Informe Técnico - Technical Report
DPTOIA-IT
julio, 2023

Aplicación de un COBOT para pick-and-place inteligente en ROS

Creada por David Cruz García
Revisada por Vidal Moreno Rodilla y Belén Curto Diego

Departamento de Informática y Automática
Universidad de Salamanca

Revisado por:
Dr. Vidal Moreno Rodilla
Dra. Belen Curto Diego

Información de los Autores:

David Cruz García
david.cruz.garciaa@usal.es

Resumen

En la actualidad, la colaboración entre humanos y robots se ha convertido en una realidad gracias a los grandes avances en la robótica y la inteligencia artificial. Un tipo de robot que ha ganado popularidad en la industria es el cobot, también conocido como robot colaborativo. Estos robots están diseñados para trabajar codo a codo con los humanos, brindando asistencia en tareas repetitivas, pesadas o peligrosas, y optimizando los procesos productivos.

En este trabajo, nos centraremos en el cobot UR3e (Universal Robots 3e), un modelo de robot colaborativo desarrollado por Universal Robots. El UR3e se caracteriza por ser compacto, ligero y fácil de programar, lo que lo convierte en una opción versátil para diferentes aplicaciones industriales y entornos de trabajo.

Además de los frameworks de Robot Operating System (ROS) en sus versiones ROS 1 y ROS 2, el software asociado al cobot UR3e incluye una herramienta fundamental: Polyscope. Polyscope es el software de la tablet del robot, proporcionado por Universal Robots, que permite la interacción y programación del cobot de manera intuitiva y eficiente, dirigida para todo tipo de público y facilitando su uso sin necesidad de conocimientos avanzados de programación.

En conjunto, el uso de Polyscope como software de la tablet del cobot UR3e, junto con los frameworks ROS 1 y ROS 2, proporciona una solución completa y potente para la programación y control del cobot. Esta combinación de herramientas simplifica el proceso de desarrollo, permitiendo a los usuarios centrarse en la lógica de control y la interacción con los sensores y actuadores del UR3e.

En resumen, en este trabajo exploraremos el cobot UR3e y su programación utilizando los frameworks ROS 1 y ROS 2, junto con el software de la tablet del robot, Polyscope. Estudiaremos las capacidades de programación y control que ofrecen estas herramientas, y analizaremos su integración en entornos colaborativos.

Abstract

Today, human-robot collaboration has become a reality thanks to breakthroughs in robotics and artificial intelligence. One type of robot that has gained popularity in the industry is the cobot, also known as a collaborative robot. These robots are designed to work side by side with humans, providing assistance in repetitive, heavy or dangerous tasks, and optimising production processes.

In this paper, we will focus on the UR3e cobot (Universal Robots 3e), a collaborative robot model developed by Universal Robots. The UR3e is compact, lightweight and easy to program, which makes it a versatile option for different industrial applications and work environments.

In addition to the Robot Operating System (ROS) frameworks in its ROS 1 and ROS 2 versions, the software associated with the UR3e cobot includes a fundamental tool: Polyscope. Polyscope is the robot's tablet software, provided by Universal Robots, which allows the interaction and programming of the cobot in an intuitive and efficient way, aimed at all types of public and facilitating its use without the need for advanced programming knowledge.

Overall, the use of Polyscope as the tablet software for the UR3e cobot, together with the ROS 1 and ROS 2 frameworks, provides a complete and powerful solution for programming and controlling the cobot. This combination of tools simplifies the development process, allowing users to focus on the control logic and interaction with the UR3e sensors and actuators.

In summary, in this paper we will explore the UR3e cobot and its programming using the ROS 1 and ROS 2 frameworks, together with the robot's tablet software, Polyscope. We will study the programming and control capabilities offered by these tools, and analyse their integration in collaborative environments.

Índice

Índice de figuras

Indice de tablas

1. Introducción

La industria 5.0 ha surgido como una nueva etapa de la revolución industrial, donde la integración de la tecnología y la colaboración hombre-máquina se convierten en elementos clave para impulsar la productividad en los procesos industriales. En este contexto, los cobots (robots colaborativos) están emergiendo como una solución innovadora y accesible para las pequeñas empresas que buscan mejorar su eficiencia y generar trabajo continuo ininterrumpido, acelerando las ventas y obteniendo mayores beneficios.

Los cobots son robots diseñados para trabajar en colaboración directa con los seres humanos en un entorno de trabajo compartido. A diferencia de los robots convencionales, los cobots son más flexibles, seguros y fáciles de programar. Están diseñados para realizar tareas repetitivas, pesadas o peligrosas, liberando a los trabajadores de actividades monótonas y permitiéndoles centrarse en tareas más creativas y de alto valor añadido. Otro aspecto destacado es la facilidad de implementación de los cobots. A diferencia de los robots tradicionales que requieren una infraestructura y programación compleja, los cobots están diseñados para ser rápidamente configurables y fáciles de usar.

Los robots industriales se han convertido en una parte integral de muchas grandes empresas en los últimos años, y su integración continúa creciendo de manera significativa. Según la Federación Internacional de Robótica (IFR) [1], en 2022 se instalaron mas de medio millón de nuevos robots industriales en todo el mundo, superando en un 22 % el anterior record marcado en 2018 antes de la pandemia. Con esto, ya hay alrededor de 3.5 millones de robots industriales activos. Este crecimiento se puede ver en las figuras 1 y 2 obtenidas de la página de la IFR, donde se puede ver el gran crecimiento en el número de nuevas instalaciones en los ultimos años de forma global y por mercados según el país.

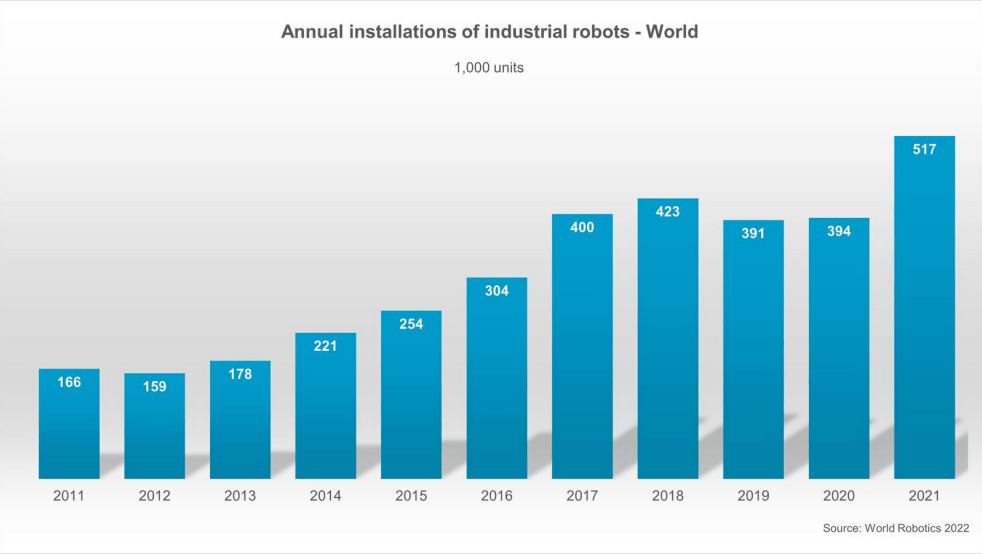

Figura 1: Crecimiento global en la instalación de robots industriales

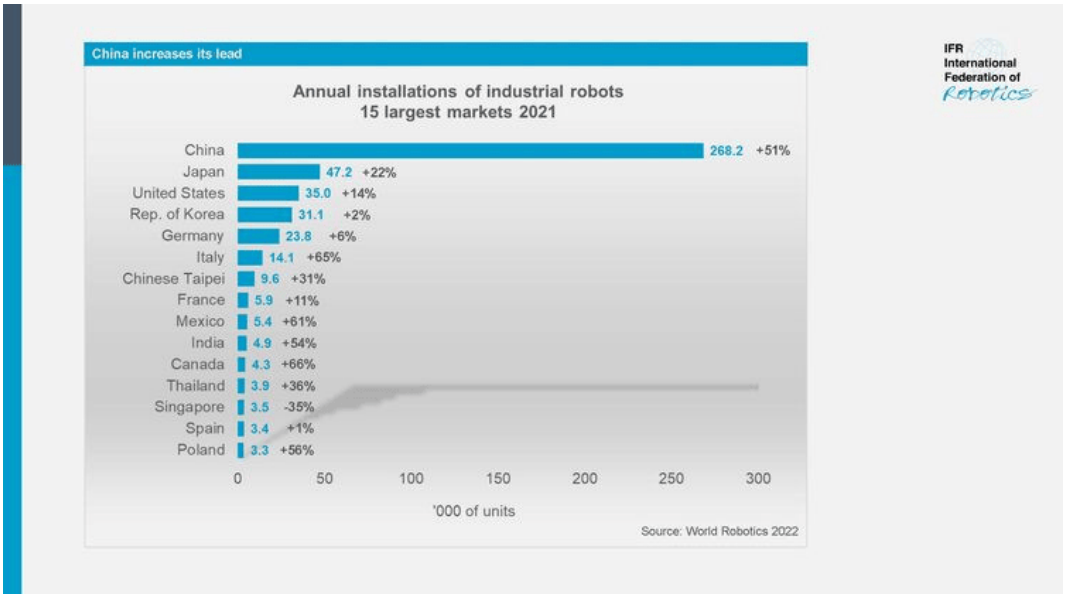

Figura 2: Crecimiento de los mercados por la instalación de robots industriales

Este crecimiento tan pronunciado y el gran desarrollo en los robots industriales ha provocado que muchas empresas decidan actuar e instalar más robots, reduciendo así tiempos de producción y plantilla de trabajadores. Según un estudio de la IFR [2], la densidad media de robots en la industria alcanzó un nuevo record en 2019 de 113 unidades por cada 10000 empleados. En la figura 3 se pueden ver los países que más han optado por el desarrollo tecnológico en controversia con el trabajo humano, convirtiéndose así en los más "automatizados".

