errores recurrentes que requirieron una mayor dedicación.

En general, la metodología en cascada permitió una estructuración efectiva del proyecto, aunque la rigidez de su enfoque lineal presentó ciertos desafíos en las fases de investigación y configuración. La necesidad de realizar ajustes en configuraciones ya establecidas retrasó el avance en algunos momentos. Sin embargo, gracias a la dedicación y ajustes realizados en las últimas etapas, el proyecto se completó dentro del tiempo estimado.

Presupuesto

El desarrollo de este proyecto ha implicado distintos costos relacionados con el uso de equipo, tiempo de dedicación y recursos técnicos. A continuación, se detalla el presupuesto estimado considerando cada uno de estos aspectos.

Equipo y Herramientas

El proyecto ha requerido el uso de un equipo de cómputo personal con las siguientes especificaciones:

- **PC Personal:** Aunque este equipo ya estaba en mi posesión y tiene varios años de uso, el costo aproximado de adquisición fue de 700 €.

Además, se ha utilizado el robot industrial **Staubli TX2-90**, facilitado para el desarrollo del proyecto. Aunque el robot no representa un costo directo, su precio de mercado se estima en aproximadamente 50,000 €. Para calcular una aproximación de los costos de uso, consideramos el número de días efectivos de trabajo en laboratorio durante la fase de pruebas y validación.

Horas de Trabajo

El proyecto fue estimado en 300 horas de dedicación, correspondientes a 12 créditos ECTS. En términos de mercado, se puede asignar un valor de 20 €/hora como costo estimado para el trabajo técnico involucrado en el desarrollo y ejecución del proyecto, lo cual incluye investigación, desarrollo, pruebas y documentación.

- **Horas de dedicación:** $300 \text{ horas} \times 20 \text{ €/hora} = 6,000 \text{ €}$

Costos de Amortización del Robot Staubli TX2-90

Dado que el Staubli TX2-90 ha sido utilizado durante la fase de pruebas y validación del proyecto, es importante incluir una estimación de su amortización. Asumiendo una vida útil de 10 años (aproximadamente 2,500 días laborables) y un costo de adquisición de 50,000 €, el costo de amortización por día se calcula en 20 €.

- **Costo de amortización diario:** $50,000 \text{ €} / 2,500 \text{ días} = 20 \text{ €/día}$

Durante el desarrollo del proyecto, el robot fue utilizado durante aproximadamente 4 días efectivos en la fase de pruebas en laboratorio, lo que resulta en un costo estimado de 80 € por el uso del robot.

Resumen de Costos

A continuación se presenta un resumen de los costos estimados del proyecto:

| Concepto | Costo (€) |
|---|----------------|
| PC personal (ya adquirido) | 700 € |
| Horas de trabajo (300 horas × 20 €/hora) | 6,000 € |
| Amortización del robot Staubli TX2-90 (4 días × 20 €/día) | 80 € |
| Total estimado del proyecto | 6,780 € |

Cuadro 1.2: Resumen de costos del proyecto.

En total, el costo estimado del proyecto asciende a 6,780 €, considerando los recursos utilizados y la dedicación requerida en términos de equipo, tiempo y recursos técnicos. Aunque

algunos de estos costos no fueron directos (como el uso del robot y el PC personal), este cálculo permite tener una estimación de los recursos necesarios para el desarrollo de un proyecto de estas características.

Estado del Arte

2.1. Introducción al Estado del Arte

En esta sección se presenta el estado del arte en el ámbito del control y la simulación de brazos robóticos industriales, con un enfoque en las tecnologías, herramientas y metodologías clave utilizadas actualmente en el desarrollo y despliegue de estos sistemas. La capacidad de lograr movimientos precisos y seguros en aplicaciones robóticas depende de un enfoque integral que abarque tanto los métodos de control de movimiento como la programación de trayectorias y la integración en entornos de simulación.

Primero, se revisarán los métodos de control de brazos robóticos industriales, incluyendo la Cinemática Inversa (IK), una técnica fundamental que permite calcular las posiciones de las articulaciones necesarias para que el robot alcance posiciones deseadas en su espacio de trabajo. Estos métodos de control, junto con diversas estrategias de programación de movimiento, son esenciales para la precisión y eficacia de los sistemas robóticos avanzados.

A continuación, en la sección de **Stack de Software y Hardware**, se explorará la computación distribuida y sus alternativas, como DDS, OPC UA, Cortex y UKAEA, que permiten la coordinación y comunicación entre los distintos componentes de un sistema robótico. En este contexto, ROS (*Robot Operating System*) y su extensión ROS-Industrial se destacan como una solución robusta para integrar diversos componentes de software y hardware en el control de robots industriales, siendo esta la elección adoptada en este proyecto.

Posteriormente, se analizarán las opciones de simulación, que juegan un papel fundamental en el desarrollo y la prueba de sistemas robóticos. Se revisarán alternativas de simulación, como Staubli SRS, Webots, Unity Robotics, V-REP y Microsoft AirSim, siendo Gazebo la herramienta elegida en este trabajo por su integración con ROS y sus capacidades de simulación física avanzada.

Además, se discutirá el uso de planificadores de movimiento, con una revisión de alternativas como el Kautham Project, Drake y OMPL. MoveIt, la elección para este proyecto, proporciona capacidades avanzadas de planificación de trayectorias y evasión de colisiones, fundamentales para la seguridad y precisión en entornos industriales.

Por último, se describirá el hardware específico utilizado en este proyecto, el brazo robótico Staubli TX2-90, junto con su controlador CS9 y el driver Val3. Estos componentes permiten la integración de un sistema robótico completo que emplea ROS, ROS-Industrial, Gazebo y MoveIt para implementar y controlar movimientos robóticos avanzados en un entorno de desarrollo y prueba.

La selección de herramientas y metodologías abordada en esta sección se ha realizado con el fin de optimizar la precisión, seguridad y eficiencia en el control y simulación del brazo robótico, permitiendo el desarrollo de aplicaciones robóticas avanzadas tanto en entornos industriales como en investigación.

2.2. Contexto y Problema General

La teleoperación y telemanipulación de brazos robóticos constituye una solución clave en aquellos entornos donde la presencia humana es limitada o presenta riesgos significativos. Esta tecnología permite a los operadores manipular objetos y realizar tareas a distancia, aprovechando sistemas robóticos que replican sus movimientos en tiempo real o de manera autónoma, según las condiciones del entorno. En el caso del presente proyecto, el uso de un sistema de control robótico está justificado debido a la presencia de alta radiación, un entorno que hace imposible la intervención directa de personas por largos períodos.

Los brazos robóticos han encontrado aplicaciones en diversos sectores industriales y científicos, donde el acceso humano es restringido o el nivel de precisión requerido es elevado. Algunos de los casos de uso más relevantes son los siguientes:

- **Industria nuclear:** En plantas de energía nuclear y laboratorios de investigación, los brazos robóticos son empleados para manipular materiales radiactivos, realizar mantenimientos y reparaciones en áreas de difícil acceso, y monitorear entornos de radiación. Su uso minimiza la exposición humana a la radiación, garantizando tanto la seguridad del personal como la precisión de las operaciones.

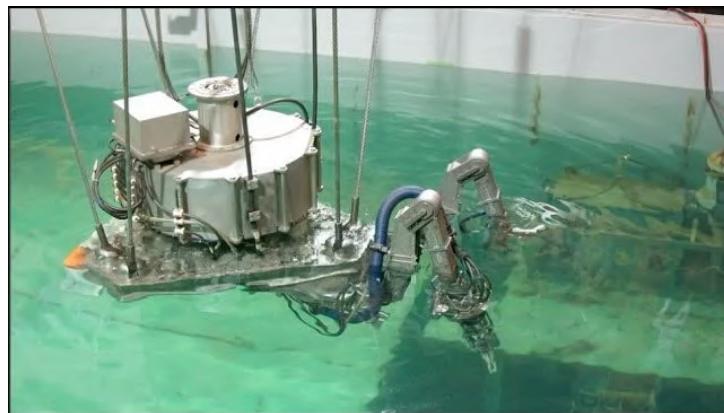


Figura 2.1: Robot utilizado para limpiar escombros en la planta nuclear de Fukushima tras el desastre nuclear de 2011. [2]

- **Industria manufacturera y de automatización:** Los sistemas de teleoperación se utilizan en la manipulación de materiales peligrosos y en el ensamblaje de componentes en líneas de producción donde la repetitividad y exactitud son esenciales. Los brazos robóticos en estas aplicaciones permiten una producción eficiente y segura, además de reducir los errores y optimizar los tiempos de operación.



Figura 2.2: Brazos robóticos en una línea de producción automotriz, utilizados para ensamblar piezas con precisión y eficiencia en procesos automatizados.

- **Medicina y cirugía:** En el ámbito quirúrgico, los robots teleoperados, como el sistema Da Vinci, han revolucionado la cirugía mínimamente invasiva. Los brazos robóticos permiten realizar procedimientos complejos con precisión y reducen el riesgo de errores humanos, proporcionando al cirujano un mayor control y visualización en operaciones delicadas. [7]



Figura 2.3: Sistema robótico da Vinci

- **Exploración espacial y rescate en entornos extremos:** En misiones espaciales y operaciones de rescate en zonas con condiciones extremas (como en el fondo del océano o en áreas contaminadas), los brazos robóticos permiten realizar tareas de manipulación de forma remota, lo cual protege a los operadores y posibilita actividades que serían inviables para una persona.



Figura 2.4: ERA en acción en la Estación Espacial Internacional. [3]

La importancia de estos sistemas radica en su capacidad para extender el alcance y las capacidades de los humanos en situaciones que demandan precisión y control en entornos adversos. En el contexto de este proyecto, el desarrollo de un sistema de control robótico orientado a la teleoperación en un entorno de alta radiación es esencial para asegurar la seguridad, la eficiencia y la continuidad de las operaciones en el marco del proyecto IFMIF-DONES.

2.3. Control de Brazos Robóticos Industriales

El control de brazos robóticos industriales es esencial para ejecutar tareas complejas con precisión y eficiencia en entornos de manufactura. Cada aplicación puede requerir un tipo de control diferente, optimizado para las características específicas de la tarea y del entorno de trabajo. A continuación, se describen los principales métodos de control aplicables a brazos robóticos industriales [8].

2.3.1. Métodos de Control

Existen cuatro métodos principales de control de brazos robóticos industriales:

- **Control Punto a Punto (PTP):** Este método regula la posición y orientación del efecto final del robot en puntos discretos del espacio operativo. El robot se desplaza rápidamente entre estos puntos sin necesidad de especificar la trayectoria exacta, enfocándose en la precisión de posicionamiento y en el tiempo de movimiento. Es comúnmente utilizado en tareas como carga y descarga, manipulación y soldadura por puntos.
- **Control de Trayectoria Continua (CP):** Este enfoque permite el control continuo de la posición y orientación del efecto final, siguiendo trayectorias y velocidades predefinidas con alta precisión. Garantiza movimientos suaves y estables, siendo ideal para aplicaciones como soldadura por arco, pintura y desbarbado.
- **Control de Fuerza (Par):** Este método regula las fuerzas o pares aplicados por el robot, lo cual es fundamental en tareas que requieren una interacción física precisa con el entorno, como en ensamblaje o manipulación delicada. Emplea sensores de fuerza o par para ajustar dinámicamente las acciones del robot en respuesta a los cambios en su entorno.
- **Control Inteligente:** Este método emplea tecnologías de inteligencia artificial para permitir que el robot perciba su entorno y tome decisiones basadas en una base de cono-

cimientos. Proporciona adaptabilidad y capacidad de autoaprendizaje, lo que permite al robot operar eficazmente en entornos dinámicos y no estructurados.

2.3.2. Cinemática Inversa (IK)

La Cinemática Inversa (IK) es una técnica esencial en el control de brazos robóticos, utilizada para calcular las posiciones de las articulaciones necesarias para que el efecto final alcance una posición y orientación específicas en el espacio de trabajo [9, 10]. La IK permite determinar cómo debe moverse cada articulación del robot para conseguir una pose deseada. No obstante, la solución de estas ecuaciones puede ser compleja, generando múltiples soluciones posibles debido a la redundancia de las articulaciones o, en algunos casos, no ofreciendo solución cuando el objetivo se encuentra fuera del alcance del brazo.

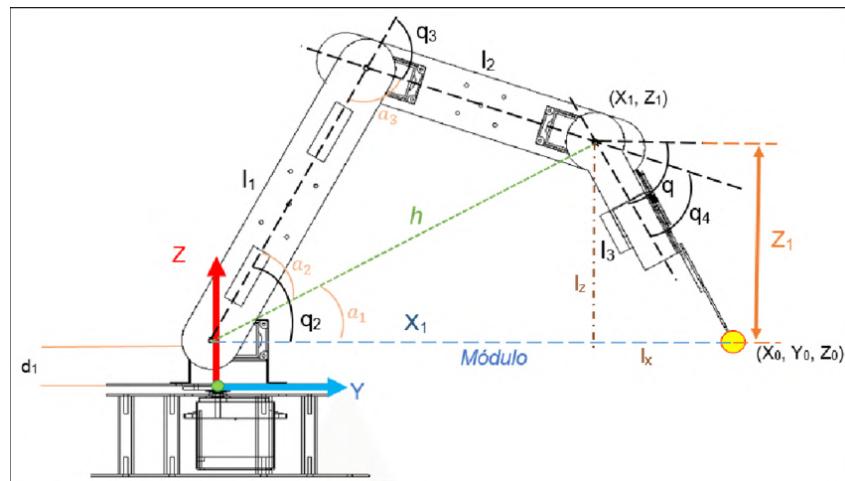


Figura 2.5: Parámetros del brazo robótico para la cinemática inversa

Existen varios métodos para resolver la Cinemática Inversa:

- **Métodos Analíticos:** Estos métodos resuelven la IK mediante un conjunto de ecuaciones cerradas, lo cual permite obtener soluciones exactas en tiempo real. Son especialmente adecuados para robots con estructuras simples y cadenas cinemáticas bien definidas.
- **Métodos Numéricos:** Los métodos numéricos aproximan la solución de IK mediante algoritmos iterativos, optimizando el error entre la pose actual del efecto final y la pose deseada. Métodos como el de Newton-Raphson y el Gradiente Descendente son comunes en este enfoque. Aunque son computacionalmente más intensivos, resultan versátiles y aplicables a estructuras robóticas complejas.
- **Métodos Basados en Optimización:** Estos métodos no solo buscan soluciones que posicione el efecto final en la posición deseada, sino que también optimizan ciertos criterios como la minimización del tiempo, la energía o el desgaste. Este enfoque es especialmente útil en aplicaciones industriales donde la eficiencia operativa es crítica.

2.3.3. Importancia de la Cinemática Inversa en el Control Robótico

La Cinemática Inversa es fundamental para la planificación y ejecución de movimientos complejos en brazos robóticos industriales. Sin IK, sería complicado o incluso imposible para los desarrolladores definir manualmente las posiciones de cada articulación para alcanzar puntos

específicos en el espacio de trabajo. La IK permite que el robot se adapte a cambios en su entorno y es especialmente útil en aplicaciones avanzadas en sectores como la automoción, la electrónica y la fabricación avanzada, donde la precisión y la adaptabilidad son esenciales.

2.4. Stack de Software y Hardware

2.4.1. Computación Distribuida

La computación distribuida en robótica permite la coordinación eficiente de múltiples dispositivos y sistemas dentro de una red, distribuyendo las cargas de procesamiento y facilitando la comunicación en tiempo real. Este enfoque es fundamental para aplicaciones robóticas complejas, donde distintos componentes deben trabajar de forma sincronizada para lograr un desempeño óptimo en tareas colaborativas.

2.4.2. Alternativas en Computación Distribuida

Existen diversas alternativas de computación distribuida aplicables a la robótica, cada una de ellas orientada a mejorar la eficiencia y la escalabilidad en redes de dispositivos. Estas opciones abarcan desde clústeres de procesamiento hasta soluciones en la nube.

DDS (Data Distribution Service)

DDS, o *Data Distribution Service*, es un estándar de comunicación middleware diseñado específicamente para sistemas distribuidos en tiempo real. Este protocolo ofrece una comunicación rápida y eficiente entre dispositivos, permitiendo el intercambio de datos de manera confiable en aplicaciones críticas. DDS es altamente utilizado en sistemas industriales y en aplicaciones de robótica avanzada debido a su capacidad de manejar grandes volúmenes de datos y su flexibilidad para adaptarse a diversas arquitecturas de red [11].

OPC UA (Open Platform Communications Unified Architecture)

OPC UA es un estándar de comunicación diseñado principalmente para la automatización industrial. Este protocolo permite la interoperabilidad entre diferentes sistemas de automatización y control, independientemente del fabricante o de las plataformas utilizadas. OPC UA es conocido por su robustez y seguridad, así como por su capacidad para integrarse con dispositivos de control industrial, facilitando la comunicación en aplicaciones de robótica industrial y manufactura avanzada [12].

Cortex

Cortex es una plataforma de computación distribuida desarrollada para gestionar redes complejas de robots y dispositivos inteligentes. Cortex proporciona una infraestructura para coordinar múltiples robots y permite la toma de decisiones descentralizada, ideal para entornos de robótica colaborativa y escenarios donde múltiples robots deben trabajar de manera coordinada. Es utilizado en aplicaciones donde se requiere escalabilidad y flexibilidad para integrar dispositivos heterogéneos [13].

ROS (Robot Operating System)

ROS es un framework de software de código abierto diseñado para el desarrollo de aplicaciones robóticas, proporcionando herramientas y bibliotecas que facilitan la comunicación y coordinación entre componentes. Este sistema permite a los desarrolladores crear sistemas robóticos complejos y modulares.

ROS 1: La primera versión de ROS utiliza una arquitectura centralizada, donde un nodo maestro gestiona la comunicación entre otros nodos. Aunque esta estructura es adecuada para investigación y prototipado, presenta limitaciones en entornos distribuidos y en sistemas que requieren alta fiabilidad y procesamiento en tiempo real. Debido a su dependencia de un maestro central, ROS 1 tiene dificultades para adaptarse a aplicaciones industriales de gran escala [14].

ROS 2: Diseñado para superar las limitaciones de ROS 1, ROS 2 emplea el estándar DDS (Data Distribution Service) como middleware de comunicación. Esta arquitectura elimina la dependencia de un nodo maestro central, permitiendo una comunicación distribuida más robusta y soporte nativo para sistemas en tiempo real. Con esta estructura, ROS 2 es más adecuado para aplicaciones industriales y entornos donde se requieren alta escalabilidad y fiabilidad en redes de dispositivos complejas [14].

2.4.3. Computación Distribuida Seleccionada

Para este proyecto, se ha escogido **ROS (Robot Operating System)** como plataforma de computación distribuida. Desde el inicio, el proyecto se planteó sobre ROS debido a su flexibilidad, modularidad y la gran cantidad de herramientas y soporte que ofrece para la robótica. Frente a otras alternativas, ROS destaca por su amplia adopción en la comunidad, su arquitectura abierta y su extensión industrial, **ROS-Industrial**, que asegura compatibilidad con hardware de manufactura y soporta entornos de simulación avanzados mediante **Gazebo** y **RViz**. Esta combinación permite desarrollar y probar aplicaciones complejas en un entorno robusto, facilitando la transición del simulador al robot físico.

ROS (Robot Operating System)

ROS, o *Robot Operating System*, es un conjunto de herramientas, bibliotecas y convenciones que simplifican el desarrollo de aplicaciones robóticas. Aunque se denomina "sistema operativo", ROS es en realidad un *middleware* que facilita la comunicación entre los componentes de software en un sistema robótico, permitiendo una plataforma modular que facilita la creación y reutilización de software robótico [15, 16].

Los sistemas robóticos suelen integrar múltiples sensores, actuadores y algoritmos que deben trabajar juntos. ROS facilita esta integración permitiendo que los distintos módulos de software se comuniquen de forma eficiente, lo que permite a los desarrolladores centrarse en las funcionalidades específicas sin preocuparse por la comunicación interna.

Estructura de ROS

La arquitectura de ROS es modular y distribuida, lo que significa que diversos programas individuales, llamados nodos, pueden ejecutarse simultáneamente y comunicarse para realizar tareas complejas. Cada nodo puede controlar un aspecto del sistema, como el manejo de un motor, la recolección de datos de un sensor o la planificación de movimientos.

La comunicación entre nodos se realiza mediante mensajes que se envían en canales llamados *topics*. ROS utiliza un sistema de publicación-suscripción donde un nodo publica mensajes

en un tópico, y otros nodos suscritos a ese tópico los reciben. Este mecanismo permite una comunicación asíncrona y eficaz entre componentes, facilitando el intercambio de información en tiempo real.

Además de los *topics*, ROS ofrece otros métodos de comunicación:

- **Servicios (Services)**: Proporcionan una comunicación síncrona donde un nodo puede enviar una solicitud a otro y esperar una respuesta, ideal para operaciones puntuales, como encender un dispositivo.
- **Acciones (Actions)**: Pensadas para tareas más prolongadas que requieren actualizaciones periódicas, las acciones permiten iniciar una operación, recibir actualizaciones de progreso y cancelar la tarea si es necesario. Esto es útil en tareas complejas como el movimiento de un brazo robótico.

Componentes Principales de ROS

- **Nodos (Nodes)**: Programas individuales que realizan funciones específicas en el sistema robótico, como manejar una cámara, procesar datos de sensores o controlar el movimiento. Esta separación facilita la organización y reutilización del software.
- **Topics**: Canales de comunicación donde los nodos envían y reciben mensajes. El sistema de publicación-suscripción de ROS es ideal para el flujo continuo de datos, como la transmisión de imágenes o las lecturas de un sensor.
- **Servicios (Services)**: Ofrecen una comunicación síncrona que permite interacciones directas y rápidas entre nodos, útiles para operaciones como encender un motor o verificar el estado de un sensor.
- **Acciones (Actions)**: A diferencia de los servicios, están pensadas para tareas de larga duración. Permiten recibir actualizaciones de progreso y cancelar la operación si es necesario, ideal para tareas que implican múltiples pasos.
- **Paquetes (Packages)**: La unidad básica de software en ROS. Un paquete puede contener nodos, scripts, archivos de configuración y bibliotecas. Los paquetes se pueden agrupar en “pilas” (*stacks*) para proporcionar soluciones completas, simplificando la distribución y el mantenimiento del software.

Modularidad y Reutilización de Código

La modularidad de ROS permite crear módulos de software independientes que se pueden reutilizar en distintos proyectos. Esto acelera el desarrollo, ya que permite aprovechar soluciones existentes y centrarse en nuevas funcionalidades. Los paquetes de ROS incluyen código, dependencias, archivos de configuración y documentación, lo que facilita su distribución y colaboración en la comunidad.

La arquitectura distribuida de ROS también permite dividir los sistemas robóticos en partes manejables. Un proyecto robótico complejo puede estar compuesto por nodos distribuidos en diferentes máquinas, cada uno con una función específica, lo que simplifica el desarrollo, prueba y depuración del sistema.

Soporte Multiplataforma e Independencia de Lenguaje

ROS es compatible principalmente con sistemas basados en Unix y su desarrollo principal se realiza en Ubuntu. También hay versiones experimentales para otras distribuciones de Linux y Windows, ampliando su accesibilidad [15].

En términos de programación, ROS es independiente del lenguaje, aunque se utilizan principalmente Python y C++. También existen bibliotecas experimentales en lenguajes como Java, Lisp y Lua, lo cual permite flexibilidad a los desarrolladores.

Comunidad y Colaboración

ROS es un proyecto de código abierto con una comunidad global de desarrolladores, investigadores y empresas. La comunidad crea recursos como tutoriales, foros y repositorios de código que facilitan el aprendizaje y la implementación de ROS. La estructura federada de repositorios permite compartir y distribuir el software de manera eficiente, promoviendo la colaboración y acelerando el progreso en robótica.

Ventajas de ROS

- **Modularidad y Escalabilidad:** Permite que diferentes componentes (nodos) se desarrollen, prueben y reutilicen de manera independiente, facilitando la escalabilidad de sistemas robóticos complejos.
- **Reutilización de Código:** Los paquetes y nodos se pueden compartir y reutilizar en distintos proyectos, acelerando el desarrollo y fomentando la colaboración.
- **Independencia de Lenguaje:** Compatible con múltiples lenguajes de programación como Python, C++ y Java, brindando flexibilidad.
- **Soporte Multiplataforma:** Compatible con Linux y con versiones experimentales para Windows, ampliando su accesibilidad.
- **Comunidad Activa:** Comunidad global que contribuye con recursos y mejoras, creando un ecosistema en constante evolución.