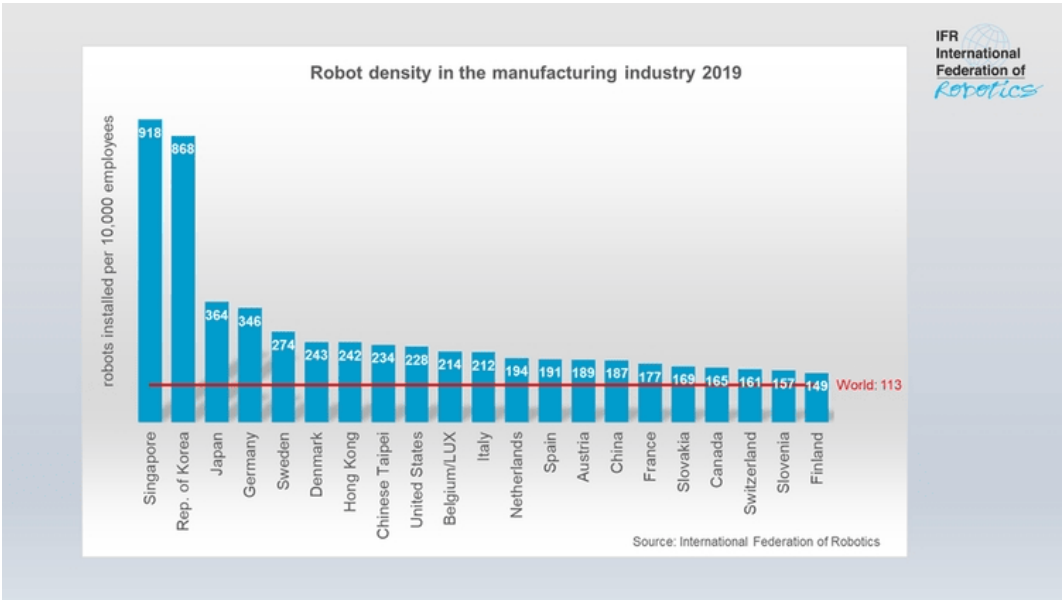

Figura 3: Densidad de robots por cada 10000 empleados

Gracias a todos los incentivos para la digitalización de las empresas en la integración de nuevos robots industriales junto a la IA, los manipuladores robóticos se

están empleando cada vez más en tareas mucho más alejadas de su diseño original para grandes industrias como la automovilística.

El desarrollo de cobots que permiten manipular objetos y trabajar de forma segura en colaboración con los humanos está permitiendo acelerar la eficiencia en los procesos realizados en muchas industrias. Según el siguiente artículo de Cadecobots [3], se espera que el mercado de los cobots haya crecido a una tasa anual del 42 % para el 2026, pasando a un valor de mercado de 7,2 mil millones de euros.

Los robots colaborativos se están convirtiendo en grandes armas de producción, utilizándose en multitud de sectores y áreas como la sanidad, la agricultura o la hostelería fuera del ámbito industrial al que están destinados la mayoría de robots desarrollados. Se ha conseguido fomentar un tipo de robot que permite, de una forma muy sencilla y rápida, adaptarse a una tarea cualquiera (ya sea pick-and-place de objetos, cortado, fresado, etc) y trabajar en conjunto con el humado, multiplicando exponencialmente el rendimiento y la producción en muchos ámbitos.

Gracias a la aparición de estos robots, las empresas buscan facilidades para mejorar los procesos de producción a partir de robots que ofrezcan una programación sencilla y flexible. Con esto, las empresas consiguen que trabajadores con menos cualidades sean capaces de trabajar con junto a estos robots codo con codo. Esto incrementa la producción y la velocidad y agilidad a la hora de servir los productos a sus clientes. Por todo esto, las pequeñas y medianas empresas que invierten más en este tipo de robot consiguen ser mucho más competitivas dentro de sus sectores.

Algunas de las empresas más importantes en la fabricación y uso de estos robots son: Universal Robots con su serie e de robots colaborativos, Kuka o ABB. Estas empresas se han convertido en fabricantes a nivel mundial, centrándose en la fabricación de robots colaborativos con un software muy sencillo e intuitivo para su control.

Universal Robots desarrolla el software Polyscope. Este software está creado específicamente para sus robots colaborativos, y permite realizar tareas complejas desde una programación muy sencilla. Por otro lado, existen otras alternativas como es ROS (Robot Operating System), un framework opensource específico para robótica y que permite integrar casi cualquier robot dentro de entornos simulados y permitiendo multitud de aplicaciones, a cambio de necesitar un alto grado de conocimiento en programación.

En resumen, en este trabajo exploraremos los dos enfoques para el software de control de robots, código abierto y propietario, y se realizará, a ser posible, una tarea inteligente de pick-and-place como el paletizado de objetos para comparar ambos enfoques y las dificultades que surgen en la integración del robot.

2. Objetivos

Este trabajo tiene objetivo estudiar las diferentes opciones software que existen para controlar un cobot de Universal Robots, comparando las diferencias y las posibilidades que ofrecen y estudiando las dificultades que presentan cada una para realizar movimientos con el brazo robótico.

Principalmente, se busca cumplir los siguientes objetivos:

- Realizar un estudio del estado del arte de los diferentes tipos de brazos robóticos y manos o pinzas existentes.
- Realizar un mapping sistemático en busca de artículos que puedan ofrecer información para tratar de alcanzar el objetivo principal del trabajo (integrar el cobot en ROS).
- Investigar y estudiar técnicas y el software necesario para controlar los cobots junto a las pinzas robóticas, así como su funcionamiento para conseguir integrarlos de forma conjunta en ROS y en el software Polyscope de UR.
- Comparar las tres opciones, ROS1, ROS2 y Polyscope, en cuanto al funcionamiento de cada tipo de software y las posibilidades y dificultades que ofrecen. Concretamente, para comparar las tres opciones en la práctica, como caso de estudio, se trabajará con el modelo de cobot UR3e, implementando diversas tareas.

3. Estudio del estado del arte

En este estudio del estado del arte, se llevará a cabo una recopilación general de los avances más relevantes en el campo de la robótica. Se centrará en cuatro aspectos clave: los robots convencionales y los COBOTs, los brazos robóticos, las manos o pinzas robóticas, y el software de control utilizado para manejarlos. Estos elementos representan componentes fundamentales en el diseño y funcionamiento de sistemas robóticos avanzados.

Desde hace años, los robots convencionales o industriales han demostrado su eficacia en procesos de fabricación y manipulación de materiales, mientras que por otro lado, los COBOTs o robots colaborativos han surgido hace algunos años como una solución que permite la interacción segura entre humanos y robots en entornos de trabajo compartidos.

En segundo lugar, se examinarán los brazos robóticos y los avances más recientes en los mismos. Estos brazos mecánicos son capaces de realizar movimientos precisos y controlados, imitando la funcionalidad del brazo humano.

En tercer lugar, se analizarán las manos y pinzas robóticas, que desempeñan un papel crucial en la interacción entre los robots y su entorno. Los manipuladores son sistemas mecánicos que permiten a los robots interactuar con objetos y realizar tareas precisas. Estos elementos mecánicos se utilizan como herramientas añadibles a los brazos robóticos, por lo que ofrecen una gran variedad de aplicaciones en conjunto.

Por último, se revisará el software de control utilizado en los sistemas robóticos. Este software es esencial para la planificación de movimientos, la coordinación de acciones y la toma de decisiones en tiempo real. Se estudiarán en particular ROS (Robot Operating System) como unos de los sistemas operativos más ampliamente usados en la robótica, y Polyscope como software diseñado específicamente para los cobots de Universal Robots.

3.1. Robots convencionales y COBOTs

Los robots convencionales, también conocidos como robots industriales, son sistemas robóticos programables diseñados para realizar tareas automatizadas en entornos industriales [4] [5] [6]. Estos robots se utilizan en una amplia gama de aplicaciones, como ensamblaje, soldadura, pintura, embalaje, manipulación de materiales y más. Están compuestos por un brazo robótico articulado con múltiples grados de libertad y una herramienta final (end effector) que les permite interactuar con el entorno y realizar tareas específicas.