ROS-Industrial: Extensión de ROS para la Industria

ROS-Industrial es una iniciativa que expande las capacidades del *Robot Operating System* (ROS) al ámbito industrial, integrando su software con hardware de manufactura avanzada. Se centra en la compatibilidad con robots industriales, sensores y otros equipos, permitiendo la automatización flexible y la robótica colaborativa en aplicaciones como ensamblaje, soldadura, y procesamiento de materiales [17].

A diferencia de ROS estándar, que está orientado principalmente a la investigación y el desarrollo de aplicaciones robóticas, **ROS-Industrial** adapta el sistema para entornos industriales, asegurando robustez y compatibilidad con equipos de fabricación común. Esto incluye la integración de robots, sensores avanzados como láseres y cámaras, y dispositivos de control industrial, haciendo posible que las empresas optimicen procesos complejos [18].

Características Clave de ROS-Industrial

- **Automatización Flexible:** Permite configurar robots para realizar múltiples tareas industriales sin limitaciones a tareas específicas. Gracias a su enfoque modular, ROS-Industrial facilita la adaptación a nuevas tareas mediante configuraciones reutilizables [19].
- **Interoperabilidad:** Destaca por ser independiente del fabricante, lo que permite que robots y periféricos de diferentes marcas trabajen juntos bajo una misma plataforma. Esto fomenta la integración de hardware diverso sin depender de software propietario, reduciendo costos y tiempos de implementación [17].
- **Comunidad y Colaboración Abierta:** Al igual que ROS, el proyecto es de código abierto y está respaldado por una comunidad global de desarrolladores, investigadores y empresas. Esto ha dado lugar a la creación de múltiples paquetes para tareas industriales complejas, asegurando la innovación y accesibilidad [18].
- **Capacidades Avanzadas de Visión e Inteligencia:** Permite integrar sistemas de visión por computadora y algoritmos de inteligencia artificial para tareas que requieren alta precisión y adaptación, como la clasificación en cintas transportadoras y la manipulación de piezas en líneas de ensamblaje.
- **Simulación y Pruebas:** Utilizando herramientas como Gazebo, las empresas pueden simular entornos de trabajo completos para probar y depurar soluciones antes de implementarlas en plantas reales, reduciendo riesgos y costos.

Aplicaciones Reales de ROS-Industrial

- **Automatización de Procesos de Manufactura:** Sectores como el automotriz, aeroespacial y electrónico lo usan para tareas precisas como ensamblaje de componentes, soldadura y manipulación de materiales [19].
- **Robótica Colaborativa:** Facilita la integración de robots colaborativos (cobots) que trabajan junto a humanos, ajustándose dinámicamente para interactuar de forma segura y eficiente.
- **Inspección y Control de Calidad:** Integrado con sistemas de visión para la inspección en tiempo real, detectando defectos y asegurando la calidad.

Conexión con ROS

ROS-Industrial se basa en la misma infraestructura de ROS, aprovechando su enfoque modular, reutilización de software y filosofía de colaboración abierta. Sin embargo, se adapta para cumplir con las necesidades del sector industrial, ofreciendo robustez y precisión para integrar maquinaria y protocolos industriales ya existentes. Ambos proyectos comparten la misma esencia de comunidad abierta, pero ROS-Industrial eleva estas capacidades para satisfacer las demandas complejas de la manufactura moderna [17].

Futuro de ROS-Industrial

El proyecto sigue evolucionando, mejorando la compatibilidad, seguridad y robustez. Se explora la integración con tecnologías emergentes como la inteligencia artificial, aprendizaje

automático y realidad aumentada, lo que permitirá capacidades avanzadas de monitoreo y diagnóstico. Además, consorcios en América, Europa y Asia apoyan la estandarización y expansión global del proyecto, impulsando la innovación en la automatización industrial [18].

RViz: Herramienta de Visualización en ROS

RViz es una herramienta de visualización dentro del ecosistema ROS (*Robot Operating System*) que permite representar visualmente los datos de un sistema robótico en un entorno tridimensional. Su principal objetivo es proporcionar a los desarrolladores y operadores de robots una representación intuitiva y en tiempo real del estado del sistema y de sus datos de sensores. Al hacerlo, facilita el desarrollo, prueba y depuración de robots en simulación o en hardware real [20].

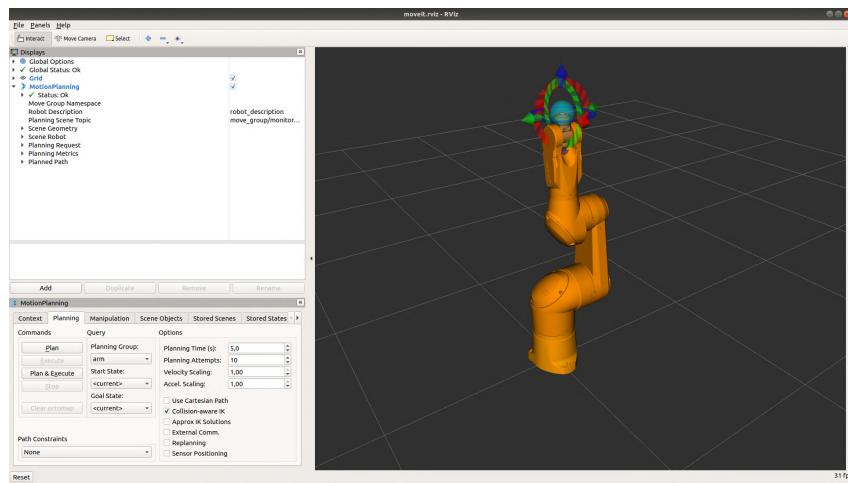


Figura 2.6: Modelo del Staubli TX-90 en RViz

Funciones Principales de RViz

- **Visualización de Modelos de Robots:** RViz permite cargar y mostrar modelos de robots en 3D, definidos generalmente en formato URDF (*Unified Robot Description Format*). Estos modelos representan la estructura física del robot, con sus enlaces y articulaciones. Gracias a esta visualización, los desarrolladores pueden observar cómo cada componente del robot se posiciona y mueve en el espacio, facilitando la comprensión de su cinemática y estructura.
- **Simulación de Trayectorias y Planificación de Movimientos:** En combinación con MoveIt (un software de planificación de movimientos), RViz permite planificar y visualizar trayectorias de movimiento. Los desarrolladores pueden simular trayectorias en el espacio de trabajo, asegurándose de que los movimientos planeados sean seguros y estén libres de colisiones antes de implementarlos en el robot real. Esto resulta crucial para reducir riesgos y optimizar el rendimiento del sistema.
- **Visualización de Datos de Sensores:** RViz puede mostrar datos de sensores conectados al robot, como cámaras, escáneres láser, sensores de profundidad, y otros dispositivos de percepción. La capacidad de visualizar en tiempo real los datos capturados permite analizar y comprender mejor cómo el robot percibe su entorno, lo cual es esencial para tareas de navegación y manipulación.

- **Representación del Entorno y de Elementos del Escenario:** En RViz se pueden agregar elementos visuales como marcadores, objetos estáticos y dinámicos, que representan obstáculos, puntos de referencia o partes del entorno de trabajo del robot. Esto permite crear una representación fiel del espacio de trabajo, lo cual ayuda en la planificación de trayectorias y en el análisis de interacción entre el robot y su entorno.
- **Interfaz Modular y Personalizable:** La interfaz de RViz es altamente modular, permitiendo a los usuarios personalizar qué datos desean visualizar. Cada componente o "display" (panel de visualización) en RViz es configurable, facilitando la organización y presentación de información específica según las necesidades de cada proyecto. Esto permite a los desarrolladores enfocarse en aspectos específicos del sistema, como sensores, estado de las articulaciones, o trayectorias de movimiento.

Usos y Ventajas de RViz en Proyectos Robóticos

- **Pruebas y Depuración:** RViz permite a los desarrolladores observar en tiempo real cómo el robot responde a comandos, mostrando visualmente el estado de sus componentes y datos de sus sensores. Esto facilita la identificación y resolución de problemas.
- **Simulación y Validación de Movimientos:** Antes de implementar movimientos en el hardware, los desarrolladores pueden simular y validar trayectorias para evitar errores en el entorno real.
- **Desarrollo Ágil en Entornos Simulados y Reales:** La integración con otros componentes de ROS, como Gazebo y MoveIt, convierte a RViz en una herramienta versátil que se adapta tanto a entornos de simulación como a configuraciones de hardware real.

2.4.4. Simulación

La simulación es una herramienta fundamental en robótica que permite probar y validar algoritmos y sistemas en un entorno seguro y controlado. Los simuladores permiten modelar tanto el robot como su entorno, lo que facilita el desarrollo, la depuración y el análisis de los sistemas robóticos sin necesidad de utilizar equipos físicos. Existen varias alternativas de simulación en robótica, cada una con sus propias características y aplicaciones específicas. A continuación, se describen algunas de las alternativas más destacadas, incluyendo el simulador Gazebo, que fue la opción seleccionada para este proyecto.

2.4.5. Alternativas en Simulación

Staubli Robotics Suite (SRS)

El **Staubli Robotics Suite (SRS)** es un simulador industrial específico para robots Staubli. Este software permite crear entornos de simulación para probar, configurar y optimizar robots Staubli en aplicaciones industriales. SRS está diseñado para integrar tanto la programación del controlador como la simulación, lo que facilita la validación de trayectorias y procesos antes de su implementación en el hardware físico. Este simulador es especialmente útil en entornos de manufactura, ya que permite a los ingenieros y técnicos probar operaciones de manipulación, ensamblaje y soldadura en un entorno virtual seguro. Sin embargo, su uso está restringido principalmente a robots Staubli, lo cual limita su aplicabilidad a otros tipos de robots [21].

V-REP (CoppeliaSim)

V-REP, actualmente conocido como **CoppeliaSim**, es un simulador de robots altamente versátil que soporta múltiples tipos de robots y aplicaciones. Este simulador permite modelar robots en 3D y ofrece un motor de física que permite la simulación precisa de dinámicas, colisiones y contactos. CoppeliaSim es compatible con varios lenguajes de programación y permite la integración con sistemas como ROS, lo que facilita el desarrollo de aplicaciones complejas. Una de sus principales ventajas es su flexibilidad, ya que se puede personalizar a través de scripts, plugins y módulos externos, adaptándose a diferentes necesidades de simulación en robótica [22].

Webots

Webots es un simulador de robots de código abierto que ofrece un entorno de simulación detallado para el modelado de robots móviles y manipuladores. Webots permite simular una gran variedad de robots y sensores, y es compatible con múltiples lenguajes de programación. Su facilidad de uso y su integración con ROS lo han convertido en una opción popular en entornos educativos e investigativos. Webots ofrece un motor de física realista que permite simular el comportamiento de robots en diversos entornos, incluyendo terrenos accidentados, superficies inclinadas y espacios confinados. Este simulador es ampliamente utilizado para la simulación de robots móviles en entornos de navegación y exploración [23].

Unity Robotics

Unity, conocido principalmente como un motor de videojuegos, también se utiliza para simulación de robots a través de **Unity Robotics Hub**. Unity permite crear entornos 3D inmersivos y visualmente detallados, y su integración con ROS facilita la simulación de robots en estos entornos. Aunque Unity no fue diseñado específicamente para robótica, su capacidad gráfica y su motor de física de alta calidad lo hacen una alternativa atractiva para aplicaciones donde el realismo visual es importante, como la simulación de robots en entornos complejos o la visualización de robots en interacción con humanos. Unity Robotics es especialmente útil en proyectos de investigación donde es importante modelar entornos visualmente complejos [24].

Microsoft AirSim

Microsoft AirSim es un simulador de código abierto desarrollado por Microsoft y basado en Unreal Engine. Este simulador está especializado en la simulación de drones y vehículos autónomos, y es compatible con ROS. AirSim proporciona un entorno realista para probar algoritmos de navegación, percepción y control en simulaciones de vuelo, lo que es ideal para aplicaciones de vigilancia, agricultura de precisión y entrega autónoma. Además, al estar basado en Unreal Engine, AirSim permite crear escenarios visualmente ricos, lo cual es útil en investigación y desarrollo de inteligencia artificial aplicada a la robótica [25].

Gazebo

Gazebo es un simulador de código abierto popular en robótica, conocido por su integración con ROS y su capacidad para crear simulaciones realistas en 3D. Es ampliamente utilizado en investigación y desarrollo para probar algoritmos en entornos virtuales antes de su implementación física [26].

2.4.6. Simulador Seleccionado

Para este proyecto, se ha seleccionado **Gazebo** como simulador principal, debido a su integración nativa con ROS y su capacidad para realizar simulaciones físicas realistas. Desde un inicio, Gazebo fue una opción natural, ya que los paquetes de **ROS-Industrial** utilizados en el proyecto incluyen configuraciones y modelos específicos para este simulador, facilitando así la implementación. Además, Gazebo permite una transición fluida entre el entorno simulado y el entorno real, ofreciendo una representación precisa de los comportamientos del robot en un entorno seguro y controlado.

Gazebo

Gazebo es un simulador de robots de código abierto que permite crear entornos virtuales en 3D donde se pueden diseñar, probar y validar sistemas robóticos. Ofrece simulaciones físicas realistas, gráficos avanzados y compatibilidad con una variedad de sensores virtuales, lo que facilita la simulación de robots en entornos complejos tanto interiores como exteriores. A diferencia de los motores de videojuegos, Gazebo se centra en la precisión física, lo que lo convierte en una herramienta esencial para el desarrollo y la investigación en robótica [26].

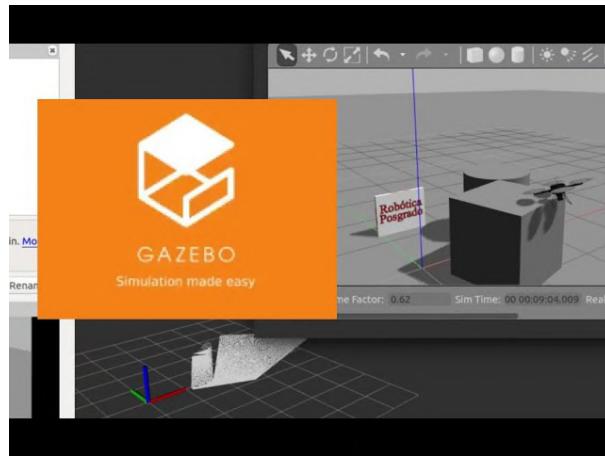


Figura 2.7: Gazebo

A diferencia de Gazebo, que se utiliza para simulaciones físicas completas, **RViz** es una herramienta de visualización enfocada en mostrar datos en tiempo real del estado del robot, tales como su posición, mapas y detecciones de sensores. Mientras que Gazebo permite probar y validar dinámicas y comportamiento de robots en entornos virtuales, RViz es utilizado para monitorizar y depurar en tiempo real, visualizando la información sensorial y de planificación en robots reales o en simulaciones.

Características principales de Gazebo

- **Simulación Física Realista:** Emplea motores de física como ODE, Bullet y DART para simular dinámicas físicas detalladas, incluyendo gravedad, fricción y colisiones [27].
- **Gráficos en 3D y Visualización:** Ofrece una interfaz gráfica avanzada para visualizar entornos complejos y permite a los usuarios controlar la simulación y analizar el comportamiento de los robots. También se puede interactuar con el entorno mediante plugins personalizados [27].

- **Sensores Virtuales:** Simula sensores comúnmente utilizados en robótica, como cámaras, LIDAR, GPS y sensores de fuerza, facilitando el desarrollo y la validación de algoritmos de percepción y navegación.
- **Extensibilidad y Personalización:** Los usuarios pueden crear sus propios modelos de robots y mundos, además de agregar plugins para personalizar comportamientos específicos de los robots y sensores.
- **Integración con ROS (Robot Operating System):** Gazebo se integra perfectamente con ROS, permitiendo controlar y monitorear robots simulados utilizando el mismo software que se emplearía en robots físicos, facilitando la transición de la simulación a la implementación real [27].

¿Para qué se usa Gazebo?

- **Desarrollo y Pruebas de Algoritmos:** Los desarrolladores pueden probar algoritmos de navegación, percepción y control en un entorno seguro antes de llevarlos al hardware físico, lo que ahorra tiempo y recursos.
- **Educación:** Universidades y centros educativos usan Gazebo para enseñar conceptos básicos y avanzados de robótica, permitiendo a los estudiantes experimentar sin la necesidad de equipos costosos.
- **Investigación:** Investigadores prueban nuevas tecnologías y algoritmos en Gazebo, permitiendo replicar condiciones específicas de manera repetible y controlada.
- **Diseño y Prototipado:** Facilita la creación de prototipos virtuales de robots, permitiendo a ingenieros y diseñadores probar configuraciones y diseños sin construir versiones físicas.
- **Simulación de Entornos Complejos:** Gazebo puede modelar entornos complejos como fábricas, ciudades y campos, permitiendo a los usuarios probar robots en situaciones como logística, rescate, agricultura y más.

Ejemplos de Uso

- **Robots Móviles:** Simulación de navegación autónoma para robots móviles que deben evitar obstáculos y seguir rutas predeterminadas.
- **Drones:** Pruebas de control y navegación para drones en misiones de vigilancia y entrega.
- **Robots Industriales:** Simulación de tareas de ensamblaje y manipulación en líneas de producción industriales.
- **Entornos Inteligentes:** Modelado de entornos con múltiples robots interactuando para aplicaciones en logística y automatización.

Ventajas de Usar Gazebo

- **Reducción de Costos:** Permite probar robots en un entorno virtual sin necesidad de construir prototipos físicos costosos.
- **Ambiente Seguro y Controlado:** Las pruebas complejas se pueden realizar sin riesgo para equipos físicos, lo que es ideal para la experimentación e investigación.

- **Rápida Iteración:** Los desarrolladores pueden modificar y probar rápidamente sus diseños, acelerando el ciclo de desarrollo.
- **Escalabilidad:** Permite desde la simulación de un solo robot hasta múltiples robots en entornos complejos, adaptándose a diferentes necesidades y aplicaciones.

2.4.7. Planificadores

La planificación de movimientos es esencial en robótica, permitiendo que los robots realicen trayectorias complejas y ejecuten tareas de manera precisa y segura. Existen diversas herramientas de planificación de movimientos, cada una con sus propias fortalezas y aplicaciones. A continuación, se presentan algunas alternativas utilizadas en la planificación de movimientos en robótica, incluyendo la solución seleccionada para este proyecto: MoveIt.

2.4.8. Alternativas en Planificación de Movimientos

Kautham Project

El **Kautham Project** es una plataforma de software para la planificación de movimientos de robots, desarrollada en la Universidad Politécnica de Cataluña (UPC). Este proyecto está diseñado para apoyar la investigación y la educación en el campo de la robótica y la automatización, proporcionando un entorno flexible en el que los usuarios pueden experimentar con diferentes algoritmos de planificación y estudiar dinámicas de robots en entornos virtuales controlados.

Kautham permite a los investigadores probar y evaluar métodos avanzados de planificación de trayectorias, con aplicaciones en tareas complejas como la manipulación de objetos, la navegación en entornos dinámicos y la evasión de colisiones. Su modularidad y capacidad de integración lo hacen ideal para entornos académicos y de investigación, en los que se requiere explorar nuevos enfoques y metodologías de planificación. Al admitir múltiples algoritmos de planificación, como el *Probabilistic Roadmap* (PRM) y el *Rapidly-exploring Random Trees* (RRT), Kautham permite la comparación y experimentación con técnicas de planificación probabilística y de muestreo.

Además de estos algoritmos, Kautham se integra con el motor de simulación física Open Dynamics Engine (ODE), que facilita simulaciones físicas realistas, incluyendo la interacción del robot con su entorno. Esto permite modelar fuerzas, colisiones y fricciones, lo cual es fundamental para probar trayectorias en condiciones que simulan el comportamiento real del robot en un entorno físico.

En el ámbito industrial, el Kautham Project incluye herramientas especializadas para manipuladores robóticos, entre ellas la biblioteca **TXRobot**, que proporciona una estructura modular y extensible para la planificación de movimientos en manipuladores específicos, adaptándose a las necesidades de precisión y seguridad de entornos industriales controlados [28].

OMPL (Open Motion Planning Library)

La **Open Motion Planning Library (OMPL)** es una biblioteca de planificación de movimientos de código abierto que proporciona una serie de algoritmos de planificación de trayectorias. OMPL está centrada en la planificación en el espacio de configuración y soporta varios algoritmos de muestreo, como RRT, PRM y sus variantes. Aunque OMPL se usa principalmente como una biblioteca de backend, muchas otras herramientas de planificación, incluido MoveIt,

utilizan OMPL para calcular trayectorias. La flexibilidad de OMPL la convierte en una excelente opción para proyectos de investigación y sistemas donde se desea implementar algoritmos de planificación personalizados [29].

Drake

Drake es una biblioteca desarrollada por el MIT que proporciona un entorno de simulación y herramientas para la planificación de movimientos y el control de robots. Drake incluye métodos avanzados de planificación y optimización de trayectorias, así como simulaciones físicas detalladas. Es especialmente adecuado para aplicaciones que requieren simulación de alta fidelidad y cálculos complejos de cinemática y dinámica. Aunque Drake es más complejo y no tan especializado en robótica como MoveIt, su capacidad para manejar simulaciones detalladas y entornos dinámicos lo hace útil en proyectos de investigación avanzada [30].

2.4.9. Planificador Seleccionado

Para este proyecto, se ha elegido **MoveIt** como planificador principal por su integración con ROS y su capacidad para la planificación de movimientos en manipuladores industriales. MoveIt facilita la creación de trayectorias precisas y la evasión de colisiones, aspectos esenciales para los requisitos del proyecto. Además, su compatibilidad con los paquetes de **ROS-Industrial** permite una configuración optimizada para el brazo Staubli, facilitando así la implementación en simulación y en el robot real.

MoveIt: Planificación y Control de Movimientos en Robots

MoveIt es una herramienta de planificación de movimientos para robots dentro del ecosistema ROS (*Robot Operating System*). Es ampliamente utilizada en robótica para tareas de manipulación, planificación de trayectorias, evasión de colisiones y control de movimientos de brazos robóticos, manipuladores y sistemas móviles. Su propósito principal es facilitar la creación de movimientos precisos, seguros y eficientes en entornos dinámicos y complejos [31].

¿Para Qué Se Utiliza MoveIt?

- **Planificación de Movimientos:** MoveIt permite planificar trayectorias para brazos robóticos y otros manipuladores. Esto significa que los desarrolladores pueden definir un punto de inicio y un objetivo, y MoveIt calculará una trayectoria que el robot puede seguir para llegar al destino deseado [32].
- **Evasión de Colisiones:** MoveIt incorpora un motor de detección de colisiones que asegura que las trayectorias planificadas no colisionen con el entorno ni con el propio robot. Esta funcionalidad es clave para la seguridad, especialmente en entornos industriales o de alta precisión.
- **Manipulación de Objetos:** MoveIt permite integrar datos de sensores, como cámaras o escáneres de profundidad, para identificar y manipular objetos en el entorno. Esto es fundamental en aplicaciones de *picking* y montaje, donde el robot debe detectar y agarrar objetos de manera precisa.
- **Control Cinemático y Dinámico:** MoveIt ofrece herramientas tanto para la cinemática (movimientos) del robot como para la planificación dinámica, lo que permite optimizar los movimientos en función de la velocidad, la aceleración y la capacidad de carga del robot.

- **Simulación y Validación en Tiempo Real:** En combinación con RViz y simuladores como Gazebo, MoveIt permite validar y probar las trayectorias planificadas antes de ejecutarlas en el hardware real, ayudando a evitar errores costosos o peligrosos.

¿Cómo Funciona MoveIt?