Estos robots han experimentado avances significativos en muchas áreas, lo que ha llevado a mejoras en su rendimiento y capacidades. Estos avances se han logrado gracias a la investigación y desarrollo en varios campos como la inteligencia artificial. A continuación, se presentan algunos de los avances más destacados en los robots

convencionales:

1. Inteligencia Artificial y Aprendizaje Automático [7]: La aplicación de la inteligencia artificial (IA) y aprendizaje automático en los robots convencionales ha permitido mejorar sus capacidades en términos de adaptación, aprendizaje de nuevas tareas. Todo esto gracias a la toma de datos en tiempo real utilizando sensores, por ejemplo cámaras, que permiten al robot conocer en todo momento su entorno, mejorando la toma de decisiones y permitiendo nuevas funcionalidades

2. Interacción persona-máquina [8]: En busca de mejorar la productividad de los robots, se han realizado esfuerzos para mejorar la interacción entre humanos y los robots convencionales. Esto implica el desarrollo de interfaces intuitivas y seguras que permiten una colaboración más estrecha y sencilla con el usuario, facilitando la realización de tareas y la resolución de problemas que presentaban dificultades a estos robots. Gracias a esta idea, surgieron los robots colaborativos o COBOTs.

3. Flexibilidad y Adaptabilidad [9]: Los robots convencionales siempre han presentado un rigidez fuerte frente a la resolución de problemas. Estos robots están diseñados para realizar una única tarea específica, presentando dificultades para adaptarse a otro tipo de tareas. Es por esto que los robots convencionales están tratando de ser diseñados para ser más flexibles y adaptables, lo que les permite ajustarse rápidamente a diferentes tareas y cambios en el entorno de trabajo mediante el uso de sistemas de visión avanzados, sensores y algoritmos de planificación de movimientos más eficientes.

En la figura 4 se muestran dos ejemplos de robots convencionales [10]: IRB-6, un robot lanzado en la década de 1980 y que se consideró uno de los más compactos y ligeros en su época; y SCARA (Selective Compliance Assembly Robot Arm), también de la época de los años 1970-1980, se encargaba de tareas de ensamblaje por su rigidez en su eje vertical y su buen desempeño en el horizontal.

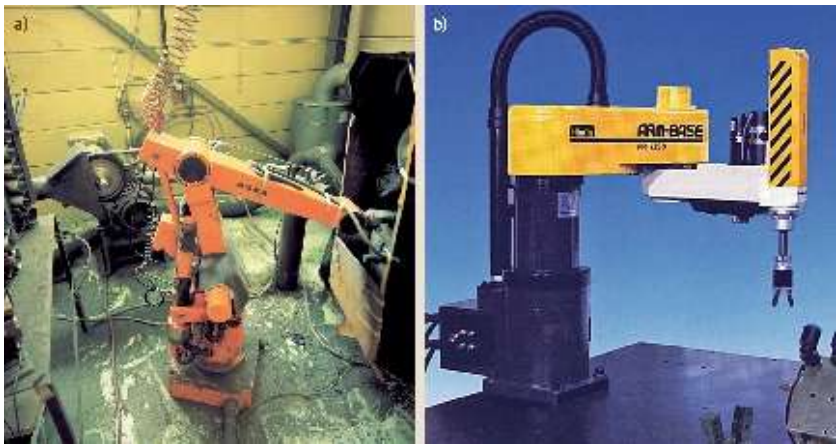

Figura 4: Robots convencionales: IRB-6 y SCARA

Gracias a la importancia que cobraron estos robots convencionales dentro de la industria, surgió la necesidad desarrollar evoluciones. Entre ellas, surgió el concepto de interacción humano-robot, donde se buscaba conseguir una colaboración entre ambos para realizar una tarea concreta, mejorando la eficiencia del humano gracias a la ayuda del robot y solventando problemas de una forma rápida gracias a la ayuda del humano al robot. Con esta idea surgió el concepto de COBOT o robot colaborativo.

Los COBOTs o robots colaborativos, son una clase especial de robots diseñados para trabajar en colaboración con los seres humanos en entornos de trabajo compartidos. A diferencia de los robots convencionales, que generalmente operan de manera independiente en áreas separadas, los COBOTs están diseñados para interactuar y colaborar de forma segura y eficiente con las personas.

La introducción de los cobots ha revolucionado la forma en que los robots se integran en los procesos industriales [11]. Estos robots colaborativos están diseñados con características y capacidades que permiten una interacción segura y efectiva con los trabajadores humanos, brindando beneficios como la mejora de la productividad, la flexibilidad en la producción y la reducción de riesgos laborales.

Según Sabatini et al. (2015) [12], *"destaca que los cobots en entornos industriales requieren consideraciones importantes en términos de seguridad, interacción y confianza en la colaboración entre humanos y robots"*. Esto quiere decir que aunque los robots colaborativos representan una gran mejora en cuando a los robots convencionales, también requieren de una mayor seguridad y flexibilidad, que permitan al humano trabajar de forma segura, cómoda y confiable con el robot.

El desarrollo de los robots colaborativos ha tomado una gran fuerza en los últimos tiempos. Universal Robots puede presentarse como una de las mayores creadoras y proveedora de estos robots. Ofrecen un amplia gama de productos y tipos de robots colaborativos. Esta empresa ha creado robots como el UR3e, un brazo robótico colaborativo y que será el pilar de la investigación realizada en este documento. Se puede ver este robot en la figura 5.

3.2. Brazos robóticos

Un brazo robótico es un componente fundamental en la industria de la robótica, diseñado para realizar movimientos precisos y controlados, imitando la funcionalidad del brazo humano. Se trata de un sistema mecánico que está compuesto por una serie de articulaciones conectadas que permiten la realización de tareas específicas de manipulación de objetos. Los brazos robóticos se utilizan en ensamblaje, manipulación de materiales, operaciones quirúrgicas, exploración espacial y muchas otras aplicaciones.

Existen diferentes tipos de brazos robóticos, cada uno con características y capacidades particulares. Aunque existe una gran diversidad de variaciones, algunos de los tipos más comunes de brazos robóticos son:

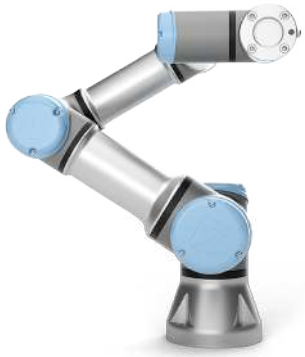

Figura 5: Robots colaborativo UR3e

- Brazos articulados: Surgieron en los años 60 y consisten en una serie de segmentos articulados, conectados por juntas giratorias, simulando las articulaciones humanas. Cada junta proporciona un grado de libertad y permite movimientos en diferentes direcciones. Los brazos articulados son conocidos por su versatilidad y capacidad de movimiento. [13].
- Brazos cartesianos: Estos brazos robóticos se mueven a lo largo de ejes lineales ortogonales, es decir, los ejes son perpendiculares entre sí. Utilizan un sistema de coordenadas cartesianas para controlar los movimientos en los ejes X, Y y Z, por lo que son utilizados para aplicaciones que requieren movimientos lineales precisos. [14]
- Brazos cilíndricos: Tienen diseño en forma de cilindro, con una articulación giratoria en la base y un brazo extensible en forma de varilla. Los brazos cilíndricos permiten movimientos de rotación y extensión, y son comúnmente utilizados en aplicaciones que requieren un alcance extendido [15].

Ya que el robot utilizado es el UR3e, un robot articulado, y siguiendo el ejemplo de antes, otro ejemplo de brazo robótico articulado es el robot SCARA [13] como se puede ver en la figura 6.

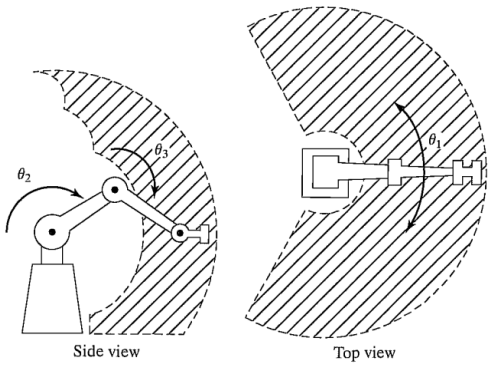

Figura 6: Brazo robótico articulado SCARA

3.3. Manos o pinzas robóticas

Una mano o pinza robótica es un componente esencial para los sistemas robóticos, ya que les permiten agarrar y manipular objetos de diversas formas y tamaños. Estas herramientas desempeñan un papel fundamental en aplicaciones como ensamblaje, manipulación de objetos, recolección y clasificación. [16]

Existen varios tipos de manos y pinzas robóticas, cada una diseñada para satisfacer unos requisitos de agarre y manipulación. Algunos ejemplos destacados son:

1. Garras paralelas: Dos o más dedos paralelos que se abren y cierran para agarrar objetos. Son especialmente útiles en tareas que implican agarre firme y seguro, como la paletización de cajas o la manipulación de objetos robustos. [17].

2. Pinzas con forma de gancho: Estas pinzas tienen una forma en gancho que les permite envolver objetos y sostenerlos de manera segura. Son adecuadas para objetos con formas irregulares o asimétricas que puedan ser elevados desde un tirador. [18].