MoveIt se compone de varios módulos que, en conjunto, permiten realizar la planificación y el control de movimientos de manera eficiente:

- **Cinemática Inversa (IK - Inverse Kinematics):** MoveIt utiliza cálculos de cinemática inversa para determinar cómo debe moverse cada articulación del robot para alcanzar una posición o pose deseada en el espacio. Esto es esencial para lograr posiciones específicas del manipulador (por ejemplo, una pinza o herramienta al final de un brazo robótico). Varios métodos de IK están disponibles en MoveIt, y los usuarios pueden elegir el más adecuado según el tipo y la configuración del robot.
- **Módulo de Detección de Colisiones:** MoveIt incluye un motor de detección de colisiones que analiza tanto el modelo del robot como los objetos en el entorno para evitar colisiones. Este módulo asegura que las trayectorias sean seguras y que el robot no se mueva a través de obstáculos. Las colisiones se detectan basándose en los modelos 3D del robot y del entorno, y se configuran parámetros para determinar zonas de seguridad y márgenes de error.
- **Planificación de Trayectorias:** MoveIt ofrece varios algoritmos de planificación de trayectorias, como los algoritmos de planificación basados en gráficos (*Graph-Based Planners*) y de muestreo (*Sampling-Based Planners*), incluyendo algoritmos como RRT (*Rapidly-exploring Random Tree*) y PRM (*Probabilistic Roadmap*). Estos algoritmos calculan trayectorias que cumplen con las restricciones definidas (evitación de colisiones, límites de velocidad y aceleración) y permiten que el robot realice movimientos eficientes y seguros.
- **Simulación y Visualización:** A través de la integración con RViz, MoveIt permite visualizar las trayectorias planificadas en un entorno 3D. Los usuarios pueden observar y ajustar los movimientos del robot antes de ejecutarlos en el hardware real. Esta simulación visual es fundamental para ajustar detalles de las trayectorias y verificar la precisión en entornos complejos o con varios obstáculos.
- **Control de Ejecución:** MoveIt no solo planifica trayectorias sino que también proporciona controladores para ejecutar movimientos en tiempo real en el robot. Estos controladores garantizan que las trayectorias se sigan de manera precisa, respetando las restricciones de velocidad, aceleración y sincronización de las articulaciones. La ejecución de los movimientos puede realizarse en simulación o directamente en el hardware del robot, facilitando el desarrollo ágil y seguro.

Ventajas de Usar MoveIt

- **Flexibilidad:** Compatible con una amplia variedad de robots y manipuladores, MoveIt se adapta a distintas configuraciones y tipos de tareas, desde brazos robóticos en aplicaciones industriales hasta manipuladores de precisión en laboratorios.
- **Eficiencia y Seguridad:** Al incorporar planificación de trayectorias y detección de colisiones, MoveIt permite movimientos eficientes y seguros, minimizando el riesgo de errores y daños en el entorno de trabajo.

- **Integración en ROS:** MoveIt es una extensión de ROS, lo que facilita su uso en sistemas robóticos complejos que ya emplean ROS para comunicación y control, permitiendo así la coordinación con otros módulos y sensores.

2.4.10. Brazo Robótico Staubli TX2-90

El Staubli TX2-90 es un brazo robótico industrial de seis ejes diseñado para combinar alta velocidad, precisión y seguridad en entornos industriales complejos. Forma parte de la línea TX2 de Staubli, que incluye robots colaborativos adaptados para interactuar de manera segura con los operadores humanos, lo que amplía su versatilidad en diversas aplicaciones industriales.



Figura 2.8: Brazo Robótico Staubli TX2-90.

Características Técnicas

El TX2-90 está optimizado para una amplia gama de aplicaciones industriales, destacando en términos de diseño, integración, rendimiento, seguridad, confiabilidad y conectividad. Sus principales características incluyen:

- **Alcance y Capacidad de Carga:** El TX2-90 tiene un alcance máximo de hasta 1450 mm y una capacidad de carga de hasta 14 kg, lo que lo hace adecuado para manejar piezas y herramientas de tamaño mediano en distintas configuraciones de producción.
- **Diseño Inteligente y Resistencia:** El brazo robótico TX2-90 cuenta con un diseño encapsulado y sin cables externos, compatible con entornos exigentes, desde sucios hasta estériles. Ofrece protección IP65 en todo el brazo, y con un kit de presurización, es adecuado para operar en condiciones que requieren resistencia al polvo y la humedad.
- **Flexibilidad de Integración:** El TX2-90 permite una integración modular con montaje en 360° y un amplio rango esférico de trabajo. Está diseñado para una fácil integración de componentes eléctricos y neumáticos, incluyendo circuitos Ethernet Cat5e encapsulados para facilitar la conexión de cámaras y sensores adicionales.
- **Cinemática de Alto Rendimiento:** Este modelo incorpora un sistema de caja de engranajes inteligente JCS patentado por Staubli, optimizado para robots compactos. También

utiliza encoders absolutos digitales de 19 bits, que permiten una precisión excepcional en tareas guiadas por cámara y alta repetibilidad en movimientos.

- **Conectividad Avanzada:** El TX2-90 es compatible con protocolos de comunicación industrial estandarizados, incluyendo Ethernet en tiempo real, y cuenta con un servidor web integrado en su controlador CS9. Esto facilita la conectividad y el monitoreo de datos de producción en tiempo real, adaptándose a la digitalización en la industria.

Para obtener más detalles técnicos e imágenes adicionales del TX2-90, consulte el Apéndice A.

Funciones de Seguridad y Colaboración

El TX2-90 está diseñado para operar en entornos colaborativos, garantizando una interacción segura con operadores humanos mediante una serie de funcionalidades de seguridad certificadas bajo SIL3/PLe. Estas incluyen:

- **Velocidad Limitada Segura:** Controla la velocidad del robot para minimizar riesgos en caso de proximidad con operadores.
- **Parada Segura y Zonas Seguras:** Permite al robot detenerse de forma segura o limitar sus movimientos a zonas predefinidas.
- **Control de Herramientas Seguras:** Mantiene el control de las herramientas del robot para evitar incidentes durante la colaboración.
- **Interacción Táctil Segura:** En su versión TX2touch, permite una interacción directa con el robot de manera segura.

Aplicaciones Industriales

Gracias a su combinación de precisión, velocidad y seguridad, el TX2-90 se utiliza en diversas aplicaciones industriales, tales como:

- **Manufactura y Ensamblaje:** Adecuado para operaciones de ensamblaje y montaje en líneas de producción que requieren precisión y alta repetibilidad.
- **Manipulación de Materiales:** Ideal para la transferencia, empaquetado y manejo de materiales en procesos automatizados.
- **Industria Electrónica:** Permite el ensamblaje de componentes delicados y pequeños, gracias a su precisión y flexibilidad.
- **Medicina y Farmacéutica:** Con su diseño higiénico, el TX2-90 es adecuado para entornos estériles y aplicaciones con altos estándares de limpieza y seguridad.

El TX2-90 destaca como una solución avanzada y segura para la robótica colaborativa, brindando una alta productividad y adaptabilidad en diversos sectores industriales.

Driver Val3

El **driver Val3** es un controlador desarrollado específicamente para robots Staubli, permitiendo su programación y control mediante el sistema Val3, un lenguaje propio de Staubli. Val3 proporciona una plataforma de alta precisión y fiabilidad para tareas de automatización industrial.

1. Características principales:

- **Flexibilidad de programación:** Permite controlar movimientos, gestionar entradas y salidas, y configurar parámetros específicos del robot.
 - **Integración en red:** A través de Ethernet y TCP/IP, facilita la conexión del robot a redes industriales y su control remoto.
 - **Control preciso:** Ajusta velocidad, aceleración y precisión de movimientos, ideal para tareas industriales complejas.
2. **Arquitectura:** Val3 se comunica con el controlador CS8 o CS9 del robot para interpretar y ejecutar comandos en tiempo real, permitiendo también la ejecución de múltiples tareas simultáneamente.
3. **Integración con ROS:** Permite el control del robot desde ROS mediante ROS-Industrial y el protocolo TCP/IP, facilitando el uso de herramientas avanzadas de planificación como MoveIt y simuladores como Gazebo.
4. **Aplicaciones:** Utilizado en tareas de ensamblaje de precisión, manipulación de materiales y robótica colaborativa, el driver Val3 es adecuado para entornos de producción industrial que requieren fiabilidad y precisión.

En resumen, el driver Val3 permite controlar robots Staubli con precisión y flexibilidad, ofreciendo integración con redes industriales y plataformas de desarrollo como ROS para aplicaciones avanzadas en automatización [33].

Controlador CS9

El **controlador CS9** de Staubli es una plataforma avanzada de control de robots diseñada para mejorar la seguridad, precisión y conectividad de los robots industriales de la serie TX2. Este controlador es ideal para aplicaciones de automatización en entornos colaborativos y de alta demanda.

- **Seguridad mejorada:** El CS9 cumple con estándares de seguridad internacionales como ISO 10218-1 y ISO/TS 15066, proporcionando funciones avanzadas de seguridad que permiten su uso en aplicaciones colaborativas junto a operadores humanos. Esto incluye zonas de seguridad configurables y monitorización de velocidad para garantizar un entorno seguro.
- **Integración en redes industriales:** Admite múltiples protocolos de comunicación, incluyendo Ethernet, Modbus, ProfiNet y EtherCAT, lo que facilita su integración en entornos industriales y de automatización complejos.
- **Control de precisión:** El CS9 permite ajustar la velocidad y la trayectoria del robot con alta precisión, lo que lo hace adecuado para tareas de ensamblaje, manipulación y procesos de fabricación que requieren control detallado.

- **Monitorización avanzada:** Incorpora capacidades de monitorización en tiempo real que permiten supervisar el estado y las variables operativas del robot, optimizando así el mantenimiento preventivo y la productividad.

En resumen, el controlador CS9 combina una arquitectura segura y precisa con una conectividad robusta, ofreciendo una solución confiable y adaptable para la automatización avanzada en industrias que requieren alto nivel de seguridad y precisión.[34]

Metodología de Trabajo

3.1. Configuración de ROS

Esta sección describe los pasos básicos para configurar ROS Melodic, crear un espacio de trabajo, instalar paquetes necesarios y utilizar comandos esenciales de ROS. Además, se explican los pasos para clonar el repositorio del proyecto y asegurarse de que el entorno esté correctamente configurado para el desarrollo.[35]

Creación de un Espacio de Trabajo

Un espacio de trabajo (*workspace*) en ROS organiza todos los paquetes y archivos necesarios para el desarrollo. Los pasos son:

1. **Crear el Directorio del Workspace:** El primer paso para configurar un workspace en ROS es crear el directorio que contendrá todos los paquetes y archivos de desarrollo. Para ello, abre una terminal y ejecuta el siguiente comando:

```
mkdir -p ~/catkin_ws/src
```

Este comando crea un directorio llamado `catkin_ws` en tu carpeta de inicio (`~/`), el cual actuará como el workspace principal. Dentro de `catkin_ws`, se crea una subcarpeta llamada `src` (source) donde se colocarán todos los paquetes que desarrolles o descargues en el futuro.

2. **Inicializar el Workspace:** Una vez creado el directorio del workspace, es necesario inicializarlo para que ROS lo reconozca como un entorno válido. Para ello, navega al directorio principal del workspace y ejecuta el comando `catkin_make`:

```
cd ~/catkin_ws
catkin_make
```

El comando `catkin_make` compila el workspace, generando dos carpetas adicionales: `build` y `devel`. La carpeta `build` contiene los archivos generados durante la compilación, mientras que la carpeta `devel` incluye scripts y configuraciones necesarios para ejecutar los paquetes. Este proceso de inicialización debe repetirse cada vez que se añadan o modifiquen paquetes en el workspace.

3. **Clonar el Repositorio del Proyecto:** Una vez creado e inicializado el workspace, clona el repositorio del proyecto en la carpeta `src`. Esto añade los archivos y configuraciones específicos necesarios para el proyecto. Para ello, ejecuta:

```
cd ~/catkin_ws/src  
git clone https://github.com/juanantms/TFG.git
```

Esto descargará todos los archivos del proyecto en el directorio `src`, listos para ser compilados y utilizados en el entorno ROS.

4. **Configurar el Workspace en el Entorno:** Para que ROS pueda encontrar automáticamente los paquetes de tu workspace, es necesario añadirlo al entorno de ROS. Esto se hace añadiendo una línea al archivo `.bashrc`, que es un archivo de configuración que se ejecuta cada vez que abres una nueva terminal. Ejecuta los siguientes comandos:

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

El primer comando añade el workspace al archivo `.bashrc` ejecutando el script `setup.bash`, que configura las rutas y variables necesarias para que ROS reconozca el workspace. El segundo comando, `source ~/.bashrc`, recarga el archivo `.bashrc` para aplicar inmediatamente los cambios, permitiendo que los paquetes en `catkin_ws` estén disponibles sin necesidad de reiniciar la terminal.

Instalación de Paquetes Básicos de ROS

Para usar herramientas comunes de ROS, instala la versión completa de ROS Melodic:

```
sudo apt-get install ros-melodic-desktop-full
```

Instalación de Paquetes y Dependencias Adicionales

Además de la instalación básica, es necesario instalar varios paquetes adicionales para garantizar la funcionalidad completa en simulación, control y manipulación con ROS. Estos incluyen controladores, herramientas de simulación en Gazebo, y MoveIt! para la planificación de trayectorias.

Para instalar todos los paquetes necesarios, ejecuta el siguiente comando en una terminal:

```
sudo apt-get install ros-melodic-gazebo-ros-pkgs  
ros-melodic-gazebo-ros-control  
ros-melodic-controller-manager  
ros-melodic-joint-state-controller  
ros-melodic-position-controllers  
ros-melodic-effort-controllers  
ros-melodic-joint-trajectory-controller ros-melodic-moveit  
ros-melodic-ros-control ros-melodic-ros-controllers  
ros-melodic-transmission-interface ros-melodic-xacro  
ros-melodic-joint-state-publisher  
ros-melodic-joint-state-publisher-gui  
ros-melodic-rviz
```

Este comando instala los siguientes paquetes:

- **ros-melodic-gazebo-ros-pkgs** y **ros-melodic-gazebo-ros-control**: Necesarios para ejecutar y controlar simulaciones en Gazebo.
- **Controladores**: Paquetes como `controller-manager`, `joint-state-controller`, `position-controllers`, y `effort-controllers` son necesarios para activar y gestionar los diferentes tipos de control sobre las articulaciones del robot.
- **MoveIt!**: Instalado con el paquete `ros-melodic-moveit`, permite la planificación y ejecución de movimientos complejos para el robot.
- **Interfaz de Transmisión (transmission-interface)**: Facilita el uso de controladores que integran motores y transmisiones mecánicas en el robot.
- **Xacro**: Herramienta para procesar archivos URDF, muy útil para simplificar la configuración del modelo del robot.
- **Joint State Publisher y GUI**: Muestra los estados de las articulaciones del robot, útil para la visualización en RViz.

Con todos estos paquetes, tendrás las herramientas necesarias para una configuración completa de ROS, incluyendo simulación en Gazebo, planificación en MoveIt!, y visualización en RViz.

Comandos Básicos de ROS

Algunos comandos clave en ROS incluyen:

- **roscore**: Inicia el núcleo de ROS, necesario antes de ejecutar otros nodos:

```
roscore
```

- **roslaunch**: Lanza nodos y configuraciones desde un archivo `launch`:

```
roslaunch <nombre_paquete> <archivo.launch>
```

- **rosrun**: Ejecuta un nodo específico de un paquete:

```
rosrun <nombre_paquete> <nombre_nodo>
```

Verificación de la Configuración

Para verificar que los nodos y temas están activos:

```
rosversion  
rostopic list
```

Estos comandos permiten monitorear el estado del sistema ROS y verificar que los nodos están ejecutándose correctamente.

3.2. Configuración y Ejecución de la Simulación en Gazebo

La configuración y ejecución de simulaciones en Gazebo sigue una estructura básica que permite modelar robots, definir entornos, y probar movimientos en un entorno seguro. Esta sección describe los pasos esenciales para configurar y ejecutar el robot en Gazebo, en particular para el brazo robótico Staubli TX-90.

Configuración del Modelo del Robot

Para simular el robot en Gazebo, es necesario definir su modelo, lo cual se realiza generalmente en un archivo URDF (*Unified Robot Description Format*) o Xacro. Este archivo describe la estructura física del robot, incluyendo:

- **Estructura del Robot:** Definición de cada enlace y articulación, lo que determina el alcance y la movilidad de cada parte del robot.
- **Propiedades Físicas:** Especificación de materiales, peso y tamaño, lo cual afecta las dinámicas durante la simulación.
- **Plugins de Control:** Añadir plugins que permitan la comunicación entre Gazebo y ROS para controlar el robot en tiempo real. Esto incluye los plugins de control de articulaciones, que facilitan el movimiento del robot dentro de la simulación.

Definición del Entorno de Simulación

En Gazebo, además del modelo del robot, se define el entorno en el que operará. Esto implica configurar un archivo de mundo que incluye:

- **Objetos y Obstáculos:** Modelos adicionales en el entorno, como mesas, paredes u otros elementos, que interactúan con el robot.
- **Superficies y Terreno:** Configuración del tipo de suelo y las propiedades de fricción, lo que afecta la dinámica del robot.
- **Luces y Sensores Virtuales:** Elementos que mejoran la precisión de la simulación y permiten probar sensores integrados en el robot.

Ejecución de la Simulación en Gazebo

Una vez que el modelo del robot y el entorno están configurados, la simulación se puede iniciar mediante ROS. Los pasos básicos para ejecutar la simulación incluyen:

1. **Lanzamiento de Gazebo y el Robot en ROS:** Usar archivos de lanzamiento (*launch files*) de ROS que integren el modelo del robot y el entorno en Gazebo. Estos archivos permiten cargar la simulación con un solo comando.
2. **Control del Robot:** Con los controladores de ROS configurados, se pueden enviar comandos al robot para que ejecute movimientos. Esto puede hacerse mediante scripts en Python o utilizando MoveIt para planificar y ejecutar trayectorias de manera segura.

3. **Monitoreo en Tiempo Real:** Durante la simulación, Gazebo ofrece una visualización en tiempo real, lo que permite observar el movimiento del robot y verificar la precisión de los controles. Esta visualización ayuda a identificar ajustes necesarios en los parámetros del robot o del entorno.

Validación y Ajustes

Una vez ejecutada la simulación, es importante revisar y ajustar parámetros según los resultados obtenidos:

- **Ajustes de Precisión:** Verificar que el robot alcance los puntos deseados con precisión y sin colisiones, ajustando las configuraciones si es necesario.
- **Optimización de Trayectorias:** Analizar las trayectorias para asegurar que sean eficientes y seguras.
- **Corrección de Errores:** Registrar cualquier anomalía durante la simulación y realizar cambios en el modelo o en los comandos para mejorar la confiabilidad del sistema.

Estos pasos básicos permiten una configuración y ejecución efectiva en Gazebo, asegurando que el robot funcione correctamente en la simulación antes de llevarlo a un entorno físico. Este proceso de simulación es fundamental para reducir riesgos y optimizar el comportamiento del robot.

3.3. Configuración de MoveIt y Ejecución en Gazebo

Para configurar MoveIt, es necesario pasar por varias secciones en el *Setup Assistant* que permiten cargar el modelo del robot, configurar los grupos de planificación, crear los controladores y generar el paquete de configuración. A continuación, se detallan cada uno de estos pasos.

Carga del URDF

El primer paso en el Setup Assistant de MoveIt es cargar el modelo del robot en formato URDF o Xacro. Este archivo contiene la descripción del robot, incluyendo sus enlaces, articulaciones y geometría. Es fundamental asegurarse de que el modelo URDF esté correctamente configurado, ya que representa la estructura física y cinemática del robot.

Creación de Virtual Joint

El *Virtual Joint* permite vincular el robot a un marco de referencia externo, lo cual es útil para robots móviles o montados en una base móvil. Este tipo de articulación virtual se utiliza para definir el movimiento del robot en relación al entorno, permitiendo que la planificación tenga en cuenta el desplazamiento de la base o del marco de referencia.

- **Tipo de Joint:** Generalmente, se selecciona el tipo `floating` o `planar` dependiendo de los grados de libertad requeridos.
- **Parent Frame:** Define el marco de referencia en el que se mueve el robot.

Generación de la Matriz de Colisiones

En la sección de *Self-Collisions*, MoveIt genera una matriz de colisiones interna que permite optimizar la planificación evitando colisiones innecesarias entre partes del propio robot. Esta configuración asegura que MoveIt considere solo las colisiones relevantes, mejorando la eficiencia de la planificación de movimientos.

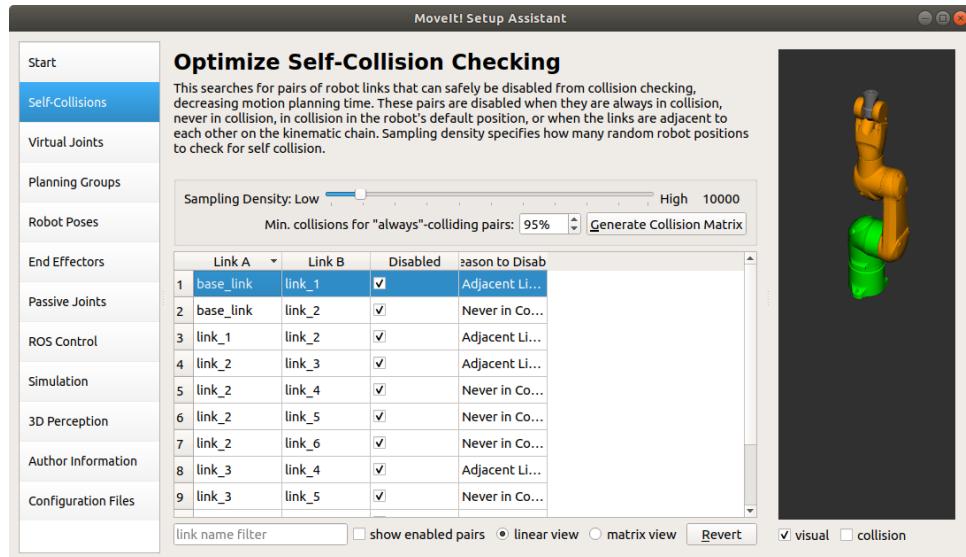


Figura 3.1: Matriz de Colisiones en el Setup Assistant de MoveIt.

Creación del Planning Group

El *Planning Group* define un conjunto de articulaciones y enlaces que se utilizarán para planificar movimientos en el robot. Para un brazo robótico, esto generalmente incluye todas las articulaciones necesarias para mover el efecto final. En esta sección también es importante seleccionar el solucionador de cinemática, como *KDL Kinematics*, que permite resolver problemas de cinemática inversa.

- **Selección del Solucionador de Cinemática:** Seleccionar un solucionador adecuado, como KDL, es crucial para resolver la cinemática inversa con precisión.
- **Asignación de Articulaciones:** Se deben añadir las articulaciones involucradas en el movimiento del efecto final.

Robot Poses

En la sección de *Robot Poses*, se definen poses predeterminadas que el robot puede alcanzar, como posiciones de reposo o de inicio. Estas poses facilitan la planificación de movimientos al permitir que el robot tenga puntos de referencia definidos.

End Effectors

El *End Effector* se configura para definir la herramienta o pinza en el extremo del brazo robótico. Esto permite a MoveIt realizar la planificación teniendo en cuenta el efecto final y los

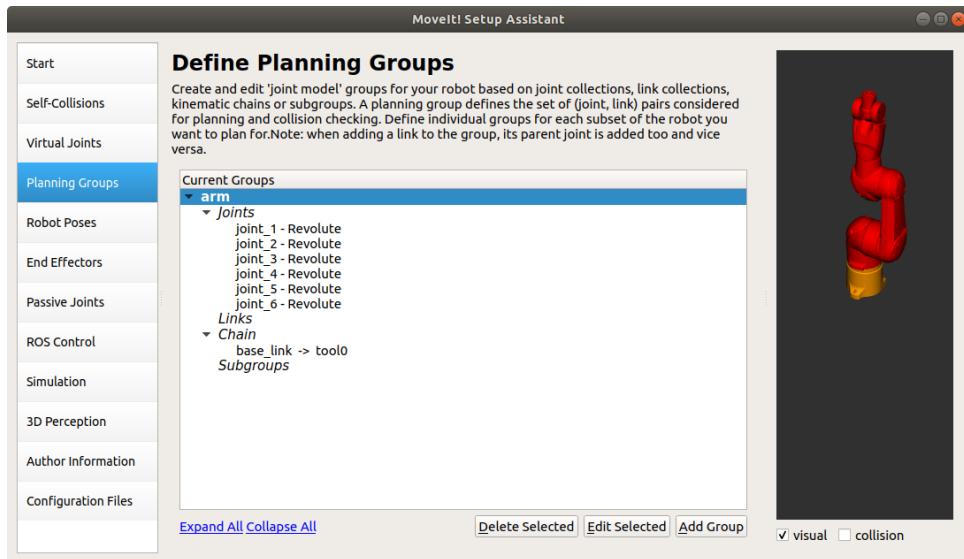


Figura 3.2: Definición de Planning Group en el Setup Assistant de MoveIt.

objetos que debe manipular. En esta sección se especifica el enlace final y su posición relativa al último enlace del brazo.

Passive Joints

Las *Passive Joints* son articulaciones que forman parte del modelo, pero no son controladas directamente. Estas articulaciones no participan activamente en la planificación, lo que optimiza el rendimiento.

ROS Control y Creación de Controladores

En la sección de *ROS Control*, se configuran los controladores para gestionar las articulaciones del robot mediante ROS. Aquí, se añade un controlador de tipo `FollowJointTrajectory` que permite el control de trayectoria de las articulaciones.

- **Tipo de Controlador:** `FollowJointTrajectory` es el tipo de controlador que permite enviar trayectorias de movimiento a las articulaciones.
- **Asignación de Articulaciones:** Se debe asociar cada articulación del robot al controlador.

Generación del Paquete de Configuración

Una vez completadas todas las configuraciones en el *Setup Assistant*, se puede generar el paquete de configuración de MoveIt. Este paquete contiene los archivos necesarios para lanzar MoveIt junto con el modelo del robot y los parámetros configurados. Para generar el paquete, selecciona la opción *Configuration Files* en el menú del *Setup Assistant* y guarda los archivos en un paquete de ROS.

Este paquete de configuración permitirá cargar y controlar el robot en RViz y Gazebo, facilitando la planificación y ejecución de trayectorias.

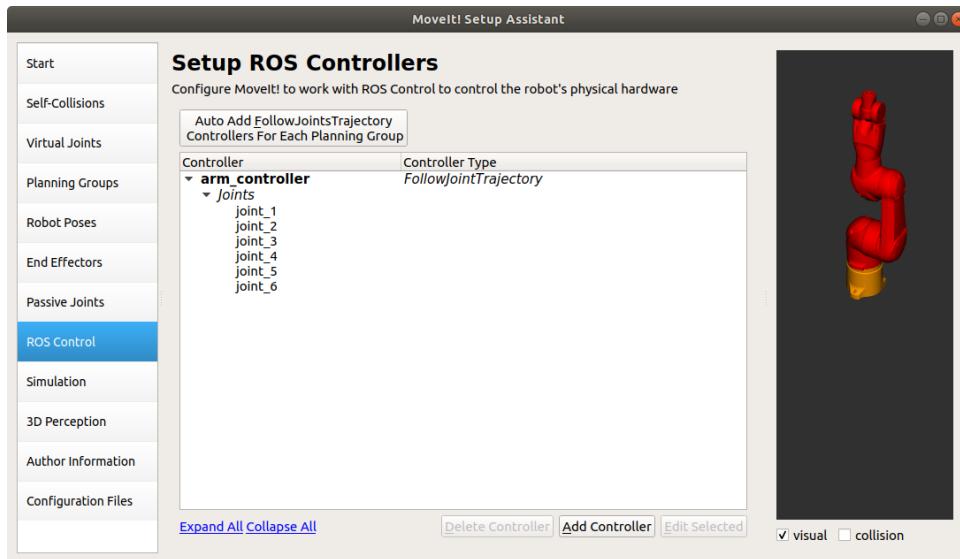


Figura 3.3: Configuración de Controladores en el Setup Assistant de MoveIt.

Ejecución de la Simulación en Gazebo y MoveIt

Con el paquete de configuración generado, existen diferentes opciones de lanzamiento dependiendo de si deseas ejecutar solo Gazebo, solo MoveIt en RViz, o ambos simultáneamente.

- **Ejecutar solo Gazebo:** Para lanzar únicamente la simulación en Gazebo con el modelo del robot, utiliza el siguiente comando en la terminal:

```
roslaunch <nombre_paquete> gazebo.launch
```

Este archivo de lanzamiento (`gazebo.launch`) cargará Gazebo con el modelo del robot, permitiéndote interactuar con el robot en el entorno simulado sin MoveIt.

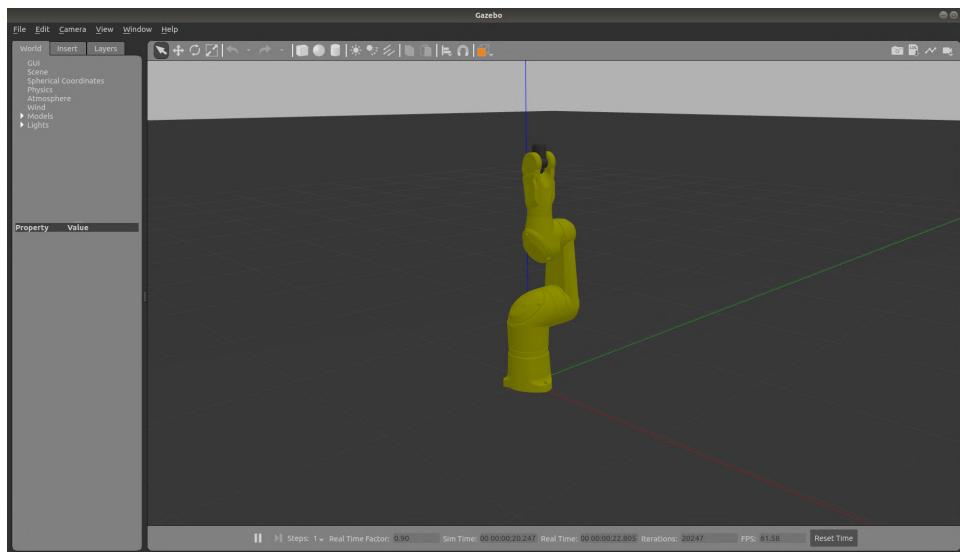


Figura 3.4: Simulación del robot en Gazebo.

- **Ejecutar solo MoveIt en RViz:** Para lanzar MoveIt en RViz, que permite planificar y visualizar los movimientos del robot sin cargar Gazebo, utiliza el siguiente comando:

```
roslaunch <nombre_paquete> demo.launch
```

Este archivo de lanzamiento (`demo.launch`) abrirá RViz con MoveIt, proporcionando una interfaz para planificar y verificar las trayectorias en tiempo real.

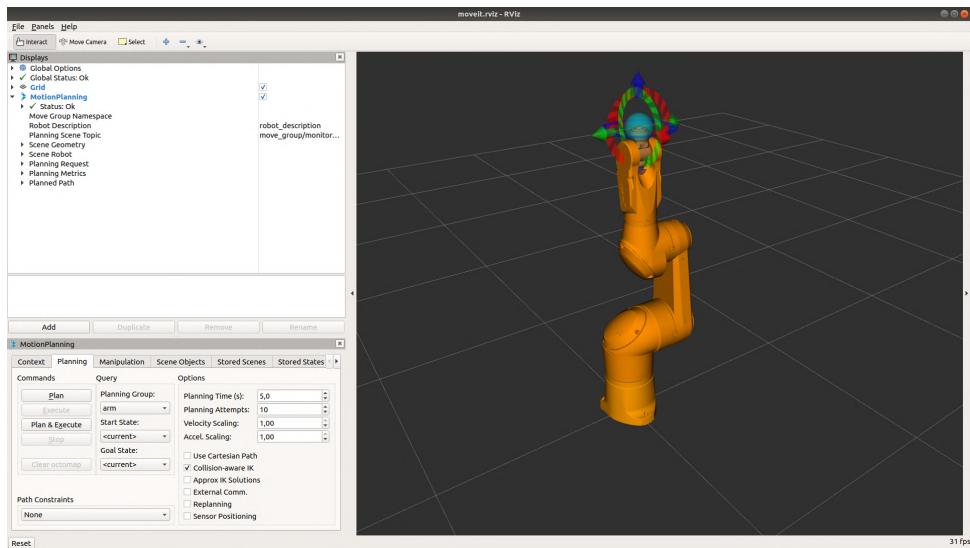


Figura 3.5: Interfaz de MoveIt en RViz para la planificación de movimientos.

- **Ejecutar Gazebo y MoveIt en RViz Simultáneamente:** Para ejecutar ambos Gazebo y MoveIt en RViz al mismo tiempo, utiliza el siguiente comando:

```
roslaunch <nombre_paquete> demo_gazebo.launch
```

Este archivo de lanzamiento (`demo_gazebo.launch`) cargará Gazebo con el modelo del robot y abrirá la interfaz de MoveIt en RViz, permitiendo planificar y ejecutar movimientos en el entorno de simulación de Gazebo. Esta opción es ideal para una integración completa, donde los movimientos planificados en RViz se pueden visualizar y ejecutar en el entorno simulado.

Opciones Básicas en RViz para MoveIt

RViz proporciona una serie de herramientas que facilitan la planificación y ejecución de movimientos en el robot:

- **Planificación de Movimientos:** En la sección de *MotionPlanning* en RViz, selecciona el grupo de planificación (por ejemplo, `arm`) y define el objetivo de posición y orientación del efecto final. Luego, presiona el botón `Plan` para que MoveIt calcule una trayectoria.
- **Ejecución de la Trayectoria:** Una vez calculada la trayectoria, puedes visualizarla en RViz. Si estás satisfecho con el plan, presiona el botón `Plan & Execute` para ejecutar el movimiento en el robot simulado en Gazebo.

- **Selección del Algoritmo de Planificación:** En la pestaña de *Context* dentro de la configuración de *OMPL* en *RViz*, puedes seleccionar el algoritmo de planificación adecuado, como *RRTConnect*. Este algoritmo, basado en el método de *Rapidly-exploring Random Tree*, es especialmente útil para trayectorias rápidas y eficientes, optimizando el movimiento, evitando colisiones y mejorando la eficiencia en la ejecución. La elección del algoritmo dependerá de los requisitos específicos de precisión y velocidad de la tarea.
- **Configuración de Parámetros de Movimiento:** También puedes ajustar los parámetros de velocidad y aceleración, lo cual es útil para personalizar el comportamiento del robot según la tarea específica. En *RViz*, puedes configurar la escala de velocidad y aceleración en las opciones de *MotionPlanning*.

Estas herramientas permiten planificar, verificar y ejecutar trayectorias de manera segura, optimizando el control del robot y asegurando que los movimientos sean precisos y seguros antes de implementarlos en un entorno físico.

Implementación

4.1. Soluciones Software Desarrolladas

4.1.1. Preparación del Espacio de Trabajo

Para integrar el brazo robótico Staubli en el entorno de ROS, primero se preparó el espacio de trabajo. Este proceso incluyó los siguientes pasos:

1. **Creación del espacio de trabajo:** Se creó un espacio de trabajo de ROS donde se organizarán todos los paquetes y dependencias necesarias.
2. **Clonación de repositorios:** En la carpeta `src` del espacio de trabajo, se clonaron los siguientes repositorios:
 - `ros-industrial/staubli`: Este repositorio contiene el paquete `staubli_resources`, que proporciona recursos y configuraciones básicas para trabajar con robots Staubli en ROS.
 - `ros-industrial/staubli_experimental`: Este repositorio incluye modelos experimentales de robots Staubli, como el TX-90 y el TX2-90, junto con archivos de configuración adicionales necesarios para la simulación en Gazebo.
3. **Limpieza de archivos:** Para simplificar la estructura del proyecto, se eliminaron archivos y carpetas de modelos de robots que no se utilizarán, conservando solo:
 - La carpeta `staubli/staubli_resources`
 - Las carpetas `staubli_tx90_gazebo`, `staubli_tx90_support` y `staubli_tx2_90_support`
4. **Compilación del espacio de trabajo:** Finalmente, el espacio de trabajo se compila con el comando `catkin_make` para construir los paquetes y dependencias.

4.1.2. Solución para Simulación

En esta sección se detalla la configuración de MoveIt para el uso del simulador Gazebo con el modelo del TX-90.

1. **Generación del archivo URDF:** Primero, se genera el archivo URDF del robot TX-90 a partir de su archivo `xacro` con el siguiente comando:

```
rosrun xacro xacro staubli_tx90_gazebo/urdf/tx90.xacro > tx90.urdf
```

2. **Configuración con el Asistente de MoveIt:** Usando el Setup Assistant de MoveIt, se carga el archivo URDF en la opción de crear un nuevo paquete.
 - **Matriz de colisiones:** En la pestaña de *Self-Collisions*, se crea la matriz de colisiones para evitar que el robot choque consigo mismo.
 - **Grupos de planificación:** En *Planning Groups*, se crea un grupo llamado **arm**. Se selecciona **kdl_kinematics_plugin/KDLKinematicsPlugin** como el *solver* cinemático y **RRTConnect** como el planificador predeterminado. Se añaden las juntas del brazo desde **joint_1** hasta **joint_6**.
 - **Controladores ROS:** En *ROS Control*, se crea un controlador llamado **arm_controller** de tipo **FollowJointTrajectory**, asignando el grupo de movimiento **arm**.
3. **Generación del paquete de configuración:** El paquete de configuración de MoveIt se genera y se guarda bajo el nombre **staubli_tx90_moveit_config**.

Para que MoveIt funcione correctamente, se realizaron varios ajustes adicionales en los archivos de configuración:

- **Archivo moveit_controllers.yaml:** Este archivo se creó en la carpeta **config** y define el controlador del brazo. Su contenido es:

```
controller_list:
  - name: arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - joint_1
      - joint_2
      - joint_3
      - joint_4
      - joint_5
      - joint_6
```

- **Archivo staubli_tx90_gazebo_moveit_controller_manager.launch.xml:** Se incluye el archivo anterior de esta manera:

```
<launch>
  <!-- loads controller list to the param server -->
  <rosparam file="$(find staubli_tx90_moveit_config)
    /config/ros_controllers.yaml"/>
  <rosparam file="$(find staubli_tx90_moveit_config)
    /config/moveit_controllers.yaml" command="load"/>

  <!-- Define the controller manager plugin to use for trajectory
  execution -->
  <param name="moveit_controller_manager"
    value="moveit_simple_controller_manager/
    MoveItSimpleControllerManager"/>
</launch>
```

- **Archivo ros_controllers.launch:** En este archivo, se agrega el siguiente nodo para cargar los controladores:

```
<node name="controller_spawner" pkg="controller_manager" type="spawner"
      respawn="false" output="screen" args="joint_state_controller
      arm_controller" />
```

- **Archivo ros_controllers.yaml:** Contiene la configuración del controlador, con al menos lo siguiente:

```
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50
controller_list:
  [arm_controller]
arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:
    - joint_1
    - joint_2
    - joint_3
    - joint_4
    - joint_5
    - joint_6
```

Con esta configuración, MoveIt podrá planificar trayectorias en RViz y comunicarse con Gazebo para simular los movimientos del robot.

4.1.3. Solución para el Caso Real

Para el brazo físico TX2-90, el proceso de configuración es similar, pero requiere el **driver Val3** para la comunicación con el controlador CS9 del robot.

1. **Clonación del driver Val3:** En la carpeta **src**, se clona el repositorio de Val3:

```
https://github.com/ros-industrial/staubli\_val3\_driver
```

2. **Configuración del paquete MoveIt para TX2-90:**

- Se genera el archivo URDF a partir de **tx2_90.xacro** en la carpeta **staubli_tx2_90_support/urdf** con:

```
rosrun xacro xacro tx2_90.xacro > tx2_90.urdf
```

- El URDF se carga en el Asistente de MoveIt, generando la matriz de colisiones, el grupo de planificación como en el caso anterior, pero sin crear el controlador ya que utilizará el controlador real que proporcionará val3 al establecer conexión.

3. **Ajustes en los archivos de configuración:**

- **moveit_controllers.yaml**: Este archivo se configura para definir el controlador de seguimiento de trayectorias. El contenido es el siguiente:

```
controller_list:  
  - name: joint_trajectory_action  
    action_ns: ""  
    type: FollowJointTrajectory  
    joints:  
      - joint_1  
      - joint_2  
      - joint_3  
      - joint_4  
      - joint_5  
      - joint_6  
    action_monitoring_timeout: 15.0
```

Este archivo le permite a MoveIt reconocer el controlador de seguimiento de trayectorias del brazo, asegurando que el sistema esté listo para ejecutar movimientos planificados en el hardware.

- **ros_controllers.yaml**: Este archivo contiene configuraciones avanzadas del controlador, necesarias para la interacción del hardware y el control del robot. La configuración debe incluir:

```
generic_hw_control_loop:  
  loop_hz: 300  
  cycle_time_error_threshold: 0.01  
  
hardware_interface:  
  joints:  
    - joint_1  
    - joint_2  
    - joint_3  
    - joint_4  
    - joint_5  
    - joint_6  
  sim_control_mode: 0 # 0: posición, 1: velocidad  
  
joint_state_controller:  
  type: joint_state_controller/JointStateController  
  publish_rate: 100  
  
controller_list:  
  [joint_trajectory_action]
```

En esta configuración:

- **loop_hz** establece la frecuencia de actualización del bucle de control en 300 Hz para mejorar la capacidad de respuesta.
- **sim_control_mode** se fija en 0 para trabajar en modo de posición, adecuado para control directo de las juntas del robot.
- **joint_state_controller** controla el estado de las juntas, publicando los datos a 100 Hz para asegurar una actualización fluida en tiempo real.

- `demo.launch`: Se realizan varios ajustes en este archivo de lanzamiento:

- Establecer `fake_execution` en `false` con el siguiente argumento:

```
<arg name="fake_execution" value="false"/>
```

Este ajuste asegura que MoveIt utilice datos reales de estado del robot en lugar de datos simulados, adecuados solo para pruebas en entornos virtuales.

- Modificar el parámetro `source_list` en el nodo `joint_state_publisher` para usar los estados de las juntas reales:

```
<rosparam param="source_list">[joint_states]</rosparam>
```

Esto permite que el sistema reciba el estado actual de las juntas del brazo robótico real en lugar de usar valores generados por el controlador `fake`.

- Cargar `moveit_controllers.yaml` en el servidor de parámetros con:

```
<rosparam command="load"
  file="$(find staubli_tx2_90_moveit_config
  /config/moveit_controllers.yaml" />
```

Este comando asegura que MoveIt tenga acceso a la configuración del controlador de seguimiento de trayectorias definido anteriormente.

- `trajectory_execution.launch.xml`: En este archivo, se ajusta el parámetro `trajectory_execution/allowed_execution_duration_scaling` a un valor de 2.5:

```
<param name="trajectory_execution/
  allowed_execution_duration_scaling" value="2.5"/>
```

Este parámetro aumenta el tiempo máximo permitido para la ejecución de trayectorias, al multiplicarlo por 2.5 respecto del tiempo estimado. Esto fue necesario porque el tiempo límite inicial provocaba errores de `timeout` en la ejecución, y con el ajuste el sistema tolera ligeras variaciones en el tiempo de ejecución sin interrumpir el movimiento.

Con estas configuraciones, MoveIt puede comunicarse eficazmente con el controlador CS9 a través del driver Val3, permitiendo el control del brazo real para aplicaciones industriales.

4.2. Arquitectura de la Solución

4.2.1. Estructura del software

La estructura del proyecto está organizada en distintas carpetas para separar las configuraciones y archivos necesarios para la simulación, el soporte del brazo real y la planificación de movimientos en MoveIt. Esta organización facilita el desarrollo y la gestión de las configuraciones para los diferentes entornos y herramientas utilizadas.

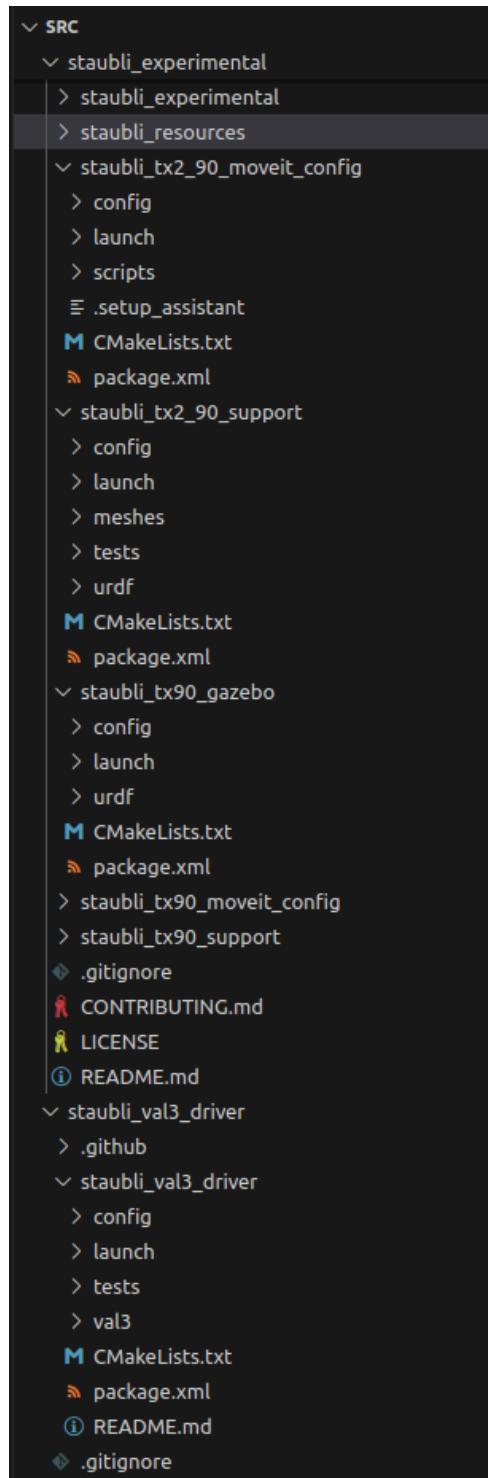


Figura 4.1: Estructura de carpetas del proyecto

A continuación, se describen las carpetas principales y su contenido:

- **staubli_tx90_gazebo**: Esta carpeta contiene configuraciones para la simulación en Gazebo del brazo Staubli, incluyendo los archivos necesarios para modelar el robot en un entorno virtual, con propiedades físicas y dinámicas específicas.
- **staubli_tx90_support** y **staubli_tx2_90_support**: Incluyen las configuraciones de soporte para el uso del brazo en un entorno real, como las descripciones del robot, mallas y

configuraciones específicas para operar el brazo físico.

- **staubli_tx90_moveit_config** y **staubli_tx2_90_moveit_config**: Estas carpetas contienen los archivos de configuración de MoveIt, que incluyen parámetros para la planificación de trayectorias, detección de colisiones, y los controladores necesarios para trabajar tanto en simulación como en el entorno real.
- **staubli_val3_driver**: Almacena el driver Val3, que se encarga de establecer la comunicación entre ROS y el controlador CS9 del brazo Staubli para el entorno real.

Las carpetas **config** y **launch**, presentes en las configuraciones mencionadas, son especialmente importantes:

- **config**: Contiene los archivos de configuración esenciales, como `moveit_controllers.yaml` en MoveIt, que define los controladores para operar el brazo, tanto en simulación como en el entorno real.
- **launch**: Almacena los archivos de lanzamiento (*launch files*) para inicializar los nodos y configurar los parámetros de ROS automáticamente. Estos archivos permiten lanzar la simulación en Gazebo o controlar el brazo físico mediante el controlador CS9 sin necesidad de ejecutar cada nodo de forma manual.

4.2.2. Estructura de Nodos y Tópicos en la Simulación de ROS

En esta sección se presenta la estructura de nodos y tópicos que se utilizan en la simulación, en un entorno que integra Gazebo como simulador y MoveIt como planificador de trayectorias. Las imágenes a continuación muestran la organización de nodos y la comunicación entre ellos a través de tópicos.