3. Manos antropomórficas: Estas manos imitan la estructura y los movimientos de la mano humana, utilizando varias articulaciones y dedos independientes. Permiten un agarre más adaptativo y son ideales para tareas que requieren una manipulación delicada o precisa, como el ensamblaje electrónico. [19].

4. Manos neumáticas: Utilizan sistemas de aire comprimido para generar fuerza y agarre con dedos y ventosas que se adaptan a diferentes formas y tamaños de objetos. Son comúnmente utilizadas en la industria de la automatización para la manipulación de objetos frágiles o sensibles, ya que pueden aplicar una presión controlada durante el agarre. Con esto se consigue trabajar con objetos delicados de una forma rápida y segura. [20]

5. Manos magnéticas: Emplean la manipulación magnética para el agarre de objetos metálicos o que puedan ser trabajados utilizando un imán o electroimán.

Utilizan campos magnéticos para controlar y manipular objetos. Son especialmente útiles en el sector hospitalario para cirugía no invasiva donde se debe tratar. [21].

Aunque existen más tipos de manos y pinzas robóticas, cada una de ellas está diseñada para un tipo de tarea específica. Para este caso, nos centraremos en las garras paralelas.

Este tipo de pinza robótica puede trabajar utilizando sensores en los dedos que proporcionan retroalimentación durante la manipulación de objetos: pueden detectar fuerzas, medir el contacto con un objeto y ajustar el agarre de forma dinámica según se requiera.

Algunos tipos de sensores utilizados son los siguientes:

1. Sensores de fuerza: Estos sensores miden las fuerzas ejercidas sobre los dedos o garras de la mano robótica. Proporcionan información precisa sobre la fuerza aplicada durante el agarre de un objeto, lo que permite al robot ajustar su fuerza de agarre en función de las necesidades específicas de la tarea. . [22].

2. Sensores táctiles: Los sensores táctiles están diseñados para detectar el contacto físico entre la mano robótica y los objetos. Pueden utilizar diferentes tecnologías, como sensores de presión, sensores capacitivos o sensores ópticos, para medir la distribución de la presión o el contacto en la superficie de los dedos. Estos sensores permiten al robot tener una percepción táctil similar a la de un ser humano y obtener información sobre el objeto que está manipulando. [23].

3. Microsensores: Son dispositivos de pequeño tamaño que se pueden integrar en las manos robóticas paralelas para proporcionar una retroalimentación de alta resolución y detección de fuerzas en puntos específicos. Estos sensores permiten al robot obtener información detallada sobre la distribución de fuerzas a lo largo de los dedos y adaptar su agarre de manera precisa. Se utilizan en aplicaciones donde la escala de fuerzas a detectar o aplicar es muy pequeña, como por ejemplo, aplicaciones dentro del sector sanitario. [24].

La empresa Universal Robots también desarrolla pinzas y manos robóticas que son compatibles con sus brazos robóticos colaborativos. Desarrollan pinzas de varios tipos y utilizando las diferentes tecnologías explicadas en el punto anterior.

Por ejemplo, una garra paralela con sensores de fuerza en los dedos puede ser la pinza RG2 de Universal Robots. Esta pinza está diseñada para levantar carga de hasta 2kg, proporcionando retroalimentación de agarre y fuerza gracias a sus sensores, lo que permite manejar de forma consistente y controlada todo tipo de objetos. [16]. Se puede ver esta pinza en la figura 7.

Figura 7: Garra paralela Universal Robots RG2.

3.4. Software de control

Un software de control de robots es una herramienta fundamentale para programar, supervisar y controlar las operaciones de los robots industriales. Este tipo de software proporciona interfaces intuitivas y funciones avanzadas que permiten a los usuarios interactuar con los robots y definir sus movimientos y comportamientos, permitiéndoles realizar casi cualquier tarea de manipulación de objetos.

Los primeros softwares de control de robots convencionales y/o COBOTs fueron desarrollados hace varias décadas. En los inicios de la robótica, no existían softwares específicos para controlar robots industriales. En su lugar, se utilizaban sistemas de control basados en cables y levas, como en el caso del Unimate, considerado el primer robot industrial, desarrollado por George Devol y Joseph Engelberger en 1961 [25]. A medida que la tecnología robótica avanzaba, surgieron lenguajes de programación como el VAL (Versatile Assembly Language) en la década de 1970, que se utilizaba para controlar robots manipuladores [26].

Los software de control antiguos para robots convencionales solían tener interfaces de usuario más limitadas y complejas, requiriendo programación detallada, compleja y manual mediante comandos específicos. En la actualidad, los software modernos ofrecen interfaces gráficas intuitivas y amigables, simplificando la programación a través de interfaces visuales y lenguajes de programación más intuitivos. Además, los software de control de robots modernos cuentan con potentes capacidades de simulación y planificación, permitiendo a los usuarios probar y optimizar programas de control en entornos virtuales antes de su implementación, algo que en el pasado era inviable por las limitaciones de potencia de los dispositivos empleados.

En el caso de los cobots, los software modernos se centran en la colaboración y la seguridad, proporcionando funciones avanzadas de detección de colisiones, control de fuerza y adaptabilidad para garantizar que la interacción robot-humano sea segura y eficiente, evitando daños personales y mejorando la calidad del trabajo realizado.

Estas diferencias reflejan los avances tecnológicos y las necesidades cambiantes

de la industria, haciendo que los software modernos sean más accesibles, eficientes y seguros en la programación y control de robots convencionales y cobots.

En la actualidad, se dispone de una amplia gama de softwares para controlar robots convencionales y cobots, diseñados para satisfacer las necesidades específicas de cada aplicación y fabricante. Algunos ejemplos de softwares modernos son:

1. ROS (Robot Operating System): ROS es una plataforma de software de código abierto ampliamente utilizada en la comunidad robótica. Fue desarrollada inicialmente en 2007 por el Laboratorio de Inteligencia Artificial de Stanford (Stanford AI Lab) y actualmente cuenta con una comunidad activa de desarrolladores que contribuyen con módulos y herramientas [27].

2. Siemens TIA Portal: TIA Portal es un entorno de programación y control utilizado para los robots de Siemens. Proporciona una interfaz gráfica intuitiva y herramientas avanzadas para la programación, simulación y monitoreo de los robots de la compañía [28].

3. ABB RobotStudio: RobotStudio es un software de programación y simulación utilizado para los robots de ABB. Permite la creación y simulación de programas de control, así como la optimización y visualización de las tareas de los robots en un entorno virtual [29].

4. Fanuc RoboGuide: RoboGuide es una herramienta de simulación y programación utilizada para los robots de Fanuc. Permite la creación, simulación y optimización de programas de control, así como la generación de visualizaciones 3D de las tareas planificadas. [30]

5. Universal Robots Polyscope: Universal Robots es conocido por sus cobots colaborativos, y el software de control utilizado para ellos es Polyscope. Según un artículo publicado en la revista ÏEEE Robotics & Automation Magazine", el software Polyscope proporciona una interfaz de programación intuitiva y basada en iconos que permite a los usuarios configurar tareas, movimientos y lógica de control de forma sencilla [31].

Entrando un poco más en detalle en cada uno, se van a explicar las características de cada uno de ellos.

3.4.1. ROS (Robot Operating System)

ROS (Robot Operating System) es marco de trabajo flexible de código abierto diseñado para facilitar el desarrollo de software para robots. Además es muy flexible ya que proporciona bibliotecas y herramientas para ayudar a los desarrolladores a crear aplicaciones robóticas. [32]

ROS se basa en una arquitectura de comunicación distribuida, en la cual los diferentes componentes del sistema, llamados nodos, se comunican entre sí a través

de mensajes. Estos nodos pueden ejecutarse en diferentes máquinas, lo que permite una mayor flexibilidad y escalabilidad en el desarrollo de sistemas robóticos complejos, además de proporcionar servicios como la gestión de paquetes, la simulación y la visualización de entornos completos, lo que facilita el desarrollo de aplicaciones robóticas y permite ahorrar fallos y costes sobre robots reales.

ROS se ha convertido en una plataforma ampliamente utilizada y popular en la comunidad robótica debido a varias razones. En primer lugar, al ser de código abierto fomenta la colaboración entre los investigadores y desarrolladores robóticos de todo el mundo. Además, ROS ofrece una amplia gama de bibliotecas y herramientas que abordan problemas comunes en robótica, como la percepción, la planificación de movimiento, la interacción con el entorno o la simulación.

Una de las ventajas clave de ROS es su modularidad y su capacidad para la reutilización de software. Los desarrolladores pueden utilizar los paquetes y componentes de otros desarrolladores evitando tener que reinventar la rueda en cada proyecto robótico. Esto acelera el desarrollo, reduce los costos y fomenta la colaboración y el intercambio de ideas en la comunidad robótica.