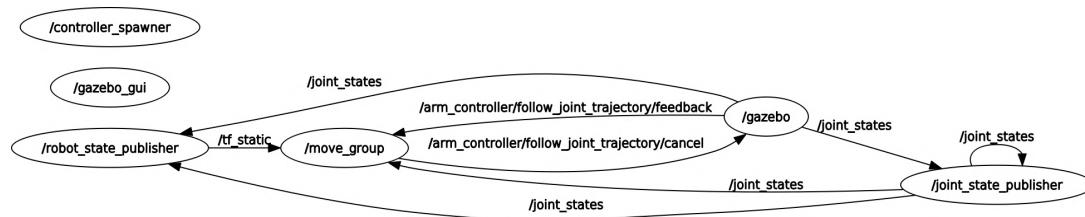


Figura 4.2: Estructura de nodos en la simulación de ROS

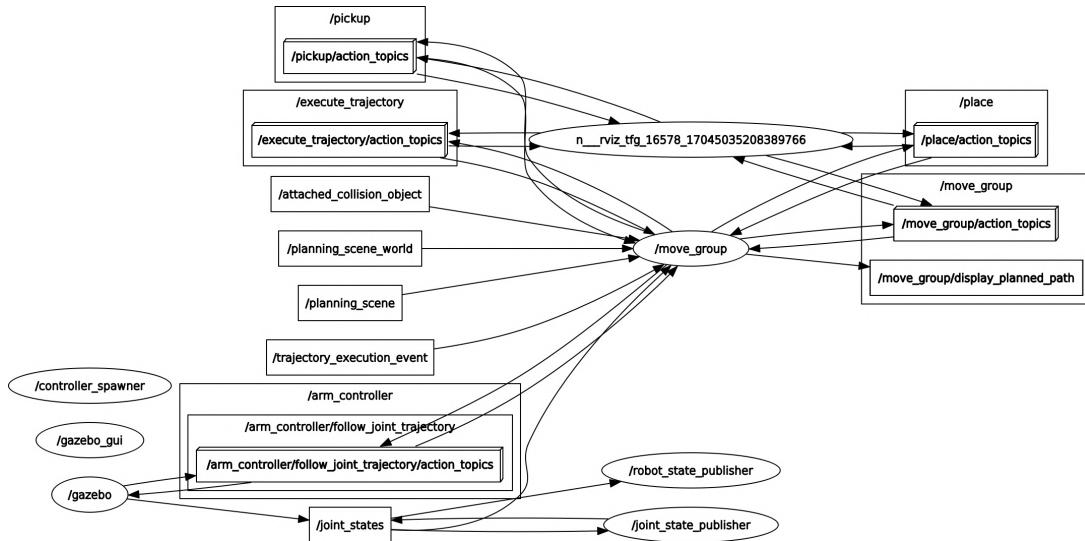


Figura 4.3: Estructura de nodos y tópicos en la simulación de ROS

Estas imágenes fueron generadas al ejecutar el archivo `demo_gazebo.launch`, configurado para lanzar todos los nodos y parámetros necesarios para realizar una simulación completa. Este archivo de lanzamiento permite probar la planificación y ejecución de una trayectoria en un brazo robótico simulado en Gazebo, utilizando MoveIt para calcular las trayectorias y controlar los movimientos del robot.

Estructura de Nodos en la Simulación de ROS

En la simulación, los nodos de ROS están organizados para coordinar la planificación y ejecución de trayectorias en un entorno virtual. A continuación, se describen los principales nodos y sus funciones dentro de la arquitectura del sistema:

- **/move_group**: Nodo central de MoveIt encargado de planificar y ejecutar movimientos. Proporciona una interfaz para definir trayectorias y monitorear la planificación.
- **/gazebo** y **/gazebo_gui**: Nodos fundamentales para la simulación en Gazebo. **/gazebo** gestiona la física y la simulación del entorno, mientras que **/gazebo_gui** proporciona la interfaz gráfica para visualizar el entorno simulado.
- **/controller_spawner**: Responsable de lanzar y gestionar los controladores de las articulaciones del brazo en la simulación. En este caso, inicia el **arm_controller**, un controlador de trayectoria para ejecutar movimientos.
- **/arm_controller**: Actúa como el controlador de trayectoria del brazo robótico, recibiendo comandos de posición y ejecutándolos mediante los tópicos que controlan las articulaciones individuales.
- **/robot_state_publisher** y **/joint_state_publisher**: Publican el estado actual del robot en la simulación, incluidos los ángulos y posiciones de las articulaciones. **/joint_state_publisher** publica en tiempo real los estados de las juntas, mientras que **/robot_state_publisher** transmite la posición global del robot.

Estructura de Tópicos y Comunicación en la Simulación

En ROS, la comunicación entre nodos se realiza a través de tópicos. A continuación se detallan los principales tópicos utilizados en la simulación:

- **/joint_states**: Publicado por **/joint_state_publisher**, contiene información en tiempo real sobre el estado de cada articulación del brazo. Los nodos **/move_group** y **/robot_state_publisher** se suscriben a este tópico para actualizar el estado del robot.
- **/move_group/action_topics** y **/move_group/display_planned_path**: Utilizados por **/move_group** para coordinar la planificación y visualización de movimientos. **/display_planned_path** permite ver la trayectoria planificada antes de su ejecución.
- **/execute_trajectory** y **/execute_trajectory/action_topics**: Gestionan la ejecución de la trayectoria planificada en el robot. **/move_group** publica las trayectorias en estos tópicos para que **/arm_controller** las ejecute en la simulación.
- **/arm_controller/follow_joint_trajectory**: Gestionado por **/arm_controller**, permite ejecutar las trayectorias calculadas por MoveIt en el simulador. Controla directamente las articulaciones a través de este tópico.
- **/planning_scene_world** y **/planning_scene**: Publicados y suscritos por **/move_group** para gestionar la planificación del entorno y detectar colisiones, permitiendo ajustar trayectorias en función de los obstáculos.

En conjunto, estos nodos y tópicos aseguran que la planificación y ejecución de las trayectorias del brazo robótico se realicen de manera precisa. La configuración del sistema en Gazebo y MoveIt permite validar y ajustar los movimientos en un entorno virtual antes de su implementación en un robot físico, proporcionando una herramienta de simulación valiosa.

4.2.3. Servicios Disponibles en la Simulación

La simulación utiliza varios servicios en ROS que permiten la interacción entre distintos componentes, como **Gazebo**, **MoveIt!**, y el sistema de control del robot. A continuación, se describen algunos de los servicios más relevantes que facilitan la comunicación y la gestión del entorno de simulación:

Servicios de Gazebo Los servicios proporcionados por Gazebo permiten modificar el estado del mundo simulado y controlar propiedades físicas. Algunos servicios clave incluyen:

- **/gazebo/apply_body_wrench** y **/gazebo/apply_joint_effort**: Aplican una fuerza o un torque a un cuerpo o a una articulación específica en la simulación, lo que permite pruebas de comportamiento bajo condiciones dinámicas.
- **/gazebo/get_model_state** y **/gazebo/set_model_state**: Recuperan y establecen el estado (posición y orientación) de los modelos en la simulación, lo cual es esencial para monitorizar el estado del robot o cambiar su configuración inicial.
- **/gazebo/spawn_urdf_model** y **/gazebo/delete_model**: Permiten añadir o eliminar modelos URDF (Unified Robot Description Format) en el entorno de simulación. Estos servicios son útiles para la configuración inicial del robot y el entorno.

- **/gazebo/pause_physics** y **/gazebo/unpause_physics**: Pausan y reanudan la simulación física, lo que permite realizar ajustes en el entorno o en el robot sin la interferencia de la física en tiempo real.
- **/gazebo/reset_simulation** y **/gazebo/reset_world**: Restablecen el estado completo de la simulación o únicamente el mundo, devolviendo el entorno a su configuración original.

Servicios de MoveIt! Los servicios de MoveIt! habilitan el planeamiento y ejecución de trayectorias en el robot, además de la validación de movimientos en el espacio de trabajo simulado:

- **/compute_ik** y **/compute_fk**: Realizan cálculos de cinemática inversa y directa, respectivamente, lo cual es fundamental para determinar la posición y la orientación del extremo del robot en función de sus ángulos articulares.
- **/check_state_validity**: Verifica la validez de un estado dado, identificando si la posición planificada del robot podría causar una colisión o infringir restricciones.
- **/plan_kinematic_path**: Planifica una trayectoria cinemática entre dos posiciones del robot, asegurando que se eviten obstáculos y que se sigan las restricciones especificadas.

Servicios de Gestión y Monitoreo de Controladores El controlador del robot y sus componentes pueden gestionarse mediante servicios que permiten su activación, desactivación y monitoreo:

- **/controller_manager/list_controllers** y **/controller_manager/switch_controller**: Listan y alternan entre distintos controladores en la simulación, lo que permite probar diferentes enfoques de control en el entorno simulado.
- **/controller_manager/load_controller** y **/controller_manager/unload_controller**: Cargan y descargan controladores específicos, facilitando una gestión modular de los controladores y su configuración.

Servicios Generales de ROS Algunos servicios adicionales permiten la configuración y ajuste de parámetros globales y la configuración de escenas en el entorno de planeación:

- **/clear_octomap**: Limpia el mapa octomapa generado por sensores, lo cual es útil en caso de cambios en el entorno de simulación.
- **/get_planning_scene** y **/apply_planning_scene**: Permiten obtener y aplicar configuraciones de escenas de planeación, posibilitando cambios en el entorno de trabajo en tiempo real.

Estos servicios juegan un papel fundamental en el control del entorno de simulación, en la configuración de modelos y en la planificación de movimientos. Su uso adecuado permite realizar una simulación precisa y realista, asegurando que los movimientos del robot sean ejecutados de acuerdo con las condiciones y restricciones establecidas en el entorno simulado.

Pruebas y Validaciones

En esta sección se presentan tres pruebas clave que se realizarán para validar el correcto funcionamiento del sistema de control del brazo robótico, evaluando su desempeño en distintos contextos y condiciones. Cada una de estas pruebas responde a preguntas específicas relacionadas con la comunicación, planificación y efectividad de los algoritmos involucrados en el proyecto.

La primera prueba, denominada **Prueba en Simulación**, tiene como objetivo verificar si existe una comunicación fluida entre el modelo simulado del brazo en Gazebo y el planificador de trayectorias MoveIt, a través de ROS. La pregunta principal que responde esta prueba es: ¿Es posible enviar y ejecutar correctamente una serie de puntos de trayectoria en un entorno simulado? Para ello, se ejecutará un script en Python que envía al planificador una secuencia de puntos, y se observará si la simulación es capaz de recibir y ejecutar esta trayectoria de manera adecuada.

La segunda prueba, llamada **Prueba con Brazo Robótico Físico en un Entorno Realista**, se realiza en un entorno real, donde el brazo robótico físico reproduce una trayectoria específica. En este caso, la pregunta que se intenta responder es: ¿El brazo robótico físico puede ejecutar la planificación en un entorno real sin problemas de comunicación o precisión? El procedimiento consiste en ejecutar un script en Python que contiene los puntos de los lados de un cuadrado, de modo que el brazo, utilizando un rotulador, dibuje dicha figura en una hoja de papel. Esto permite validar la precisión y capacidad de control en condiciones físicas reales.

Finalmente, la tercera prueba, denominada **Comparación de Algoritmos**, evalúa el desempeño de distintos algoritmos de planificación en el mismo entorno simulado. La pregunta clave en esta prueba es: ¿Cuál de los algoritmos evaluados ofrece el mejor rendimiento en términos de eficiencia y precisión en la planificación de trayectorias para este proyecto? Para responderla, se realizarán 10 ejecuciones de planificación para una misma trayectoria y bajo las mismas condiciones, comparando los resultados para identificar las diferencias y semejanzas en su desempeño. Esto permitirá analizar cuál de los algoritmos es más adecuado para los requerimientos específicos del sistema.

A continuación, cada una de estas pruebas se describe en detalle, presentando sus resultados y discutiendo sus implicancias.

5.1. Prueba en Simulación

En esta prueba en simulación, se utilizó un script en Python para controlar un brazo robótico mediante el uso de MoveIt! y ROS. La finalidad del script es planificar y ejecutar una serie de movimientos a través de waypoints (puntos intermedios) definidos en el espacio tridimensional. La prueba consiste en que, en función de una serie de waypoints con diferentes coordenadas y diferentes orientaciones, se calcule una trayectoria y se ejecute. Es importante que estos puntos sean válidos y alcanzables por el brazo robótico. Cada waypoint se define en términos de las

coordenadas X , Y , Z y la orientación del brazo en el espacio, que se representa mediante un cuaternion calculado a partir de los ángulos de Euler.

Los ángulos de Euler son una forma de representar la orientación de un objeto en el espacio tridimensional mediante tres ángulos, que describen las rotaciones alrededor de los ejes X , Y y Z . Estos ángulos son importantes para asegurar que el robot pueda realizar movimientos en las direcciones y orientaciones deseadas sin colisiones ni movimientos inadecuados. [36]

Sin embargo, los ángulos de Euler tienen limitaciones, especialmente cuando se trata de representar rotaciones complejas en tres dimensiones. Uno de los principales problemas con los ángulos de Euler es el fenómeno conocido como "bloqueo de gimbal", que ocurre cuando dos de los tres ejes de rotación se alinean, lo que provoca una pérdida de grados de libertad en la rotación. Esto puede generar distorsiones o movimientos no deseados en el robot, lo cual no ocurre al usar cuaterniones. [37]

Los cuaterniones son una extensión de los números complejos y permiten representar rotaciones de manera más eficiente y estable. A diferencia de los ángulos de Euler, los cuaterniones no sufren de bloqueo de gimbal y proporcionan una interpolación suave entre las rotaciones, lo que facilita el control preciso del brazo robótico. Además, los cuaterniones requieren menos cálculos para combinar varias rotaciones, lo que los hace más rápidos y precisos para el control de robots en tiempo real. Por estas razones, en este caso, se utiliza un cuaternion para representar la orientación de cada waypoint en lugar de los ángulos de Euler.

A continuación, se explica cada parte del código y su función dentro de la prueba. El código completo se encuentra disponible en mi repositorio de GitHub: [Enlace al código](#)

Importación de Librerías

El script comienza con la importación de varias librerías necesarias para la ejecución del movimiento:

- **sys**: Permite trabajar con los parámetros de la línea de comandos.
- **rospy**: Interfaz de Python para trabajar con ROS, que permite la interacción con nodos y mensajes en el sistema ROS.
- **moveit_commander**: Interfaz de Python para MoveIt!, utilizada para controlar y planificar trayectorias de robots.
- **geometry_msgs.msg**: Mensajes estándar de ROS que representan posiciones y orientaciones en el espacio.
- **trajectory_msgs.msg**: Mensajes utilizados para describir trayectorias de articulaciones del robot.
- **tf.transformations**: Utilizado para convertir ángulos de Euler a cuaterniones, necesarios para representar las orientaciones.
- **numpy (np)**: Biblioteca utilizada para cálculos matemáticos, como el cálculo de distancias en la función de longitud de la trayectoria.

Función para Calcular la Longitud de la Trayectoria

El script incluye la función `calculate_trajectory_length()`, que calcula la longitud total de la trayectoria en el espacio cartesiano. La función recorre cada punto de la trayectoria y suma la distancia entre posiciones consecutivas:

```
def calculate_trajectory_length(trajectory):
    length = 0.0
    previous_point = None
    for point in trajectory.joint_trajectory.points:
        current_position = point.positions
        if previous_point is not None:
            length += np.linalg.norm(np.array(current_position)
                                      - np.array(previous_point))
        previous_point = current_position
    return length
```

Inicialización del Nodo y la Planificación del Robot

A continuación, se inicializan MoveIt! y el nodo ROS:

```
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_robot', anonymous=True)
```

- `moveit_commander.roscpp_initialize()`: Inicializa MoveIt! con los parámetros pasados en la línea de comandos.
- `rospy.init_node()`: Inicializa un nodo ROS llamado `move_robot`, necesario para la comunicación entre ROS y Python.

Creación de Objetos para Manejar el Robot

Se crean objetos para interactuar con el robot y su grupo de movimientos:

```
robot = moveit_commander.RobotCommander()
group = moveit_commander.MoveGroupCommander("arm")
```

- `RobotCommander()`: Objeto que proporciona información sobre el robot, como el modelo y el estado del sistema.
- `MoveGroupCommander()`: Se utiliza para controlar un grupo de movimiento específico del robot, en este caso, el brazo (`.arm`).

Configuración de la Planificación

Se establece el tiempo máximo para planificar una trayectoria:

```
group.set_planning_time(60)
```

- `set_planning_time(60)`: Define el tiempo máximo de planificación de la trayectoria, en este caso 60 segundos.

Obtención de la Posición Inicial del Robot

Se obtiene la pose actual del robot como punto de partida para los movimientos:

```
start_pose = group.get_current_pose().pose
```

- `get_current_pose()`: Obtiene la pose (posición y orientación) actual del robot.

Definición de Waypoints

Se define una lista de waypoints que representarán los puntos intermedios en la trayectoria, comenzando con la pose inicial del robot:

```
waypoints = [start_pose] # Agregar el punto inicial (A)
```

- `waypoints`: Lista que contendrá los waypoints (puntos de paso) de la trayectoria.
- El primer waypoint es la posición inicial del robot (`start_pose`).

Definición de los Objetivos (Waypoints)

Se definen tres objetivos, cada uno con desplazamientos en los tres ejes (X , Y , Z) y una orientación específica utilizando cuaterniones.

Objetivo B

```
target_pose_1 = geometry_msgs.msg.Pose()
target_pose_1.position.x = start_pose.position.x + 0.1
target_pose_1.position.y = start_pose.position.y - 0.2
target_pose_1.position.z = start_pose.position.z - 0.1
q1 = quaternion_from_euler(0, 0, 1.57)
target_pose_1.orientation.x = q1[0]
target_pose_1.orientation.y = q1[1]
target_pose_1.orientation.z = q1[2]
target_pose_1.orientation.w = q1[3]
waypoints.append(target_pose_1)
```

El primer objetivo (`target_pose_1`) tiene un desplazamiento de $+0.1\text{ m}$ en X , -0.2 m en Y y -0.1 m en Z . La orientación se establece con una rotación de 90 grados en *Yaw* (alrededor del eje Z).

Objetivo C

```
target_pose_2 = geometry_msgs.msg.Pose()
target_pose_2.position.x = target_pose_1.position.x + 0.15
target_pose_2.position.y = target_pose_1.position.y - 0.1
target_pose_2.position.z = target_pose_1.position.z - 0.05
q2 = quaternion_from_euler(0.4, 1.57, -0.3)
target_pose_2.orientation.x = q2[0]
```

```

target_pose_2.orientation.y = q2[1]
target_pose_2.orientation.z = q2[2]
target_pose_2.orientation.w = q2[3]
waypoints.append(target_pose_2)

```

El segundo objetivo (`target_pose_2`) tiene un desplazamiento de $+0.15\text{ m}$ en X , -0.1 m en Y y -0.05 m en Z , con una rotación diferente.

Objetivo D

```

target_pose_3 = geometry_msgs.msg.Pose()
target_pose_3.position.x = target_pose_2.position.x + 0.5
target_pose_3.position.y = target_pose_2.position.y + 0.3
target_pose_3.position.z = target_pose_2.position.z - 0.75
q3 = quaternion_from_euler(0.785, -0.5, 0)
target_pose_3.orientation.x = q3[0]
target_pose_3.orientation.y = q3[1]
target_pose_3.orientation.z = q3[2]
target_pose_3.orientation.w = q3[3]
waypoints.append(target_pose_3)

```

El tercer objetivo (`target_pose_3`) tiene un desplazamiento de $+0.5\text{ m}$ en X , $+0.3\text{ m}$ en Y y -0.75 m en Z , con una orientación ajustada que corresponde a los valores `quaternion_from_euler(0.785, -0.5, 0)`.

Volver al Punto Inicial

Para cerrar el movimiento, se añade la posición inicial al final de los waypoints, de modo que el robot regrese a su posición de partida:

```
waypoints.append(start_pose)
```

Sin embargo, en este ejemplo, se ha comentado esta línea para que la posición final del robot sea diferente de la inicial, permitiendo observar claramente el desplazamiento realizado a través de los waypoints definidos.

Planificación de la Trayectoria

Se calcula la trayectoria cartesiana que conecta todos los waypoints definidos y se mide el tiempo de planificación:

```

planning_start_time = time.time()
(plan, fraction) = group.compute_cartesian_path(
    waypoints,    # Lista de waypoints a seguir
    0.02,          # Resolución de la trayectoria en metros
    0.0,
    avoid_collisions=True)
planning_time = time.time() - planning_start_time

```

- `compute_cartesian_path()`: Planifica una trayectoria cartesiana que pasa por los waypoints definidos.
- `planning_time`: Almacena el tiempo que ha tomado la planificación.

Cálculo de la Longitud Total de la Trayectoria

Una vez calculada la trayectoria, se evalúa su longitud total mediante la función `calculate_trajectory_length()`:

```
trajectory_length = calculate_trajectory_length(plan)
```

- `trajectory_length`: Representa la longitud total de la trayectoria en metros.

Ejecución de la Trayectoria Planificada

Si la planificación es completa (`fraction = 1.0`), se asignan tiempos incrementales para cada punto de la trayectoria y se ejecuta el plan:

```
if fraction == 1.0:
    for i, point in enumerate(plan.joint_trajectory.points):
        point.time_from_start = rospy.Duration(0.05 * i)
    execution_start_time = time.time()
    group.execute(plan, wait=True)
    execution_time = time.time() - execution_start_time
    rospy.loginfo("Tiempo de ejecución de la trayectoria: %.2f segundos" % execution_time)
else:
    rospy.logwarn("La trayectoria no se pudo calcular completamente.")
```

- `execution_time`: Tiempo de ejecución de la trayectoria.

Finalización del Movimiento

Al finalizar el movimiento, se detiene el robot y se cierra el nodo:

```
group.stop()
moveit_commander.roscpp_shutdown()
```

- `stop()`: Detiene cualquier movimiento en curso del robot.
- `roscpp_shutdown()`: Finaliza la sesión de MoveIt! y cierra el nodo ROS.

Validación del Resultado

La validación del resultado consistió en verificar que el brazo robótico pudo ejecutar correctamente la trayectoria calculada. La trayectoria fue calculada para los waypoints definidos, y el brazo ejecutó el movimiento siguiendo los puntos intermedios sin desviaciones significativas.

Durante la simulación, se observó que el brazo alcanzó todas las posiciones y orientaciones previstas en los waypoints, cumpliendo con los requisitos de alcance y precisión. Esto confirma que el sistema es capaz de planificar y ejecutar trayectorias de manera efectiva.

La imagen siguiente muestra la posición inicial del brazo antes de ejecutar el movimiento:

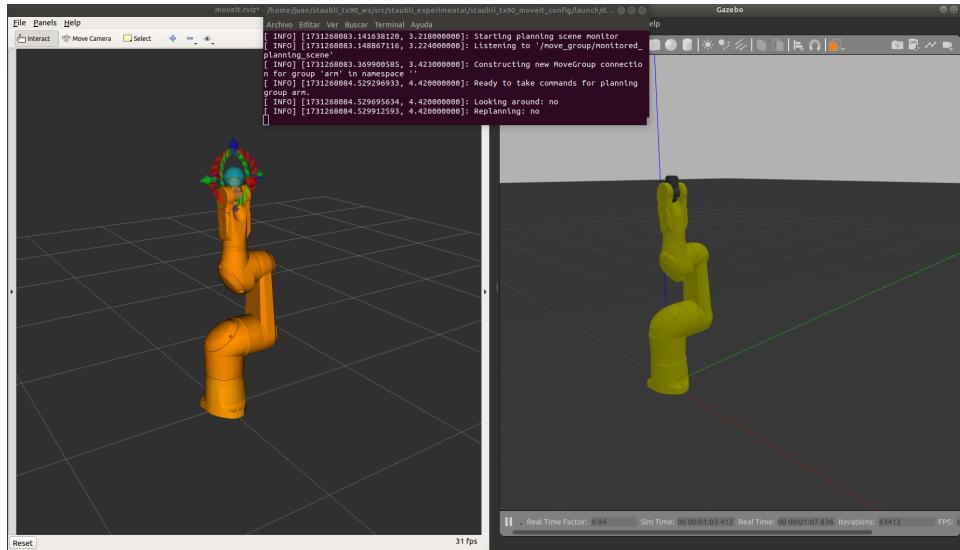


Figura 5.1: Posición inicial del brazo robótico.