3.4.2. Siemens TIA Portal

Siemens TIA Portal es un entorno de programación y control utilizado para los robots de Siemens. Proporciona una interfaz gráfica intuitiva y herramientas avanzadas para la programación, simulación y monitoreo de los robots de la compañía. TIA Portal integra múltiples funciones, como la configuración de hardware, la programación de controladores y la visualización de datos en un único entorno de desarrollo. [33]

El objetivo principal de Siemens TIA Portal es mejorar la eficiencia y la productividad en el desarrollo y mantenimiento de sistemas de automatización para los robots de su compañía. Con TIA Portal, los ingenieros pueden programar y configurar robots y otros dispositivos de automatización de manera más rápida y sencilla, lo que reduce el tiempo de desarrollo y puesta en marcha de los sistemas.

Además, TIA Portal ofrece una amplia gama de características avanzadas, como la simulación de procesos, la detección de colisiones y la visualización en 3D, lo que permite probar y optimizar las tareas de los robots antes de implementarlas en el entorno de producción.

3.4.3. ABB RobotStudio

ABB RobotStudio es un software de programación y simulación utilizado para los robots de ABB. Permite a los usuarios crear, simular y optimizar programas de control, así como visualizar y supervisar las tareas de los robots en un entorno virtual. RobotStudio es una herramienta integral que agiliza el proceso de desarrollo y puesta en marcha de sistemas robóticos. [34]

ABB RobotStudio ofrece una interfaz de usuario intuitiva y funciones avanzadas

para la programación de robots. Los usuarios pueden utilizar un lenguaje de programación específico de ABB, así como opciones de programación basada en flujo de trabajo y programación en línea. Además, el software permite la simulación en 3D de las tareas planificadas, lo que permite detectar posibles colisiones y optimizar el rendimiento antes de implementar los programas en el mundo real.

La capacidad de simulación y visualización de RobotStudio es una característica destacada. Los usuarios pueden crear entornos virtuales que replican las condiciones de trabajo reales, lo que facilita la evaluación de la eficiencia y la seguridad de las operaciones robóticas. Esta función reduce los errores y los tiempos, lo que ahorra costos y mejora la productividad.

3.4.4. Fanuc RoboGuide

Fanuc RoboGuide es una herramienta deprogramación de robots utilizada para los robots de la empresa Fanuc. RoboGuide permite a los usuarios crear, simular y optimizar programas de control, así como generar visualizaciones 3D de las tareas planificadas. Esta herramienta facilita el desarrollo y la implementación de sistemas robóticos, mejorando la eficiencia y la precisión de las operaciones. [35]

Fanuc RoboGuide ofrece funcionalidades avanzadas para la programación de robots. Los usuarios pueden utilizar un lenguaje de programación específico de Fanuc (TPE), así como opciones de programación basada en iconos y programación en línea. Además, el software proporciona capacidades de simulación en tiempo real, lo que permite a los usuarios visualizar y evaluar el comportamiento de los robots antes de la implementación en el entorno de producción.

La simulación en 3D es una característica destacada de RoboGuide. Los usuarios pueden crear modelos virtuales de robots y entornos de trabajo, lo que les permite detectar y corregir posibles colisiones, optimizar los movimientos y verificar la eficiencia de las trayectorias.

3.4.5. Universal Robots Polyscope

Universal Robots Polyscope es el software utilizado para controlar los robots colaborativos de Universal Robots. Polyscope proporciona una interfaz de programación intuitiva basada en iconos y permite la configuración de tareas, movimientos y lógica de control de manera sencilla. Este software se centra en la colaboración y la seguridad, brindando funciones avanzadas de detección de colisiones, control de fuerza y adaptabilidad para garantizar una interacción segura con los trabajadores. Para ello ofrece la capacidad de detectar cualquier tipo de colisión o indicente gracias a la gran cantidad de sensores que incorpora, permitiéndoles detectar fuerzas aplicadas sobre cualquier parte del robot o incluso controlarlo de forma libre en caso de error. [36]

La interfaz de programación de Polyscope es altamente visual y fácil de usar, lo que permite a los usuarios programar y personalizar las tareas del cobot de manera

rápida y eficiente. El software ofrece una amplia gama de funciones y comandos predefinidos, lo que simplifica la creación y modificación de programas sin requerir conocimientos de programación avanzados.

Una característica destacada de Polyscope es su enfoque en la seguridad y la colaboración. El software está diseñado para garantizar la detección de colisiones y la protección de los trabajadores, además de ser increíblemente sencillo de utilizar, permitiendo una interacción directa y segura con los cobots. Además, Polyscope ofrece control de fuerza y funciones de adaptabilidad que permiten a los cobots ajustar automáticamente su comportamiento en respuesta a los cambios en su entorno de trabajo.

3.5. Mapping sistemático

El mapping sistemático es una metodología que se utiliza para explorar, analizar y sintetizar la información existente en un campo de estudio y visualizarla de manera clara utilizando para ello herramientas visuales que pueden representar las relaciones y conexiones entre los diferentes elementos. En el ámbito de los manipuladores y brazos robóticos, esta técnica de investigación adquiere un papel fundamental para comprender y evaluar los avances, enfoques y tendencias relacionadas con la interacción humano-robot (COBOTs) y su aplicación en el diseño y desarrollo de plataformas interactivas.

El objetivo principal de la realización de este mapping sistemático es la de recopilar y analizar la información disponible en el área de los brazos robóticos y manipuladores, centrando la investigación en el brazo robótico UR3e y la pinza RG2 de Universal Robots, utilizando como software de control ROS. Este mapeo está enfocado únicamente a la investigación sobre la integración del robot en el sistema operativo ROS para posteriormente compararlo con el software Polyscope. Esto es porque se quiere investigar sobre la integración del robot en un marco de trabajo genérico para robots como es ROS, para posteriormente compararlo con Polyscope, un software específico para este brazo robótico.

3.5.1. Metodología

Para realizar este mapeo sistemático, se ha utilizado Parsifal como herramienta para el manejo de la trazabilidad del proceso. Mediante el uso de Parsifal, se aplicará el proceso PICOC (Population, Intervention, Comparison, Outcome, Context) para formular preguntas de búsqueda que faciliten la tarea de realizar el mapeo sistemático. Gracias a estas preguntas se han obtenido una lista de artículos que se escogerán para aceptar como válidos aquellos que respondan a las preguntas planteadas. Para ello, se seguirá la metodología PRISMA para la selección de los artículos utilizando para ello el diagrama de flujos. De esta manera, se obtendrán los artículos finales que se utilizarán en este mapeo sistemático de la literatura relacionado con brazos robóticos y manipuladores.

3.5.2. Preguntas planteadas

En primer lugar, se plantean las siguientes preguntas para el proceso de mapping. Se buscarán artículos relacionados con el área de los brazos robóticos y manipuladores que puedan resolver las siguientes preguntas:

- RQ1. ¿Cuál es el impacto de los robots industriales, los robots manipuladores, los brazos robóticos, las manos articuladas y los cobots en el contexto de la industria?
- RQ2.¿Cuál es la eficacia de ROS, ROS1 y ROS2, para mejorar el rendimiento de robots industriales, robots manipuladores, brazos robóticos, manos articuladas y cobots?
- RQ3. ¿Cuáles son las diferencias en los resultados al comparar el uso de ROS en robots industriales, robots manipuladores, brazos robóticos, manos articuladas y cobots?
- RQ4. ¿Qué efectos tiene la incorporación de cámaras, mecanismos de pick and place y control de fuerzas en el rendimiento de robots industriales, robots manipuladores, brazos robóticos, manos articuladas y cobots?
- RQ5. ¿Cuáles son los factores que influyen en la implantación y utilización de robots industriales, robots manipuladores, brazos robóticos, manos articuladas y cobots en relación con ROS, así como en el uso de cámaras, mecanismos de pick and place y control de fuerzas?

3.5.3. Criterios de inclusión y exclusión

Los criterios de inclusión y exclusión permiten filtrar los artículos imponiendo condiciones que deben cumplir los mismos. Con estos criterios se descartan multitud de artículos que no traten del tema de los brazos robóticos o manipuladores o incluso que no estén en un idioma concreto o un periodo de tiempo determinado. Los criterios de inclusión establecerán las condiciones que deben cumplir los artículos para ser escogidos, mientras que los de exclusión establecen que condiciones no debe tener un artículo.

Los criterios de inclusión aplicados son los siguientes:

- IC1.Artículos centrados en robots industriales, robots manipuladores, brazos robóticos, manos articuladas o cobots.
- IC2.Artículos que mencionen o traten específicamente ROS.
- IC3.Artículos que traten los temas de cámaras, mecanismos de pick and place o control de fuerzas en el contexto de la robótica.
- IC4.Artículos publicados en un periodo de tiempo determinado (por ejemplo, los últimos 5 años).

- IC5.Artículos disponibles en inglés o español.
- IC6.Artículos accesibles a través de la Universidad de Salamanca.
- IC7,Artículos que tratan los temas de esta investigación e incluyen los términos de búsqueda yresponden a las RQs.