En la siguiente imagen, se muestra el resultado final del brazo después de ejecutar la trayectoria. En esta imagen, podemos ver la terminal con el mensaje que indica que la ejecución de la trayectoria fue exitosa:

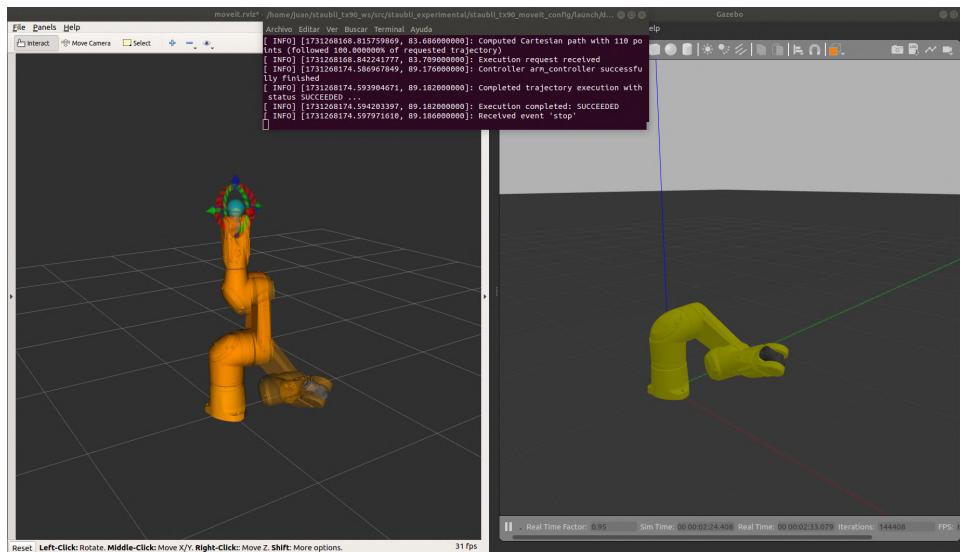


Figura 5.2: Posición final del brazo robótico y mensaje de éxito en la terminal.

En la terminal, se puede observar el siguiente mensaje que confirma la ejecución exitosa de la trayectoria:

```
[ INFO] [1731268174.594203397, 89.182000000]: Execution completed: SUCCEEDED
```

Este mensaje indica que el sistema completó la trayectoria sin problemas, y el brazo robótico alcanzó la última posición y orientación de manera precisa.

A continuación se presenta un enlace a un video que muestra un ejemplo de ejecución del brazo robótico en simulación:

[Video de ejemplo de ejecución en simulación](#)

5.2. Prueba con Brazo Robótico Físico en un Entorno Realista

La prueba de campo en la vida real se llevó a cabo en el laboratorio utilizando el brazo robótico Staubli TX90 real, siguiendo una metodología similar a la prueba de simulación previamente explicada. El objetivo de esta prueba fue verificar la correcta conexión entre ROS, MoveIt! y el controlador CS9, utilizando un script similar pero con movimientos más suaves para garantizar la seguridad y precisión de la ejecución.

Ajustes en el Controlador CS9

Para poder comunicarme con el controlador CS9 desde ROS, fue necesario realizar ciertos ajustes en la configuración del controlador. Esto fue posible gracias al uso del paquete `staubli_val3_driver`, que proporciona una interfaz entre ROS y el controlador VAL3 utilizado en el brazo Staubli TX90. El controlador CS9 permite controlar el brazo robótico a través de la programación en VAL3, pero para interactuar con ROS y MoveIt!, se requiere configurar adecuadamente la comunicación entre ambos sistemas.

Los detalles completos sobre los ajustes necesarios en el controlador CS9 para establecer la comunicación a través de VAL3 están documentados en el archivo README del repositorio `staubli_val3_driver` disponible en GitHub. Este archivo explica cómo configurar la comunicación entre ROS y el controlador CS9, incluyendo la instalación del controlador, la configuración de los parámetros de conexión, y la creación de los nodos necesarios para establecer la comunicación bidireccional entre el sistema ROS y el brazo robótico.

Puedes encontrar los detalles completos de la configuración en el siguiente enlace del repositorio GitHub: **[Guía de configuración del controlador CS9](#)**.

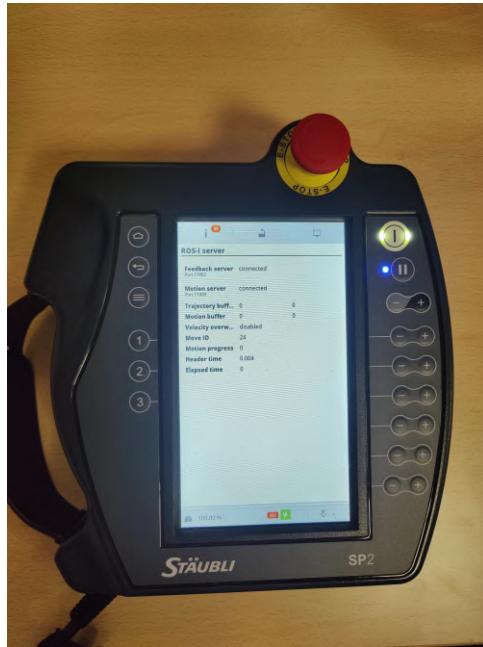


Figura 5.3: Teach pendant SP2 del controlador CS9 mostrando el estado de conexión y ejecución del robot Staubli con ROS.

Metodología de la Prueba

Una vez establecida la comunicación entre ROS, MoveIt! y el controlador CS9, se ejecutó un script de Python similar al utilizado en la prueba de simulación. Este script estaba diseñado para mover el brazo robótico a través de una serie de waypoints definidos en el espacio tridimensional, con la diferencia de que en esta prueba se implementaron movimientos más suaves y controlados para evitar cualquier tipo de daño o colisión en el robot.

El objetivo principal de esta prueba fue asegurar que los movimientos ejecutados por el brazo real fueran suaves y precisos, cumpliendo con los requisitos de control y conectividad entre los sistemas involucrados. En la simulación se pudieron verificar correctamente los movimientos entre los puntos intermedios definidos, pero en la prueba real, se enfocó en la fluidez y seguridad de esos movimientos, manteniendo una comunicación constante con el controlador CS9 a través de VAL3.

De forma similar a la prueba de simulación, en esta prueba se utilizaron los mismos principios de planificación de trayectorias y control de los movimientos del robot, pero con un mayor enfoque en la optimización de la ejecución de los movimientos en un entorno físico real. Se observó que el brazo robótico pudo ejecutar correctamente los movimientos definidos en los waypoints, con precisión y sin errores en la comunicación entre ROS y el controlador CS9.

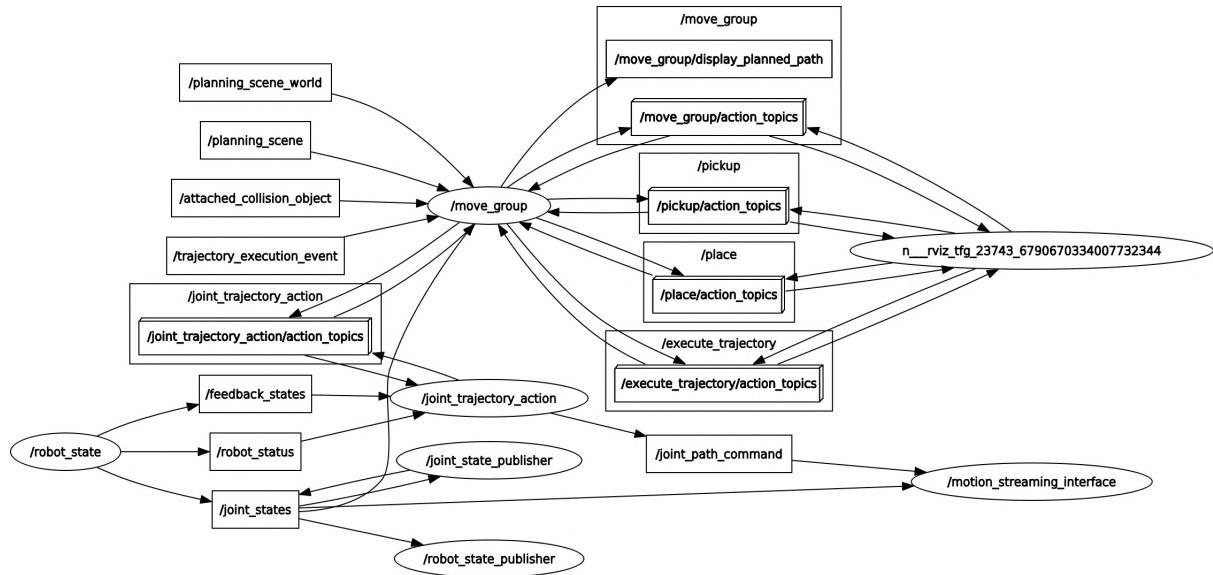


Figura 5.4: Estructura de nodos y tópicos en la prueba en entorno real de ROS

En la Figura 5.4, se muestra el grafo de nodos y tópicos involucrados en la planificación y control del brazo robótico Staubli TX2-90 con MoveIt! y ROS. Un nodo clave es `/motion_streaming_interface`, que actúa como puente de comunicación entre el entorno ROS y el controlador CS9, que utiliza el software VAL3.

Este nodo permite transmitir los comandos de ROS al controlador CS9, que ejecuta las trayectorias planificadas. Sin `/motion_streaming_interface`, la comunicación entre ROS y el sistema propietario no sería posible, lo cual impediría una ejecución precisa de las trayectorias. En resumen, este nodo facilita la integración entre el planificador y el hardware del robot, garantizando un control fluido y eficiente.

5.3. Prueba de Precisión

La prueba de precisión se realizó con el objetivo de evaluar la precisión del brazo robótico Staubli TX2-90 al ejecutar una trayectoria específica, utilizando un rotulador acoplado en el efector final. Esta prueba fue una extensión directa de la *Prueba con Brazo Robótico Físico en un Entorno Realista* descrita anteriormente. Una vez garantizada la comunicación entre ROS, MoveIt! y el controlador CS9, se llevaron a cabo pruebas adicionales con el propósito de medir la precisión de los movimientos y la capacidad del brazo robótico para ejecutar tareas precisas en un entorno realista.



Figura 5.5: Posición del brazo robótico Staubli TX2-90 y el entorno de la prueba

Objetivo de la Prueba

El principal objetivo de esta prueba fue verificar la precisión del brazo robótico al trazar una trayectoria específica en un plano. En este caso, la trayectoria definida correspondía a un cuadrado, cuyos cuatro vértices se introdujeron como waypoints en un script de Python similar al utilizado en la prueba anterior. La finalidad era comprobar que el brazo pudiera seguir esta trayectoria de forma precisa, dibujando un cuadrado con líneas claras y sin desviaciones significativas, logrando así validar la fiabilidad del sistema de control y conectividad previamente establecido.

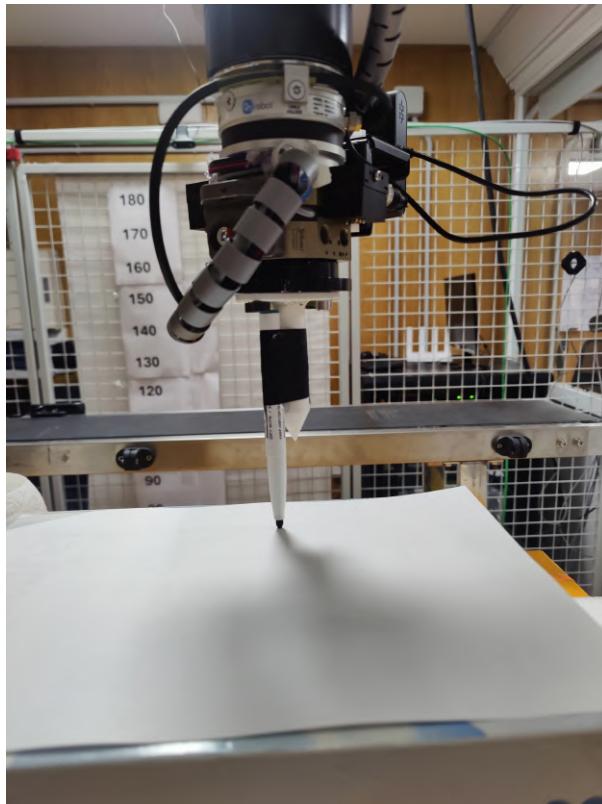


Figura 5.6: Efecto final utilizado.

Metodología

La metodología para esta prueba se basó en los mismos principios de planificación de trayectorias utilizados en la *Prueba con Brazo Robótico Físico en un Entorno Realista*, con un enfoque más estricto sobre el movimiento para asegurar que el rotulador pudiera dibujar un cuadrado perfecto sobre una superficie. La prueba se dividió en dos ejecuciones independientes, cada una diseñada para evaluar la repetibilidad y la precisión del brazo robótico al seguir la trayectoria especificada.

Ejecución y Resultados

- **Primer Test:** En esta primera ejecución, el brazo dibujó el cuadrado sobre la superficie con los parámetros definidos inicialmente.

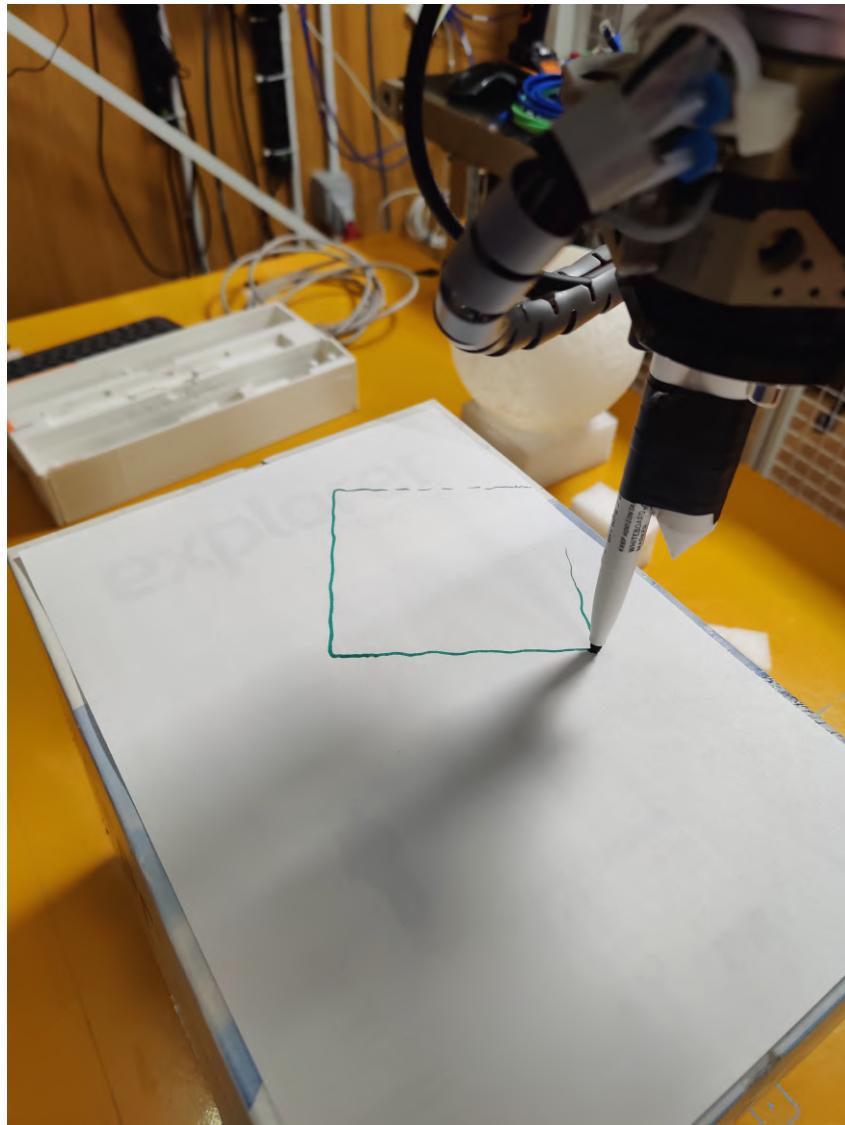


Figura 5.7: Resultado Test 1

[Video del Test 1](#)

- **Segundo Test:** Se realizó una segunda ejecución con la misma configuración, para evaluar la consistencia y la reproducibilidad de los resultados obtenidos en el primer test.

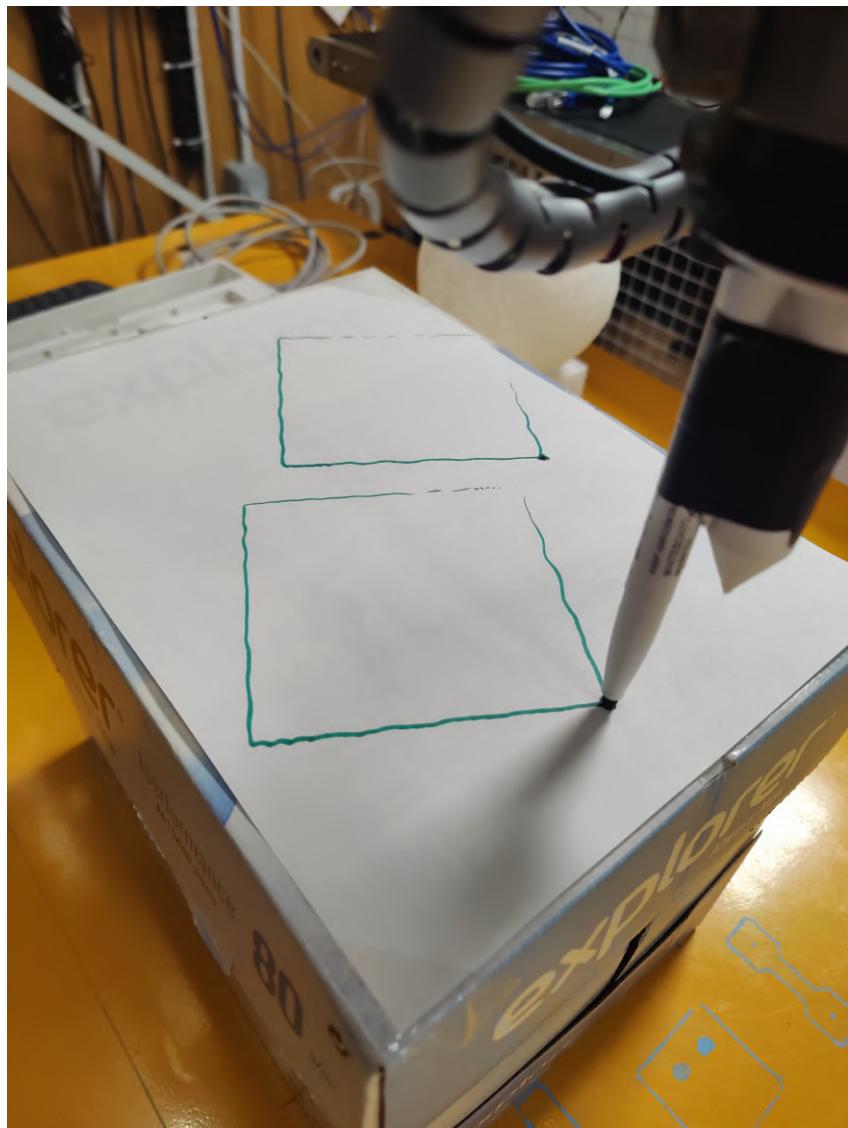


Figura 5.8: Resultado Test 2

[Video del Test 2](#)

- **Estado de la ejecución en mi equipo:** La siguiente figura muestra el estado final de la ejecución de la trayectoria planificada en mi equipo, visualizada tanto en la terminal como en RViz.

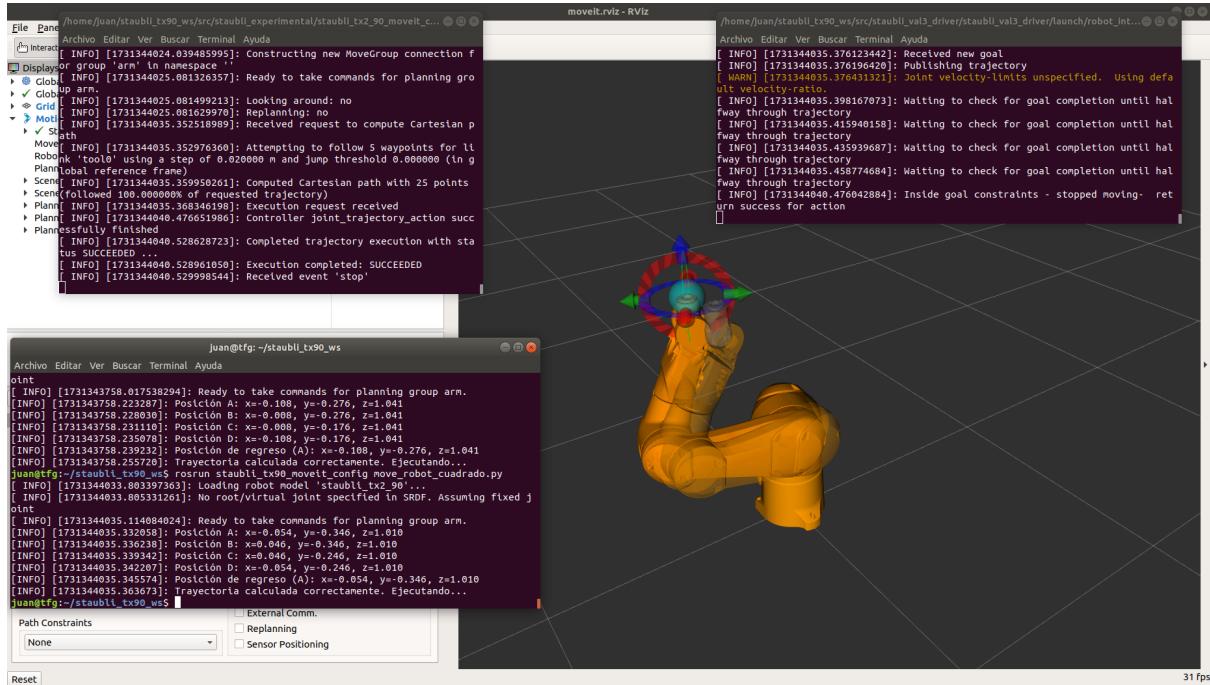


Figura 5.9: Estado final de la ejecución de la prueba en la terminal y en RViz

En la Figura 5.9, se observa la ejecución de la trayectoria planificada para el brazo robótico utilizando MoveIt! y ROS. En la terminal superior izquierda, se confirma que la trayectoria ha sido completada con éxito, como se indica con el mensaje Execution completed: SUCCEEDED. Además, en la terminal inferior izquierda se muestra la salida del script, que incluye las coordenadas por las cuales pasa la trayectoria, es decir, los vértices del cuadrado definido. Estas posiciones corresponden a las waypoints que el brazo debe alcanzar para realizar el movimiento deseado.

Análisis de Resultados

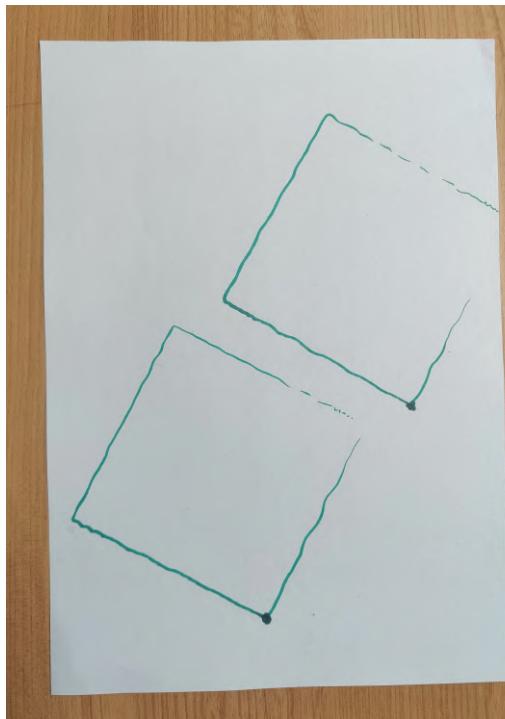


Figura 5.10:

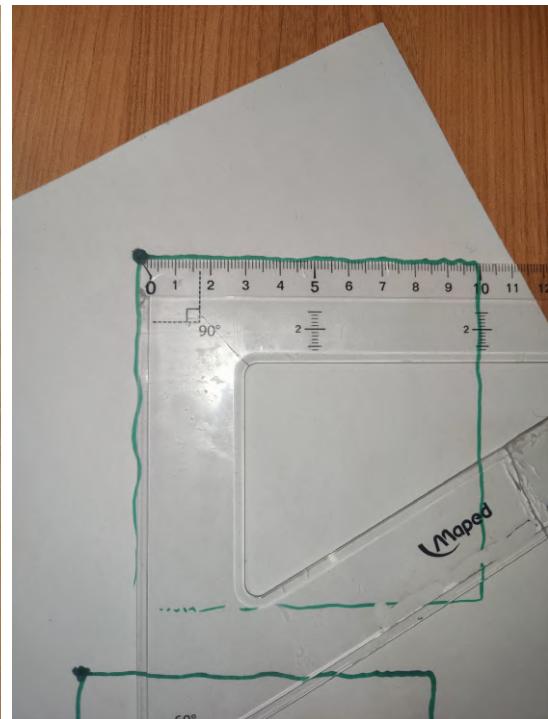


Figura 5.11:

Figura 5.12: Resultado final y comprobación de medidas del cuadrado.

En la Figura 5.10, se presentan los dos cuadrados dibujados correspondientes a las dos ejecuciones del test de precisión. A partir de esta imagen, se puede deducir que ambos cuadrados están prácticamente completos, lo cual permite considerar la prueba como exitosa en términos de ejecución de la trayectoria planificada. Aunque hay algunas pequeñas irregularidades en los trazados, la forma general del cuadrado se logró correctamente en ambas ejecuciones, lo cual demuestra una buena consistencia en la repetibilidad del movimiento.

En la Figura 5.11, se muestra una comprobación de las medidas del cuadrado. En esta imagen, se observa que los lados tienen una longitud aproximada de 10 cm, que corresponde a la variación en los ejes que se había programado en el script de Python. Además, se puede apreciar que el ángulo entre los lados del cuadrado es prácticamente de 90° , lo cual confirma la precisión esperada en la trayectoria.

Sin embargo, durante la ejecución de la trayectoria, el brazo robótico presentó un cierto temblor, lo cual quedó reflejado en la irregularidad de las líneas de ambos cuadrados. Este patrón de vibración fue similar en ambas ejecuciones, lo que sugiere que podría deberse a una discrepancia entre los ajustes y parámetros del robot real y el modelo utilizado por el planificador. Al tratarse de movimientos en un espacio reducido, es posible que se haya generado una tensión adicional que provocó estas vibraciones.

Otro aspecto importante a mencionar es que existen algunas partes de los lados que no fueron bien pintadas por el rotulador. Consideramos que esto se debió a la superficie sobre la que estaba colocado el papel, que era una caja cuya rigidez no era uniforme, lo cual provocó una cierta falta de contacto en algunas zonas, afectando la calidad del trazo.

En conclusión, los resultados muestran que el brazo robótico Staubli TX2-90 pudo seguir la

trayectoria planificada y dibujar el cuadrado con una precisión aceptable, a pesar de algunas irregularidades debido a la vibración y la naturaleza de la superficie de apoyo. Estos aspectos deberían ser optimizados para mejorar aún más la precisión del sistema en futuras pruebas.

5.4. Comparación de Algoritmos

En esta sección se compararán los resultados obtenidos utilizando diez algoritmos de planificación de trayectorias: RRTConnect, RRTStar, LazyPRM, BKPIECE y SPARS. Cada uno de estos algoritmos tiene diferentes enfoques para resolver el problema de planificación de trayectorias, y se evaluarán sus rendimientos en términos de longitud de la trayectoria, tiempo de ejecución y tiempo de cálculo.

Para esta evaluación, se realizará la misma prueba en simulación descrita en la sección **Prueba en Simulación**, la cual consistirá en ejecutar el mismo script un total de diez veces por cada algoritmo. Con estos datos se realizará un análisis comparativo final para determinar cuál de los algoritmos es el más adecuado según los criterios de rendimiento definidos.

Antes de analizar los resultados específicos de cada algoritmo, es importante entender cómo se calcula la longitud de una trayectoria en este contexto. A continuación, se describe el método empleado en el script para calcular la longitud y el significado de los valores obtenidos.

5.4.1. Cálculo de la Longitud de la Trayectoria

En esta prueba de planificación de trayectorias, vamos a trabajar con la **distancia entre las posiciones de las articulaciones del robot**, en lugar de una distancia medida en metros en el espacio físico. Esto significa que, al calcular la longitud de la trayectoria, estamos midiendo cuánto cambian las posiciones de las articulaciones del robot a lo largo del movimiento, en vez de la distancia que recorre un punto en el espacio.

Medir la distancia en términos de las articulaciones nos permite entender mejor el **esfuerzo necesario para que cada articulación complete la trayectoria**. Esta medida es útil porque una trayectoria que minimice los cambios en las posiciones de las articulaciones reduce el desgaste de los componentes y puede ahorrar energía. Esto es especialmente importante en robots con varias articulaciones, donde cada cambio de posición representa un esfuerzo y consumo de recursos.

La función `calculate_trajectory_length` del script se encarga de calcular esta longitud de la siguiente manera. Dados los puntos sucesivos de la trayectoria en el espacio de las articulaciones, representados como $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n$, la longitud total de la trayectoria L se calcula como la suma de las distancias entre cada par de puntos sucesivos en el espacio de articulaciones:

$$L = \sum_{i=1}^{n-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|, \quad (5.1)$$

donde $\|\mathbf{p}_{i+1} - \mathbf{p}_i\|$ representa la **distancia euclíadiana** entre las posiciones de articulaciones consecutivas \mathbf{p}_i y \mathbf{p}_{i+1} .

La distancia euclíadiana es una medida de la distancia en línea recta entre dos puntos en un espacio euclidiano [38]. Se utiliza para calcular la longitud del segmento de recta que une dos puntos en un espacio multidimensional. Matemáticamente, para dos puntos $\mathbf{a} = (a_1, a_2, \dots, a_n)$ y $\mathbf{b} = (b_1, b_2, \dots, b_n)$ en un espacio de n -dimensiones, la distancia euclíadiana se calcula como:

$$\|\mathbf{b} - \mathbf{a}\| = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \dots + (b_n - a_n)^2}. \quad (5.2)$$

En el contexto del script, cada \mathbf{p}_i representa las posiciones de las articulaciones del robot en un punto específico de la trayectoria. Esta distancia entre posiciones se calcula en Python usando la función `np.linalg.norm`, como se muestra en la función `calculate_trajectory_length`:

```
def calculate_trajectory_length(traj):
    length = 0.0
    previous_point = None
    for point in traj.joint_trajectory.points:
        current_position = point.positions
        if previous_point is not None:
            length += np.linalg.norm(np.array(current_position)
                                      - np.array(previous_point))
        previous_point = current_position
    return length
```

Explicación del Cálculo

1. Inicialización: La función inicia con un valor de `length` igual a 0.0, que se usará para acumular la longitud total de la trayectoria.

2. Iteración sobre los puntos de la trayectoria: Para cada punto en la trayectoria, se obtiene la posición de las articulaciones del robot, representada como un conjunto de valores de ángulos o posiciones de las articulaciones.

3. Cálculo de la distancia en el espacio de las articulaciones: Para cada par de puntos consecutivos, la función calcula la distancia entre las posiciones de las articulaciones utilizando la **distancia Euclídea**. Esto se realiza en el espacio de las articulaciones, lo que significa que se está midiendo cuánto han cambiado las posiciones de las articulaciones entre cada par de puntos.

4. Suma acumulativa de la distancia: La distancia entre cada par de puntos sucesivos se suma a `length`, proporcionando la longitud total de la trayectoria al final.

Interpretación del Valor de la Longitud

La longitud obtenida representa la **cantidad total de movimiento que realizan las articulaciones del robot** a lo largo de toda la trayectoria. Este valor no se mide en unidades físicas convencionales, ya que no describe una distancia en el espacio físico, sino los ajustes acumulados en las posiciones de las articulaciones. Un valor de longitud más alto indica que el robot ha realizado movimientos más extensos en sus articulaciones, mientras que un valor más bajo sugiere que ha seguido una trayectoria más directa, con cambios menores en las posiciones articulares.

Significado de un Valor de Longitud Más Alto o Más Bajo

- Un **valor de longitud más alto** generalmente indica que el robot ha seguido una trayectoria más compleja, que ha requerido mayores ajustes en sus articulaciones. Esto puede ocurrir cuando el robot necesita evitar obstáculos o cumplir con restricciones específicas, lo cual incrementa el movimiento necesario en las articulaciones.

- Un **valor de longitud más bajo** sugiere una trayectoria más directa o eficiente, donde las articulaciones han realizado menos ajustes. Este valor suele ser deseable desde el punto de

vista de la eficiencia, ya que implica un menor esfuerzo para las articulaciones y, por lo tanto, menos desgaste de los componentes y un posible ahorro energético.

Con esta interpretación, podemos analizar los resultados obtenidos por cada algoritmo de planificación, evaluando la eficiencia del movimiento en función del valor de longitud calculado para las articulaciones del robot.

5.4.2. RRTConnect

El algoritmo RRTConnect es una variante mejorada del algoritmo RRT (*Rapidly-exploring Random Tree*) diseñada para mejorar la eficiencia en la planificación de trayectorias. Este algoritmo emplea dos árboles de exploración: uno que crece desde el punto de inicio y otro desde el punto de destino, intentando conectar ambos árboles de manera rápida y eficaz. Esta estrategia bidireccional permite que RRTConnect encuentre soluciones rápidamente, incluso en espacios de configuración de alta dimensión, donde los métodos unidireccionales pueden requerir un mayor tiempo de exploración. La capacidad de RRTConnect para expandir los árboles en ambas direcciones lo convierte en una opción adecuada para problemas de planificación que requieren una respuesta ágil y eficiente. [39]

| Ejecución | Número de puntos | Longitud total | Tiempo ejecución (s) | Tiempo de cálculo (s) |
|-----------|------------------|----------------|----------------------|-----------------------|
| 1 | 169 | 31.822 | 9.21 | 0.11 |
| 2 | 169 | 19.321 | 9.27 | 0.06 |
| 3 | 169 | 19.318 | 9.20 | 0.06 |
| 4 | 169 | 19.316 | 9.16 | 0.05 |
| 5 | 169 | 19.314 | 9.15 | 0.05 |
| 6 | 169 | 19.313 | 9.20 | 0.06 |
| 7 | 169 | 19.312 | 9.25 | 0.05 |
| 8 | 169 | 19.311 | 9.28 | 0.06 |
| 9 | 169 | 19.310 | 9.18 | 0.06 |
| 10 | 169 | 19.309 | 9.23 | 0.05 |

Cuadro 5.1: Resultados de RRTConnect

El promedio de RRTConnect es el siguiente:

- Longitud total de la trayectoria: 20.8646
- Tiempo de ejecución de la trayectoria: 9.203 s
- Tiempo de cálculo de la trayectoria: 0.061 s

En las diez ejecuciones, RRTConnect muestra una longitud de trayectoria consistente, con una media de 20.8646 y ligeras variaciones. Los valores más altos se observan en la primera ejecución (31.822), lo que podría deberse a la exploración inicial del algoritmo, mientras que las siguientes iteraciones muestran una mayor consistencia en la longitud de la trayectoria.

El tiempo de cálculo promedio es bajo, de 0.061 segundos, y el tiempo de ejecución es estable, con una media de 9.203 segundos y pequeñas variaciones. En general, RRTConnect resulta ser una opción eficaz para la planificación de trayectorias, ofreciendo una buena combinación entre optimización de movimiento y tiempos de cálculo reducidos.

5.4.3. RRTStar

El algoritmo **RRTStar** (*Rapidly-exploring Random Tree Star*) es una mejora del algoritmo original **RRT** que no solo busca una solución en el espacio de configuración, sino que también optimiza la calidad de la trayectoria encontrada. Durante su proceso de construcción, **RRTStar** ajusta la conexión de los puntos del árbol para minimizar el costo total del trayecto, lo que resulta en una ruta más corta o más eficiente. Este enfoque de refinamiento incremental lo hace adecuado para aplicaciones donde la calidad de la trayectoria es crítica, aunque esto puede implicar un tiempo de cálculo ligeramente mayor debido a la reconfiguración continua de los puntos del árbol. [40]

| Ejecución | Número de puntos | Longitud total | Tiempo ejecución (s) | Tiempo de cálculo (s) |
|-----------|------------------|----------------|----------------------|-----------------------|
| 1 | 169 | 41.764 | 8.98 | 0.10 |
| 2 | 169 | 39.911 | 9.16 | 0.12 |
| 3 | 169 | 19.321 | 9.02 | 0.05 |
| 4 | 169 | 19.316 | 9.06 | 0.05 |
| 5 | 169 | 19.314 | 9.08 | 0.07 |
| 6 | 169 | 19.313 | 9.22 | 0.06 |
| 7 | 169 | 19.312 | 9.16 | 0.05 |
| 8 | 169 | 19.311 | 9.12 | 0.05 |
| 9 | 169 | 19.310 | 9.10 | 0.06 |
| 10 | 169 | 19.309 | 9.08 | 0.06 |

Cuadro 5.2: Resultados de RRTStar

El promedio de **RRTStar** es el siguiente:

- Longitud total de la trayectoria: 23.6181
- Tiempo de ejecución de la trayectoria: 9.104 s
- Tiempo de cálculo de la trayectoria: 0.067 s

En las diez ejecuciones, **RRTStar** muestra una longitud de trayectoria que varía significativamente en las primeras iteraciones, con valores que oscilan entre 19.309 y 41.764. La longitud promedio de la trayectoria es de 23.6181, lo cual refleja que la optimización de la trayectoria puede variar más en las primeras ejecuciones, mientras que las últimas iteraciones tienden a ser más consistentes.

El tiempo de cálculo promedio es bajo, de 0.067 segundos, con pequeñas variaciones entre ejecuciones. El tiempo de ejecución también es consistente, con un promedio de 9.104 segundos.

En general, **RRTStar** ofrece un buen equilibrio entre eficiencia de trayectoria y rapidez de cálculo, siendo una opción confiable para la planificación cuando se busca una trayectoria optimizada con tiempos de planificación mínimos. Las mayores desviaciones en la longitud de la trayectoria se observan en las primeras ejecuciones, posiblemente debido a factores iniciales de ajuste o a la naturaleza iterativa del algoritmo.

5.4.4. LazyPRM

El algoritmo **LazyPRM** (*Probabilistic Roadmap Method*) es un método de planificación probabilística que genera un mapa de caminos en el espacio libre del robot, representado como un

grafo, y conecta estos caminos solo cuando es necesario. Este enfoque permite una planificación rápida al retrasar la validación de colisiones de las conexiones entre nodos del grafo hasta que se necesite, en lugar de hacerlo de inmediato, de ahí el nombre **Lazy** (perezoso). La estrategia es particularmente útil en espacios de configuración grandes, ya que evita calcular trayectorias completas que luego pueden no ser necesarias. [41]

En general, LazyPRM es eficiente en situaciones donde la planificación rápida es prioritaria, aunque puede generar caminos subóptimos si no se realiza una refinación adicional del mapa.

| Ejecución | Número de puntos | Longitud total | Tiempo ejecución (s) | Tiempo de cálculo (s) |
|-----------|------------------|----------------|----------------------|-----------------------|
| 1 | 169 | 62.769 | 9.12 | 0.14 |
| 2 | 169 | 28.035 | 9.04 | 0.08 |
| 3 | 169 | 19.318 | 9.00 | 0.05 |
| 4 | 169 | 19.316 | 9.22 | 0.06 |
| 5 | 169 | 19.314 | 9.07 | 0.05 |
| 6 | 169 | 19.313 | 9.03 | 0.05 |
| 7 | 169 | 19.312 | 9.09 | 0.05 |
| 8 | 169 | 19.311 | 9.08 | 0.04 |
| 9 | 169 | 19.310 | 9.10 | 0.06 |
| 10 | 169 | 19.309 | 9.09 | 0.05 |

Cuadro 5.3: Resultados de LazyPRM

El promedio de LazyPRM es el siguiente:

- Longitud total de la trayectoria: 24.8606
- Tiempo de ejecución de la trayectoria: 9.084 s
- Tiempo de cálculo de la trayectoria: 0.073 s

En las diez ejecuciones, LazyPRM muestra una longitud de trayectoria que varía significativamente en las primeras iteraciones, con valores entre 19.309 y 62.769, y una longitud promedio de 24.8606. Esto sugiere que, en las primeras ejecuciones, el algoritmo explora trayectorias menos optimizadas, pero mejora en las siguientes, tendiendo hacia valores más bajos y consistentes.

El tiempo de cálculo promedio es bajo, de 0.073 segundos, con mínimas variaciones entre ejecuciones. El tiempo de ejecución es también consistente, con un promedio de 9.084 segundos y pequeñas variaciones.

En general, LazyPRM demuestra ser una opción confiable para planificación de trayectorias, combinando eficiencia en el tiempo de ejecución con una mejora gradual en la optimización de la trayectoria. La variabilidad en la longitud de la trayectoria en las primeras ejecuciones se alinea con la naturaleza iterativa de LazyPRM, que explora diversas conexiones en el espacio libre y se optimiza conforme avanza.

5.4.5. BKPIECE

El algoritmo BKPIECE (*Bi-directional KPIECE*) es una extensión del método KPIECE (*Kinematic Planning by Interior-Exterior Cell Exploration*), que se centra en una planificación eficiente en espacios de alta dimensión mediante una estrategia de búsqueda bidireccional. BKPIECE

utiliza una representación jerárquica del espacio de configuración, dividiéndolo en celdas de exploración, lo cual facilita la identificación de áreas no exploradas y permite guiar la expansión de la trayectoria en ambas direcciones. Esto reduce el tiempo de planificación al explorar de manera dirigida el espacio de configuración, manteniendo la calidad del camino resultante incluso en entornos complejos. [42]

| Ejecución | Número de puntos | Longitud total | Tiempo ejecución (s) | Tiempo de cálculo (s) |
|-----------|------------------|----------------|----------------------|-----------------------|
| 1 | 169 | 49.942 | 8.88 | 0.11 |
| 2 | 169 | 19.321 | 8.88 | 0.05 |
| 3 | 169 | 19.318 | 9.03 | 0.06 |
| 4 | 169 | 19.316 | 8.89 | 0.05 |
| 5 | 169 | 19.314 | 8.99 | 0.05 |
| 6 | 169 | 19.313 | 8.84 | 0.06 |
| 7 | 169 | 19.312 | 9.00 | 0.05 |
| 8 | 169 | 19.311 | 8.84 | 0.05 |
| 9 | 169 | 19.310 | 8.88 | 0.05 |
| 10 | 169 | 19.309 | 8.99 | 0.05 |

Cuadro 5.4: Resultados de BKPIECE

El promedio de BKPIECE es el siguiente:

- Longitud total de la trayectoria: 22.6466
- Tiempo de ejecución de la trayectoria: 8.922 s
- Tiempo de cálculo de la trayectoria: 0.058 s

El algoritmo BKPIECE muestra consistencia en la longitud de la trayectoria, con valores que oscilan entre 19.309 y 49.942, y un promedio de 22.6466. La desviación en la primera ejecución (49.942) puede deberse a la fase inicial de exploración, mientras que las siguientes trayectorias son más optimizadas y consistentes.

El tiempo de cálculo es bajo y estable, con un promedio de 0.058 segundos y una variación mínima entre ejecuciones. El tiempo de ejecución promedio es de 8.922 segundos, con pequeñas variaciones, lo que indica un rendimiento eficiente y predecible.

En resumen, BKPIECE destaca por su rápida planificación y trayectorias consistentes y cortas tras las primeras iteraciones, siendo ideal para aplicaciones que requieren eficiencia en cálculo y ejecución.

5.4.6. SPARS

El algoritmo SPARS (*Sparse Roadmap Spanner*) es un algoritmo de planificación de trayectorias basado en muestreo probabilístico. El objetivo de SPARS es construir una representación de camino eficiente y ligera en el espacio de configuración del robot, donde las conexiones entre puntos de la trayectoria se establecen únicamente cuando es necesario para garantizar la conectividad. Este enfoque es particularmente ventajoso en espacios de alta dimensión y complejos, ya que evita una búsqueda exhaustiva y optimiza el uso de nodos y conexiones. En general, SPARS equilibra la exploración del espacio de configuración y la eficiencia en la construcción del camino.[43]

| Ejecución | Número de puntos | Longitud total | Tiempo ejecución (s) | Tiempo de cálculo (s) |
|-----------|------------------|----------------|----------------------|-----------------------|
| 1 | 169 | 28.057 | 9.19 | 0.12 |
| 2 | 169 | 19.321 | 9.16 | 0.06 |
| 3 | 169 | 19.318 | 9.12 | 0.05 |
| 4 | 169 | 19.316 | 9.13 | 0.05 |
| 5 | 169 | 19.314 | 9.09 | 0.06 |
| 6 | 169 | 19.313 | 9.09 | 0.05 |
| 7 | 169 | 19.312 | 9.23 | 0.05 |
| 8 | 169 | 19.311 | 9.22 | 0.06 |
| 9 | 169 | 19.310 | 9.19 | 0.05 |
| 10 | 169 | 19.309 | 9.19 | 0.05 |

Cuadro 5.5: Resultados de SPARS

El promedio de SPARS es el siguiente:

- Longitud total de la trayectoria: 19.318
- Tiempo de ejecución de la trayectoria: 9.16 s
- Tiempo de cálculo de la trayectoria: 0.06 s

El algoritmo SPARS muestra alta consistencia en las longitudes de trayectoria en la mayoría de las ejecuciones, con una pequeña dispersión en los resultados. Esta estabilidad en la longitud de la trayectoria sugiere que SPARS genera trayectorias optimizadas con mínima variabilidad.

Los tiempos de cálculo y ejecución son muy estables, con tiempos de cálculo entre 0.05 y 0.12 segundos, y tiempos de ejecución entre 9.09 y 9.23 segundos. Esto convierte a SPARS en una opción eficiente para aplicaciones que requieren trayectorias predecibles y tiempos de ejecución consistentes.

5.4.7. Análisis y Comparativa de Resultados

En este análisis, cada algoritmo ha sido ejecutado diez veces. Exceptuando las dos primeras ejecuciones, se observa que en la mayoría de los casos todos los algoritmos obtienen una longitud de trayectoria consistente, con valores prácticamente constantes en las repeticiones restantes. Este comportamiento indica que los algoritmos son capaces de calcular trayectorias similares en cuanto a distancia, mostrando una estabilidad significativa a partir de la tercera ejecución.

| Métrica | RRTConnect | RRTStar | LazyPRM | BKPIECE | SPARS |
|---|------------|---------|---------|---------|--------|
| Número de puntos en la trayectoria | 169 | 169 | 169 | 169 | 169 |
| Longitud total de la trayectoria | 20.8646 | 23.6181 | 24.8606 | 22.6466 | 19.318 |
| Tiempo de ejecución (s) | 9.203 | 9.104 | 9.084 | 8.922 | 9.16 |
| Tiempo de cálculo de la trayectoria (s) | 0.061 | 0.067 | 0.073 | 0.058 | 0.06 |

Cuadro 5.6: Comparativa de Resultados Promedio de los Algoritmos

Análisis General de los Resultados Promedio

Al comparar los diferentes algoritmos, se observa que existen diferencias en términos de estabilidad, tiempos de cálculo y tiempos de ejecución. Estas diferencias están relacionadas con la filosofía de búsqueda de cada algoritmo y el comportamiento en las primeras ejecuciones.

RRTConnect: Este algoritmo muestra un buen equilibrio entre rapidez y estabilidad, con un tiempo de cálculo de 0.061 s y un tiempo de ejecución de 9.203 s. Aunque las primeras ejecuciones presentaron alguna variabilidad, RRTConnect alcanza una mayor consistencia rápidamente, lo que lo hace adecuado para tareas que requieren una planificación eficiente y resultados repetibles.

RRTStar: RRTStar, al basarse en la optimización iterativa de las trayectorias, presenta una mayor variabilidad en las primeras ejecuciones antes de converger hacia una solución más estable. Esto se refleja en un tiempo de cálculo de 0.067 s y un rendimiento ligeramente inferior en estabilidad comparado con RRTConnect. A pesar de ello, una vez superadas las primeras ejecuciones, RRTStar ofrece trayectorias optimizadas de forma consistente.

LazyPRM: Este algoritmo, con su enfoque de "planificación perezosa", resulta menos eficiente en las primeras ejecuciones debido a la falta de comprobaciones iniciales, lo que se traduce en una mayor desviación en las trayectorias calculadas. Con un tiempo de cálculo de 0.073 s, se observa que las primeras ejecuciones se desvían significativamente de las trayectorias óptimas, lo cual lo hace menos adecuado para aplicaciones que requieren consistencia desde el comienzo.