Los criterios de exclusión aplicados son los siguientes:

- EC1.Artículos que no se refieran específicamente a robots industriales, robots manipuladores, brazos robóticos, manos articuladas o cobots.
- EC2.Artículos que no mencionen o traten ROS.
- EC3.Artículos que no traten los temas de cámaras, mecanismos de pick and place o control de fuerza en el contexto de la robótica.
- EC4.Artículos publicados fuera del plazo especificado.
- EC5.Artículos que no estén disponibles en inglés o español.
- EC6.Artículos que no sean accesibles a través de la Universidad de Salamanca.
- EC7,Artículos que tratan los temas de esta investigación e incluyen los términos de búsqueda, pero no responden a las RQs

3.5.4. Términos PICOC y cadena de búsqueda

Una vez definidas las preguntas y los criterios de inclusión y exclusión, se comienza a buscar artículos de investigación relacionados con el campo de los brazos robóticos y manipuladores.

Para ello, se obtendrán los artículos desde el 2015 hasta 2023 siguiendo los criterios establecidos en los puntos anteriores. Para la búsqueda de los artículos, se han elegido las siguientes palabras como keywords siguiendo el método PICOC: *industrial robot, manipulator robot, robotic arm, articulated hand, cobot, ROS, ROS1, ROS2, camera, pick and place, force control.*.

Estas palabras las podemos separar según el método PICOC. Esta separación se muestra en la tabla 1.

Web	Consulta
Population	industrial robot, manipulator robot, robotic arm, articulated hand, cobot
Intervention	ROS, ROS1, ROS2
Comparison	
Outcome	camera, pick and place, force control
Context	

Tabla 1: Keywords según metodología PICOC

Con estas keywords establecidas se define la siguiente consulta que se aplicará en los diferentes buscadores de las webs de artículos:

(industrial AND robot OR manipulator AND robot OR robotic AND arm OR articulated AND hand OR robot) AND (ros OR rose OR rose) AND (camera OR pick and place OR force control)

3.5.5. Resultados de la búsqueda inicial

Adaptando esta consulta a cada uno de las webs, obtenemos las siguientes consultas mostradas en la tabla 2.

Web	Consulta
Scopus	TITLE-ABS-KEY ((industrial AND robot OR manipulator AND robot OR robotic AND arm OR articulated AND hand OR robot) AND (ros OR rose OR rose) AND (camera OR pick and place OR force control))
Web of Science	(TS=("industrial robot" OR "manipulator robot" OR "robotic arm" OR "articulated hand" OR "cobot")) AND (TS=("ROS" OR "ROS1" OR "ROS2") AND (TS=("camera" OR "pick and place" OR "force control")))
IEEE Xplore	("All Metadata":industrial robot OR "All Metadata":manipulator robot OR "All Metadata":robotic arm OR "All Metadata":articulated hand OR "All Metadata":cobot) AND ("All Metadata":ROS OR "All Metadata":ROS1 OR "All Metadata":ROS2) AND ("All Metadata":camera OR "All Metadata":pick and place OR "All Metadata":force control)

Tabla 2: Consultas realizadas

Los resultados obtenidos se han descargado en formato BibTex e importados en Parfisal. Con esto, se han obtenido un total de 203 artículos, 145 de IEEE Xplore, 36 de Web of Science, 22 de Scopus, distribuidos según la figura 8.

Tras seleccionar los artículos iniciales, se ha seguido la metodología PRISMA para filtrar los artículos. Para ello, se han eliminado artículos duplicados y se han filtrado

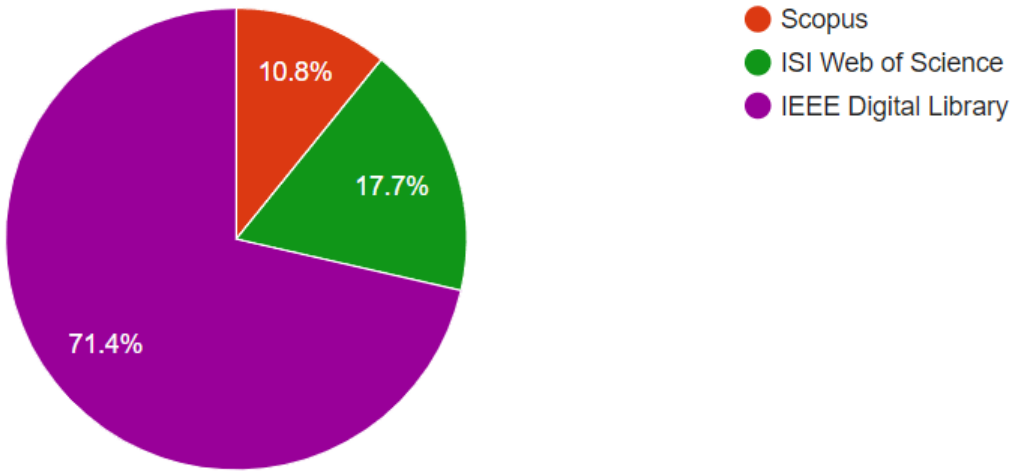

Figura 8: Distribución de artículos iniciales

por artículos que no vinieran de congresos o que fuera suficientemente relevantes, y tratando de buscar palabras clave en los títulos como pueden ser robot, manipulador, ROS, o similares. De estos artículos filtrados, se han elegido los artículos que, tras someterlos a una evaluación de calidad, han obtenido la mejor puntuación. Esta evaluación de la calidad se ha hecho a través de las siguientes preguntas:

- ¿Está el objetivo de la investigación claramente expuesto y bien definido?
- ¿Están los componentes y paquetes ROS utilizados en la investigación claramente expuestos y adecuadamente documentados?
- ¿Hay una descripción clara del manipulador o brazo robótico utilizado en el estudio?
- ¿Se explican claramente el montaje y las condiciones experimentales?

3.5.6. Diagrama PRISMA

Con estas preguntas, se evalúa la calidad de cada artículo con una calificación del 0 al 5, y se seleccionan aquellos que tengan una nota mayor a 4. Finalmente, se seleccionan 12 artículos que se incluyen para el análisis del campo de los brazos robóticos y manipuladores.

El esquema PRISMA se muestra en la figura 9.

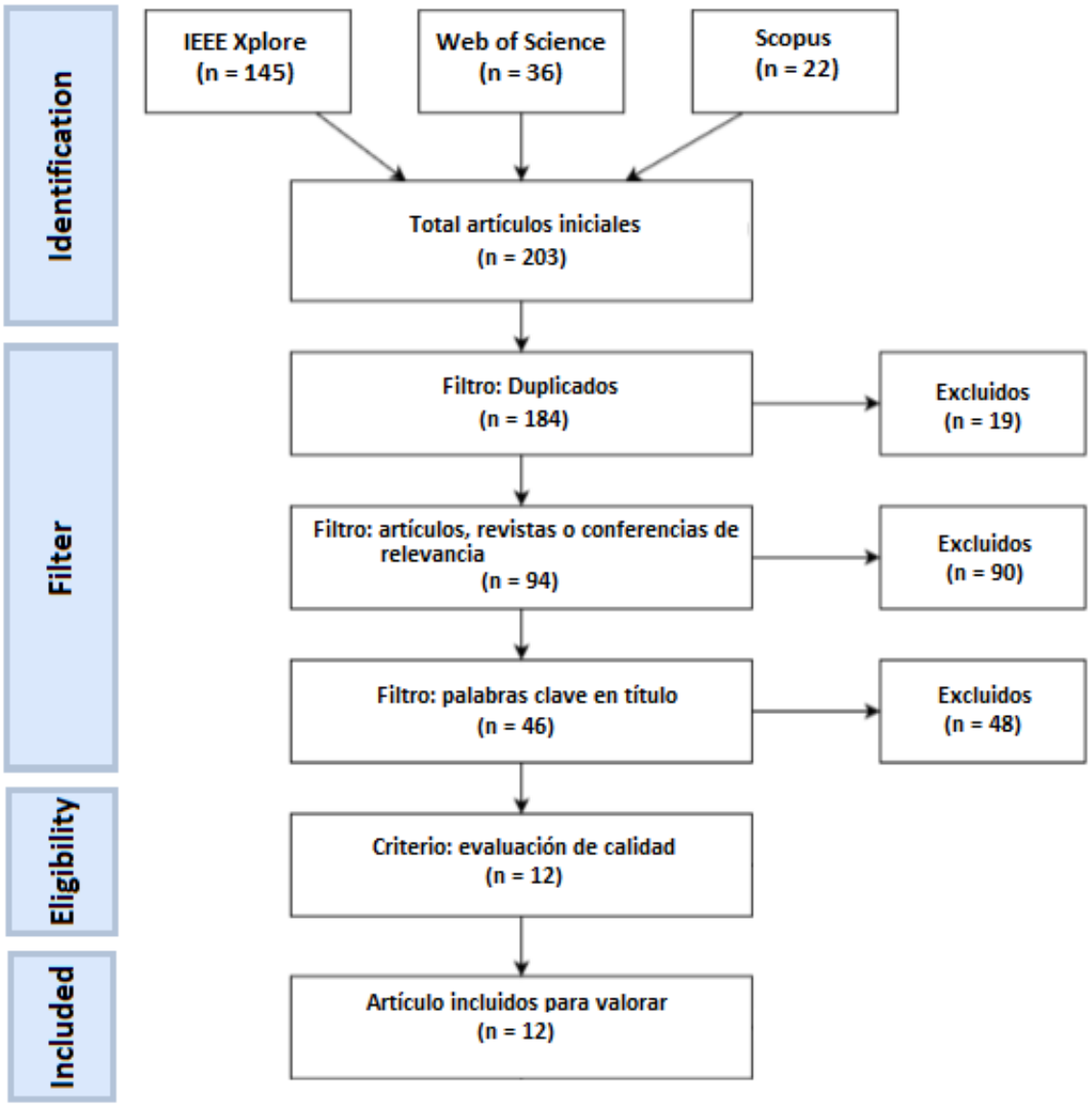

Figura 9: Esquema PRISMA

3.5.7. Análisis de los artículos y toma de decisiones

En general, los 12 artículos se centran en temas como el control y la programación de brazos robóticos, la aplicación de ROS en el desarrollo y control de robots, la visión por computadora, la detección de objetos y la seguridad en aplicaciones industriales. Se destaca el uso de ROS como un marco común para el desarrollo y control de los sistemas de manipulación de robots.