BKPIECE: BKPIECE destaca por su estabilidad y rapidez. Su tiempo de cálculo de 0.058 s, el más bajo entre los algoritmos, y su tiempo de ejecución de 8.922 s reflejan una búsqueda eficiente en regiones poco exploradas. Tras unas pocas ejecuciones iniciales, BKPIECE alcanza rápidamente una buena consistencia, convirtiéndose en la opción más adecuada cuando la estabilidad y eficiencia son fundamentales.

SPARS: SPARS, con su enfoque de generación de un conjunto disperso de nodos, permite una cobertura eficiente del espacio libre. Aunque las primeras ejecuciones presentan alguna desviación, después de estas SPARS logra una estabilidad razonable, con un tiempo de cálculo de 0.06 s. A pesar de su ligero aumento en el tiempo de ejecución (9.16 s), se muestra adecuado para situaciones donde la longitud optimizada y la repetibilidad son más importantes que la rapidez.

Análisis de Desviación en las Primeras Ejecuciones

Durante las primeras dos ejecuciones de cada algoritmo, se observaron desviaciones significativas en la longitud de la trayectoria, lo cual generó la hipótesis de que los algoritmos podrían estar optimizando sus trayectorias con cada nueva ejecución. Esta idea sugiere que podrían estar almacenando y reutilizando información de las trayectorias previas. Sin embargo, no hay evidencia clara que confirme esta teoría, y se debe considerar la posibilidad de que cada ejecución sea totalmente independiente de la anterior.

Conclusión

De los resultados obtenidos, BKPIECE se posiciona como el algoritmo más adecuado en términos de eficiencia de planificación y rapidez de ejecución, logrando un buen balance entre estabilidad y costo computacional. Aunque SPARS consigue una buena longitud de trayectoria tras la estabilización, su tiempo de ejecución es ligeramente mayor y requiere más gestión de nodos dispersos, lo cual lo hace menos eficiente para aplicaciones que necesitan rapidez.

La observación de las desviaciones en las primeras ejecuciones sugiere que, aunque se especula que los algoritmos podrían estar optimizando sobre la marcha, esta hipótesis no está confirmada, y es probable que cada ejecución sea independiente de las anteriores. En este contexto, BKPIE-CE destaca por su rápida convergencia hacia trayectorias consistentes y eficientes, haciéndolo ideal para aplicaciones que priorizan la velocidad y la estabilidad. Por otro lado, LazyPRM, con su enfoque menos estructurado y tiempos de cálculo más altos, resulta menos competitivo para entornos con restricciones de tiempo de planificación y requisitos de consistencia inmediata.

Conclusiones

6.1. Revisión de Objetivos

Para evaluar el éxito del proyecto, revisamos los objetivos establecidos al inicio y el grado en que fueron alcanzados:

- **Objetivo General:** Se planteó desarrollar y validar un sistema de control robótico para entornos simulados y reales, específicamente orientado a la manipulación de materiales en zonas de alta radiación en el contexto del proyecto IFMIF-DONES. El sistema desarrollado permite un control robusto y seguro del brazo robótico, demostrando que es adecuado para el contexto industrial de IFMIF-DONES. La combinación de simulación y pruebas en entornos físicos ha sido efectiva para garantizar una operación segura en ambientes complejos.
- **Objetivos Específicos:**
 - **OE1: Establecimiento de Comunicación mediante ROS:** La comunicación del brazo robótico mediante ROS se estableció exitosamente, permitiendo el control remoto y la integración con otros sistemas. Esta configuración facilita una operación más flexible y adaptable, crucial en un entorno con los requisitos específicos del proyecto IFMIF-DONES.
 - **OE2: Integración y Control en un Entorno Simulado:** Se logró integrar y ejecutar la simulación del brazo robótico en un entorno virtual. Esto permitió observar y analizar el movimiento y comportamiento del brazo en condiciones controladas antes de la implementación física, confirmando que el sistema es capaz de realizar movimientos precisos y coordinados.
 - **OE3: Implementación y Optimización de la Planificación de Movimientos:** La planificación de trayectorias se implementó de manera segura y eficiente, enfocándose en evitar colisiones durante las tareas. Las pruebas en simulación confirmaron que el brazo robótico sigue trayectorias óptimas para realizar tareas complejas, optimizando la operación.
 - **OE4: Validación con el Brazo Robótico Físico en un Entorno Realista:** Se realizaron pruebas en condiciones realistas y se compararon con los resultados obtenidos en simulación, demostrando que el sistema es efectivo y mantiene la precisión requerida en el entorno físico. Esto evidencia que la transición entre el entorno simulado y el real ha sido exitosa.
 - **OE5: Liberación de Código en un Repositorio GitHub:** El código desarrollado se publicó en un repositorio de GitHub, permitiendo su acceso por parte de la comunidad científica. Se documentó adecuadamente para facilitar la colaboración y posibles mejoras por otros desarrolladores, cumpliendo con el objetivo de difusión

y colaboración científica, una práctica importante y poco común en el campo de la robótica.

6.2. Documentación de la Solución en Repositorio Git

La solución se documentó en un repositorio GitHub disponible en el siguiente enlace: <https://github.com/juanantms/TFG>. Este repositorio contiene toda la información detallada en el archivo ‘README’, el cual proporciona una estructura completa para que otros investigadores puedan replicar, adaptar y mejorar el proyecto. El ‘README’ incluye:

- **Instrucciones de Instalación y Configuración:** Pasos detallados para la instalación del sistema y la configuración del entorno de trabajo, especificando las dependencias de software y hardware necesarias.
- **Guía de Uso:** Explicaciones claras y ejemplos sobre cómo ejecutar la simulación y controlar el brazo robótico.
- **Estructura del Código y Desarrollo:** Descripción de la estructura modular del código y las bibliotecas empleadas, lo cual permite a los futuros colaboradores comprender y modificar el funcionamiento del sistema.
- **Configuración de ROS:** Archivos y parámetros necesarios para integrar el sistema con ROS, facilitando la rápida configuración del entorno de comunicación del brazo robótico.

Esta documentación asegura la posibilidad de replicación del proyecto y facilita futuras colaboraciones, incentivando a la comunidad científica y de desarrollo a contribuir y expandir el alcance del proyecto.

6.3. Trabajos Futuros

Para mejorar y expandir las capacidades del sistema de control robótico, se identificaron varias áreas para el trabajo futuro:

- **Mejora de la Precisión del Brazo Robótico:** Basado en los resultados obtenidos en la sección de *Prueba de Precisión*, es necesario trabajar en la mejora de la sensibilidad y precisión del brazo robótico. Esto incluye la investigación y solución del problema del temblor observado durante la ejecución de trayectorias, posiblemente optimizando los parámetros de control y reduciendo tensiones generadas por discrepancias en el modelo y el entorno físico. Se busca garantizar un trazo más suave y preciso, minimizando la vibración y mejorando la estabilidad del sistema.
- **Implementación de Algoritmos de Aprendizaje Automático:** Integrar técnicas de aprendizaje automático para que el brazo robótico pueda adaptar sus movimientos y planificación en tiempo real, lo que podría optimizar aún más la eficiencia y precisión en tareas complejas.
- **Pruebas en Entornos Reales más Desafiantes:** Realizar pruebas en entornos con niveles de radiación similares a los de IFMIF-DONES, evaluando la resistencia y el desempeño del sistema en condiciones extremas para obtener datos más representativos de su comportamiento.

- **Optimización y Creación de una Interfaz de Usuario (UI):** Desarrollar una interfaz gráfica de usuario (GUI) que permita controlar y monitorear el brazo robótico sin la necesidad de utilizar la terminal. Esta interfaz debería facilitar el acceso a funciones clave, como la configuración de trayectorias, la supervisión de parámetros en tiempo real y la interacción remota con el sistema, permitiendo un uso más accesible e intuitivo del sistema, especialmente para operadores no especializados.

6.4. Conclusión Final

En conclusión, el proyecto ha logrado desarrollar un sistema de control robótico avanzado capaz de operar en entornos simulados y reales, cumpliendo los objetivos propuestos para su aplicación en el contexto del proyecto IFMIF-DONES. Este proyecto representa un avance significativo en el control seguro y eficiente de brazos robóticos en ambientes de alta radiación, un desafío importante en la robótica aplicada a la manipulación de materiales en condiciones extremas. La implementación del sistema permite no solo simular y observar el comportamiento del brazo robótico en un entorno virtual, sino también llevar esta funcionalidad a entornos físicos, asegurando que el sistema mantenga su precisión y eficiencia en condiciones reales de trabajo.

El proyecto abarca desde la planificación de trayectorias y optimización de movimientos hasta la configuración de la comunicación mediante el sistema ROS (Robot Operating System), lo que permite controlar el brazo de manera remota y facilita su interacción con otros sistemas. Cada uno de estos componentes ha sido evaluado y ajustado en la simulación y luego validado en condiciones físicas, demostrando que el sistema cumple con los estándares de precisión y seguridad necesarios para operar en entornos industriales complejos.

Además, la publicación del código y la documentación completa en el repositorio de GitHub (<https://github.com/juanantms/TFG>) permite que este proyecto esté disponible para la comunidad científica y técnica. La estructura detallada del código y la guía de instalación, uso y pruebas documentada ofrecen una base sólida para que otros investigadores y desarrolladores puedan replicar el sistema, adaptarlo a sus propias necesidades, o contribuir a su mejora. Esto transforma al proyecto en un recurso abierto que puede contribuir a avances en el campo de la robótica en entornos industriales y de alta radiación, incentivando la colaboración científica y tecnológica.

En el trabajo futuro, se propone el desarrollo de una interfaz gráfica de usuario (GUI) para facilitar el manejo y la accesibilidad del sistema, haciendo que su control sea más intuitivo y accesible para operadores sin experiencia técnica avanzada. Además, la integración de algoritmos de aprendizaje automático permitirá al sistema optimizar sus movimientos en tiempo real, adaptándose a condiciones cambiantes y mejorando su rendimiento en tareas complejas. Estas mejoras no solo incrementarían la eficiencia y seguridad del sistema, sino que también ampliarían su aplicabilidad en entornos industriales que requieren una operación precisa y adaptable.

En resumen, este proyecto constituye un avance en el desarrollo de soluciones robóticas capaces de operar en entornos hostiles, brindando una plataforma funcional que es tanto robusta como escalable. Cumple no solo con los objetivos técnicos propuestos, sino que además abre nuevas posibilidades para investigaciones futuras en la robótica aplicada a sectores de alta demanda técnica. De esta forma, el proyecto no solo tiene un valor práctico inmediato, sino que también establece una base para que las innovaciones en control robótico sigan evolucionando y adaptándose a las necesidades de los entornos industriales más desafiantes.

Bibliografía

- [1] CIEMAT. Proyecto dones - fuente de neutrones para la investigación en fusión. <https://www.fusion.ciemat.es/inicio/dones-2/>.
- [2] Europa Press. Un robot se encargará de limpiar la planta nuclear de fukushima sin que le afecte la radiactividad. <https://www.europapress.es/portaltic/portalgeek/noticia-robot-encargara-limpiar-planta-nuclear-fukushima-le-afecte-rad-20160119162457.html>.
- [3] ESA. El brazo robótico europeo de camino al espacio. https://www.esa.int/Space_in_Member_States/Spain/El_Brazo_Robotico_Europeo_de_camino_al_espacio.
- [4] IFMIF-DONES. Ifmif-dones - demo oriented neutron source project. <https://ifmif-dones.es/>.
- [5] Fusion for Energy. Fusion for energy - news on fusion projects. <https://fusionforenergy.europa.eu/news/?selected-tag=120>.
- [6] Wikipedia. Desarrollo en cascada. https://es.wikipedia.org/wiki/Desarrollo_en_cascada.
- [7] ABEX Excelencia Robótica. Sistema robótico da vinci. <https://www.abexsl.es/es/sistema-robotico-da-vinci/que-esl>.
- [8] MachineMFG. Control methods of industrial robots. https://www.machinemfg.com/es/control-methods-of-industrial-robots/?utm_content=cmp-true.
- [9] Wikipedia contributors. Cinemática inversa. https://es.wikipedia.org/wiki/Cinem%C3%A1tica_inversa.
- [10] Academia Lab. Cinemática inversa. <https://academia-lab.com/enciclopedia/cinematica-inversa/>.
- [11] Object Management Group (OMG). Dds - data distribution service. <https://www.omg.org/omg-dds-portal/>.
- [12] OPC Foundation. Opc ua - open platform communications unified architecture. <https://opcfoundation.org/about/opc-technologies/opc-ua/>.
- [13] Cortex Robotics. Cortex robotics - distributed computing platform. <https://www.cortex-robotics.com/>.
- [14] ROS Wiki Contributors. Ros vs ros 2. <https://design.ros2.org/articles/changes.html>.
- [15] ROS Wiki Contributors. Ros introduction. <https://wiki.ros.org/ROS/Introduction>.

- [16] Wikipedia contributors. Robot operating system. https://es.wikipedia.org/wiki/Robot_Operating_System.
- [17] ROS-Industrial. Ros-industrial: Extendiendo las capacidades de ros al Ámbito industrial. <https://rosindustrial.org/>.
- [18] Robotnik Automation. Ros-industrial: Donde se fusiona ros y el sector industrial. <https://robotnik.eu/es/ros-industrial-donde-se-fusiona-ros-y-el-sector-industrial/>.
- [19] ROS Wiki Contributors. Ros-industrial. <https://wiki.ros.org/Industrial>.
- [20] RViz Wiki. Rviz - ros wiki. <http://wiki.ros.org/rviz>.
- [21] Staubli Robotics Suite (SRS). Staubli robotics suite - simulador para robots staubli. <https://www.staubli.com/us/en/home/robotics-suite.html>.
- [22] Coppelia Robotics. Coppeliasim - robot simulator. <https://www.coppeliarobotics.com/>.
- [23] Cyberbotics Ltd. Webots - open source robot simulator. <https://cyberbotics.com/>.
- [24] Unity Technologies. Unity robotics hub. <https://github.com/Unity-Technologies/Unity-Robotics-Hub>.
- [25] Microsoft Research. Airsim - open source simulator for autonomous vehicles. <https://microsoft.github.io/AirSim/>.
- [26] Gazebo. Gazebo - home. <https://gazebosim.org/home>.
- [27] Gazebo. Gazebo - get started documentation. <https://gazebosim.org/docs/latest/getstarted/>.
- [28] Kautham Project. Kautham project - motion planning toolkit. <https://sir.upc.edu/projects/kautham/Intro.html>.
- [29] Open Motion Planning Library. Ompl - open motion planning library. <https://ompl.kavrakilab.org/>.
- [30] Drake Robotics Library. Drake - toolbox for planning and control. <https://drake.mit.edu/>.
- [31] MoveIt! Wiki. Moveit! - ros wiki. <http://wiki.ros.org/moveit>.
- [32] MoveIt! Moveit - planificación de movimiento para robots. <https://moveit.ai/>.
- [33] Staubli Val3 Driver Documentation. Staubli val3 driver - ros-industrial documentation. https://github.com/ros-industrial/staubli_val3_driver/tree/master.
- [34] Staubli Robotics. Cs9 robot controller. <https://www.staubli.com/global/en/robotics/products/robot-controllers/cs9-robot-controller.html>.
- [35] ROS Wiki. Ros - ros tutorials. <https://wiki.ros.org/ROS/Tutorials>.
- [36] Wikipedia. Ángulos de euler. https://es.wikipedia.org/wiki/%C3%81ngulos_de_Euler.
- [37] Wikipedia. Cuaternión. <https://es.wikipedia.org/wiki/Cuaterni%C3%B3n>.

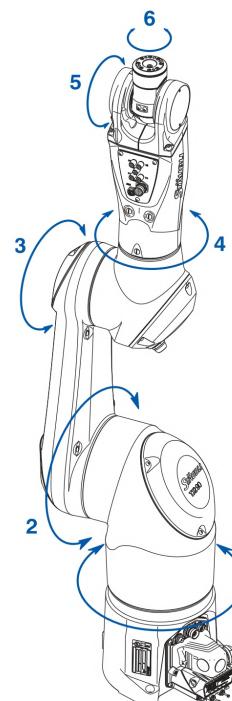
- [38] Wikipedia. Distancia euclídea. https://es.wikipedia.org/wiki/Distancia_euclídea.
- [39] OMPL - Open Motion Planning Library. Rrtconnect - rapidly-exploring random tree. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1RRTConnect.html.
- [40] OMPL - Open Motion Planning Library. Rrtstar - rapidly-exploring random tree. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1RRTstar.html.
- [41] OMPL - Open Motion Planning Library. Lazyprm - probabilistic roadmap method. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1LazyPRM.html.
- [42] OMPL - Open Motion Planning Library. Bkpiece: Bi-directional kpiece. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1BKPIECE1.html.
- [43] OMPL - Open Motion Planning Library. Spars: Sparse roadmap spanners. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1SPARS.html.

Anexo de Imágenes

En este anexo se incluyen imágenes adicionales que complementan la información del brazo robótico Staubli TX2-90.

Characteristics

| | TX2-90 | TX2-90L | TX2-90XL |
|------------------------------|---|------------|-----------|
| Load capacity | 14 kg | 12 kg | 7 kg |
| Reach (between axis 1 and 6) | 1000 mm | 1200 mm | 1450 mm |
| Number of degrees of freedom | 6 | 6 | 6 |
| Repeatability - ISO 9283 | ± 0.03 mm | ± 0.035 mm | ± 0.04 mm |
| Weight | 114 kg | 117 kg | 119 kg |
| UL certification | ✓ | ✓ | ✓ |
| Attachment methods |  | | |



Performance

| | 330°/s | 345°/s | 330°/s |
|--------------------------------------|------------|------------|------------|
| Joint speed -axis 1 | 330°/s | 345°/s | 330°/s |
| Joint speed -axis 2 | 340°/s | 340°/s | 350°/s |
| Joint speed -axis 3 | 430°/s | 420°/s | 410°/s |
| Joint speed -axis 4 | 540°/s | 540°/s | 540°/s |
| Joint speed -axis 5 | 475°/s | 475°/s | 475°/s |
| Joint speed -axis 6 | 760°/s | 760°/s | 760°/s |
| Maximum speed at load gravity center | 10.9 m/s | 11.1 m/s | 11.6 m/s |
| Maximum inertia axis 5 | 1.5 kg.m² | 1.25 kg.m² | 1 kg.m² |
| Maximum inertia axis 6 | 0.25 kg.m² | 0.20 kg.m² | 0.15 kg.m² |
| Brakes | All axes | | |

Work envelope and range of motion

| | | | |
|---|-----------------------|---------|---------|
| Maximum reach between axis 1 and 5 (R. M) | 900 mm | 1100 mm | 1350 mm |
| Minimum reach between axis 1 and 5 (R.m1) | 200 mm | 272 mm | 327 mm |
| Minimum reach between axis 2 and 5 (R.m2) | 256 mm | 320 mm | 391 mm |
| Reach between axis 3 and 5 (R.b) | 425 mm | 550 mm | 650 mm |
| Axis 1 (A) | ± 180° | | |
| Axis 2 (B) | ± 147.5°/-130° | | |
| Axis 3 (C) | ± 145 ° | | |
| Axis 4 (D) | ± 270° | | |
| Axis 5 (E) | + 140°/-115° | | |
| Axis 6 (F) | ± 270° ⁽¹⁾ | | |

(1) Software configurable up to ± 11 250°

Installation environment

| | |
|---|-------------------------------|
| Working temperature according to NF EN 60 204-1 | +5°C to +40°C |
| Humidity according to NF EN 60 204-1 | 30% to 95% max non-condensing |
| Vertical cable outlet version | Available |
| Pressurized version | Available |
| Controller | CS9 |
| Cleanroom standard ISO 14644-1 | 5 |
| Protection class according to EN 60529 | IP65/IP67 |

Cable outlet horizontal and vertical (option)

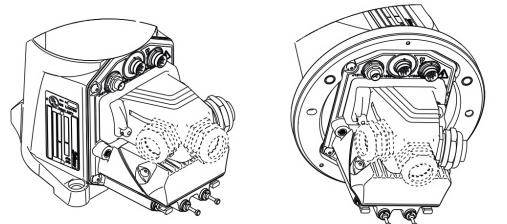
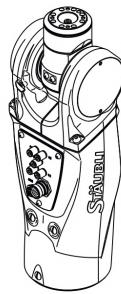


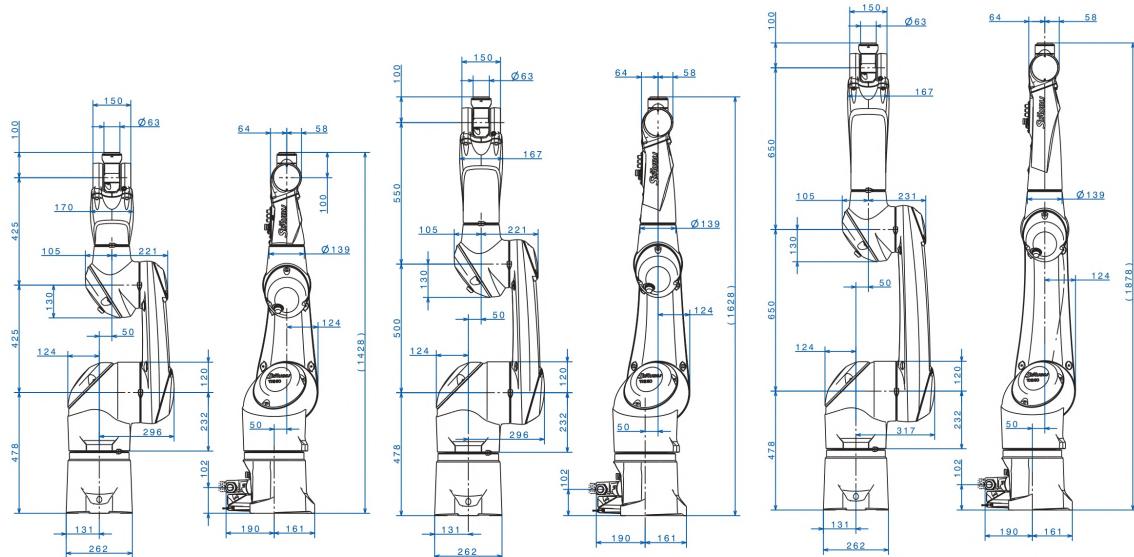
Figura A.1: Datasheet del brazo robótico Staubli TX2-90 (parte 1).

User interface

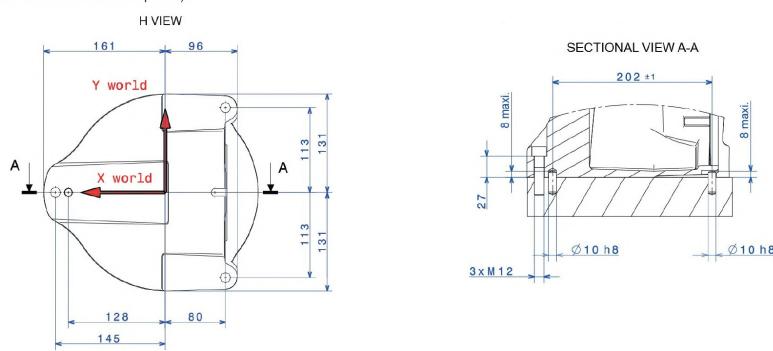
| FOREARM CONNECTIONS | |
|---------------------|--|
| Pneumatic | 2 direct lines between base and forearm or 2 optional solenoid valves: bi-stable solenoid valves 5/2 « pressure » or mono-stable « center close » solenoid valves 5/3 « pressure » or Mono-stable solenoid valves 3/2 « vacuum » |
| Electrical | 10 electrical lines I/O + 3 electrical lines + 1 Multibus cable (4 lines cat 5e lines), DIN M12 |



Dimensions



Mounting (not for vertical cable outlet option)



Sensitive environment versions

| | |
|--|-----------|
| Harsh (Automotive, Metal) | Available |
| ESD (ElectroStatic Discharge) version | Available |
| HE (Humid Environment) | Available |
| Stericlean | Available |
| SCR (Supercleanroom) - class 2 cleanliness ISO 14644-1 | Available |

Available options

| | |
|-----------------------|-------------------------|
| Vertical cable outlet | for smarter integration |
| H1 oil and grease | for hygienic demands |

Figura A.2: Datasheet del brazo robótico Staubli TX2-90 (parte 2).