Estos artículos están enfocados a la gran importancia de los cobots dentro de la industria. Todos estos proyectos presentan la utilidad del pick and place o de ROS en diferentes entornos, mostrando la gran diversidad de tareas que estos robots pueden hacer.

Algunos artículos como [37] donde se presenta el "BratWurst" enfocan el pick and place a la hostelería para la preparación de comida junto a inteligencia artificial para el control del calentado de la misma. Otros artículos [38] presentan soluciones para el sector de tratamiento de residuos desarrollando un sistema pick and place inteligente para el clasificado de basuras, o el uso del pick and place para la recolección de tomates maduros en el sector agrícola [39]. También hay artículos sobre aplicaciones pick-and-sort [40], que son aplicaciones similares al pick-and-place pero en el momento de dejar los objetos quedan ordenados (similar a un software de paletizado).

El resto de artículos [41] [42] [43] [44] [45] [46] [47] [48] tratan temas mucho más genéricos, donde no le dan una utilidad real al robot, sino que tratan de implementar ROS e IA para el control de los robots o para simulaciones. Algunos de ellos trabajan obteniendo las coordenadas del entorno donde se encuentra el robot usando algoritmos de inteligencia artificial, mientras que otros trabajan con redes neuronales que entrenan para detectar ciertos objetos y obtener sus posiciones.

Aunque todos estos artículos tratan sobre los mismos puntos basándose en la integración de robots en ROS y la visión artificial, cada uno de ellos lo aborda dentro de un área específica y presentan dificultades referentes a ese área. Los autores de estos artículos presentan los conceptos generales y se centran en su mayoría a la visión artificial. La integración de un robot en ROS es una tarea complicada y presenta multitud de dificultades [49]. Por todo esto, partir de una base donde un autor presente la integración de su robot en ROS incluyendo el modelo del robot y el driver para su control, todo unido al desarrollo de una aplicación de visión artificial, es una tarea compleja y costosa. Es por todo esto que, aunque estos artículos presentan información útil para el desarrollo del robot, se necesita seguir investigando centrando el estudio en la implementación de un robot dentro de ROS y Polyscope, para, posteriormente y una vez el robot funcione dentro de esos entornos con programas de manipulación de objetos básicos , poder continuar con la integración del robot utilizando técnicas de visión artificial.

Todos los artículos comparten la integración de un robot articulado usando inteligencia artificial para detectar los diferentes objetos. Realmente, el potencial de los robots colaborativos es su fácil configuración y trabajo junto a los humanos. Al añadir el factor IA y quitar el factor humano, realmente dejan de ser cobots, y pasan

a ser robots convencionales con IA.

4. Caso de estudio

En este apartado de caso de estudio, se analizará la implementación de un brazo robótico y una pinza en un entorno de control utilizando ROS (Robot Operating System) y el software Polyscope. El brazo robótico seleccionado es el modelo UR3e de Universal Robots, junto con una pinza de agarre RG2 para manipulación de objetos. Para la programación y control del sistema, se emplearán varias versiones de ROS comparando ROS1 y ROS2 y los problemas y resultados que presentan con la implementación del driver del robot para ROS, aprovechando los paquetes y librerías disponibles para planificación de movimientos y comunicación entre componentes como lo es el paquete Moveit. Por otro lado, el software Polyscope, desarrollado por Universal Robots, será utilizado como interfaz de control del brazo robótico, proporcionando la funcionalidad necesaria para desarrollar un programa pick and place o incluso servir como puente entre el robot y ROS para su control.

4.1. Hardware empleado

4.1.1. Brazo robótico

Para el desarrollo de esta investigación, se ha utilizado el brazo robótico UR3e de Universal Robots. [50]

Se trata de un robot manipulador colaborativo diseñado para realizar tareas precisas y delicadas en entornos de trabajo colaborativos con humanos. Tiene un diseño compacto y ligero y es altamente versátil, y se adapta fácilmente a espacios de trabajo reducidos. Dispone de una carga útil de 3 kg y puede manejar una gran variedad de objetos, lo que lo hace ideal para aplicaciones de ensamblaje, manipulación de materiales y tareas de precisión en industrias como la electrónica o la automotriz.

El UR3e cuenta con seis grados de libertad, lo que le permite realizar movimientos flexibles y precisos en un amplio rango de trabajo. Su diseño modular y su interfaz intuitiva facilitan su programación y reconfiguración, permitiendo una rápida adaptación a diferentes tareas y procesos. Además, el UR3e está equipado con una serie de características de seguridad que lo convierten en un robot colaborativo confiable incluyendo sensores de fuerza que detectan la presencia de obstáculos y detienen el movimiento del robot para evitar colisiones o accidentes con humanos. También ofrece una precisión en la repetición de tareas muy alta, lo que garantiza movimientos precisos y consistentes en todas las tareas. Además, su tecnología de control avanzada y su capacidad de realizar movimientos suaves y sin vibraciones contribuyen a una mayor eficiencia en las tareas realizadas.

Como cualquier robot colaborativo de Universal Robots, incorpora el software Polyscope, lo que permite programar tareas de forma sencilla e intuitiva, e integrar el robot en otro tipo de sistemas como puede ser ROS. Se explicará Polyscope en apartados posteriores. En la figura 10 se puede observar el cobot UR3e empleado.

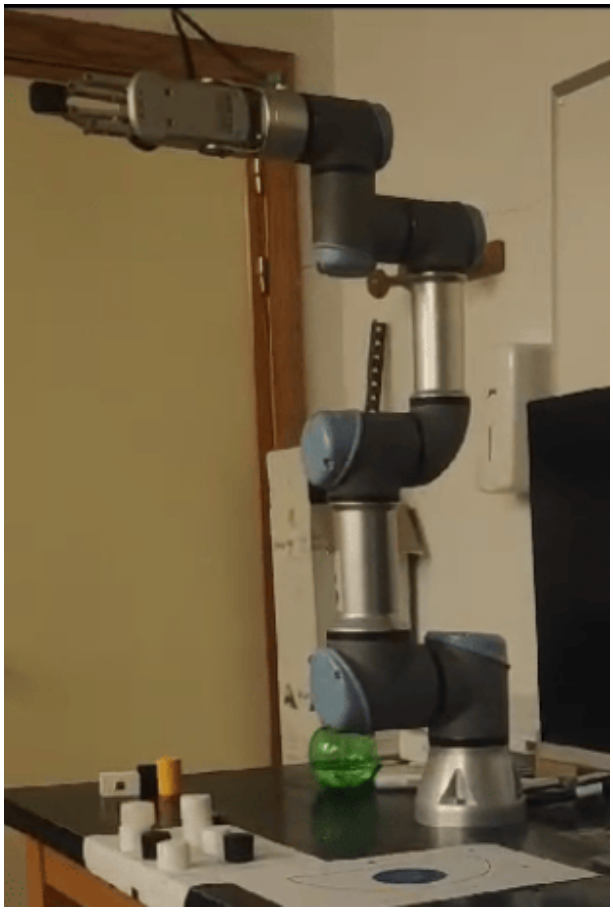

Figura 10: Brazo colaborativo UR3e utilizado

4.1.2. Pinza robótica

Para el desarrollo de esta investigación, se ha utilizado la pinza OnRobot RG2 de Universal Robots. [51]

La pinza RG2 es un accesorio diseñado para trabajar en conjunto con los brazos robóticos de Universal Robots, como el UR3e. Es una pinza eléctrica flexible utilizada en aplicaciones industriales y de ensamblaje para agarrar y manipular una amplia variedad de objetos.

Una de las características destacadas de la pinza RG2 es su rango de apertura de 0 a 110mm, lo que le permite adaptarse a diferentes tamaños y formas de objetos, haciéndola versátil y adecuada para manipular desde objetos pequeños y delicados hasta objetos más grandes y robustos.

La pinza RG2 ofrece control de agarre flexible con detección de fuerza. Puede ajustar la fuerza de agarre según las necesidades de la aplicación y también detectar si el objeto ha sido agarrado correctamente. Esto brinda precisión y adaptabilidad en la manipulación de objetos, proporcionando gran cantidad de información como feedback al usuario.

La pinza RG2 se puede programar y controlar directamente desde el software UR, lo que facilita su integración con los brazos robóticos UR. La programación se realiza a través de una interfaz gráfica intuitiva, lo que simplifica la configuración y personalización de la pinza. Esto se hace a través de un subprograma que se añade al software Polyscope del brazo robótico, permitiendo controlar la pinza desde el mismo software.

Además, la pinza RG2 es compatible con la integración de sensores externos. Esto significa que se puede utilizar en conjunto con sensores para detectar y ajustar automáticamente la posición y la fuerza de agarre en función de la retroalimentación sensorial. Esto proporciona una mayor precisión y adaptabilidad en la manipulación de objetos. En la figura 11 se puede observar la pinza robótica RG2 utilizada.

Figura 11: Pinza utilizada para el brazo robótico

4.2. Software implementado

4.2.1. Introducción básica a ROS

El primer software que se decidió utilizar fue ROS. ROS es un meta sistema operativo de código abierto que permite a los programadores de robots implementar funcionalidades de control de robots abstrayendo la parte hardware del usuario final.

ROS trabaja mediante paso de mensajes entre procesos, denominados nodos, que pueden ubicarse en la misma o diferentes máquinas. Se trata de un red peer-to-peer que permite una comunicación síncrona entre servicios o asíncrona a través de *topics*.

Este framework se diferencia de otras plataformas robóticas por su enfoque en la reutilización de código. Se basa en una estructura distribuida de nodos que se pueden diseñar de manera individualizada pero acoplarse fácilmente entre sí. Los procesos se agrupan en paquetes que pueden intercambiarse, compartirse y distribuirse fácilmente, ofreciendo funcionalidades reutilizables para otros desarrolladores o investigadores. Además, ROS ofrece un sistema unificado de repositorios de código (*federated systemof code repositories*) que facilita la colaboración. Además, ROS es ligero, compatible con múltiples lenguajes de programación y cuenta con un marco de prueba integrado. Es escalable y adecuado para sistemas y procesos de desarrollo de gran envergadura.

4.2.2. Conceptos de ROS1

ROS surgió como concepto para controlar robots en 2007, aunque su primera versión estable "ROS Box Turtle" no fue lanzada hasta 2010.

En ROS1 [52] se desarrolla un sistema centralizado en un nodo denominado nodo maestro o "rosmaster". Este nodo maestro sirve como coordinador, registrando todos los nodos, *topics* y servicios disponibles. Para entender este funcionamiento, hay que tener claros los siguientes conceptos:

1. Nodos: Son procesos que realizan alguna tarea. Un sistema de control de robots completo puede constar de uno o varios nodos distribuidos en uno o varias máquinas. Los nodos se organizan siguiendo una estructura de grafo de nodos, donde cada uno de ellos se comunica o no con otros nodos.

2. Rosmaster: Es el nodo central encargado de coordinar al resto de nodos, por lo que este nodo debe estar siempre arrancado y todos los nodos del sistema deben saber de su existencia (en qué máquina se encuentra).

3. Mensajes: Es una estructura de datos que permite a los nodos comunicarse entre ellos. Pueden ser números enteros o decimales, cadenas de texto u otros tipos de datos.

4. *Topics*: Son canales de comunicación entre nodos que actúan de forma similar a un buzón siguiendo el patrón publicador-suscriptor. Cuando un nodo (el publicador) quiere enviar información a otros nodos (los suscriptores), publicará mensajes del tipo que corresponda a un tópico del que leerán os nodos suscriptores. Permiten una comunicación muchos a muchos de forma asíncrona y simultánea entre varios nodos. Los *topics* tienen una sintaxis del tipo "/topico/subtopico/...", donde el tópico "/" es el global del espacio de trabajo. De esta forma se pueden mostrar todos los *topics* del sistema en forma de árbol y permitiendo organizar cada subsistema según los *topics*. Por ejemplo, si se quiere disponer de un tópico donde se almacenen todos los datos de dos sensores, podemos definir 2 *topics* como : "/sensores/sensor1" y "/sensores/sensor2". Con esto se define un árbol donde "/sensores" es la raíz (y contendrá todos los

mensajes), y "sensor1" y "sensor2" son dos ramas del árbol independientes para cada sensor. Los *topics* están orientados a la difusión de información entre nodos, por lo que a cada tópico se le asignará un tipo de mensaje con el que se trabajará. Por ejemplo, en un *topic* donde se publican mensajes sobre la distancia medida por un sensor ultrasonido se le asignará un tipo de mensaje de tipo entero (*int*) que corresponde a la distancia en centímetros al objetivo.

5. Servicios: Al igual que los *topics*, los servicios también son canales de comunicación que permiten una comunicación uno a uno de forma síncrona, siguiendo el patrón request-reply. Con los servicios, los nodos pueden pedir a otros nodos realizar una acción y recibir una respuesta a esa acción, es decir, un nodo solicitará y dará servicios a otros nodos para realizar acciones especificas, diferenciándose así de los *topics* cuyo objetivo es el de difundir información.

Toda esta información sobre el funcionamiento de los nodos y sus comunicaciones se puede sintetizar en la figura 12 obtenida de la página de "Generation Robots" [53].

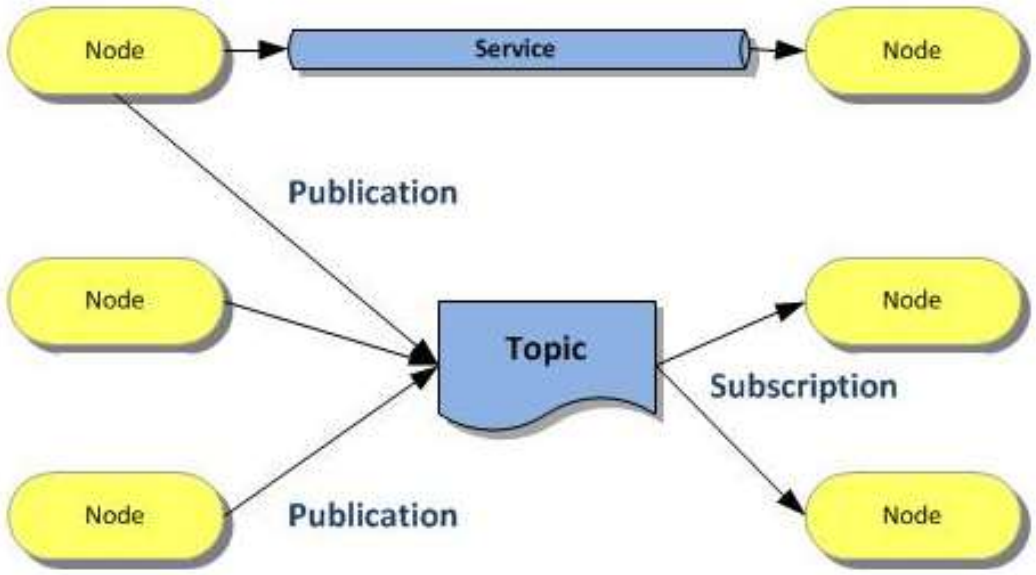

Figura 12: Comunicación en ROS 1

Aunque esta captura refleja el funcionamiento real de la comunicación entre nodos en ROS, hay que recordar que este sistema solo estará activo en el caso de que el nodo maestro o "rosmaster" esté iniciado. Este nodo también trabaja al igual que el resto de nodos con servicios y *topics*, pero es el encargado de registrar toda la información referente al resto de nodos, por lo que es indispensable en el sistema centralizado de ROS 1.

A partir de aquí surge el concepto de paquete de ROS. Una vez definida la forma de comunicación entre diferentes nodos se puede comenzar a crear una pieza software

capaz de resolver un problema utilizando este sistema de paso de mensajes entre nodos. Estas unidades de software se organizan mediante los paquetes de ROS. Estos paquetes contienen los nodos, librerías, información, tipos de datos y configuraciones de forma que se compone una unidad atómica reutilizable de software. Eso permite a un desarrollador crear y encapsular la funcionalidad que ha implementado para un determinado robot y compartirla de forma rápida y sencilla con otros desarrolladores, con lo que otros podrán utilizarla sin gran esfuerzo para sus propios proyectos. Esto convierte a ROS en un framework de trabajo perfecto para la reutilización de software gracias a la base de datos de paquetes que almacenan [54].

Gracias a la creación de paquetes surgen conceptos más avanzados de ROS. Gracias a la creación de paquetes surgen los paquetes "primitivos" que contienen funcionalidad que se podría reutilizar en todos o casi todos los proyectos.

Los paquetes que afectan de forma directa a este caso de estudio son:

1. Paquete "tf" (Transform Frame) [55]: proporciona funciones para administrar las transformaciones en un sistema robótico permitiendo manejar y transformar las coordenadas entre varios "frames" de un entorno ROS empleando para ellos una estructura de grafo. Con este paquete se pueden realizar transformaciones en cualquier elemento de un sistema robótico sin la necesidad de realizar los cálculos de forma manual. Los "frames" a los que afectan estas transformaciones son sistemas de coordenadas 3D utilizados para definir la posición y orientación de los elementos dentro de un sistema robótico: por ejemplo, las diferentes partes móviles de un robot, el propio escenario o los sensores que incorpora. En la figura 13 se puede observar los "frames" que habría que definir para un brazo robótico y un objeto que debe manejar [56]:

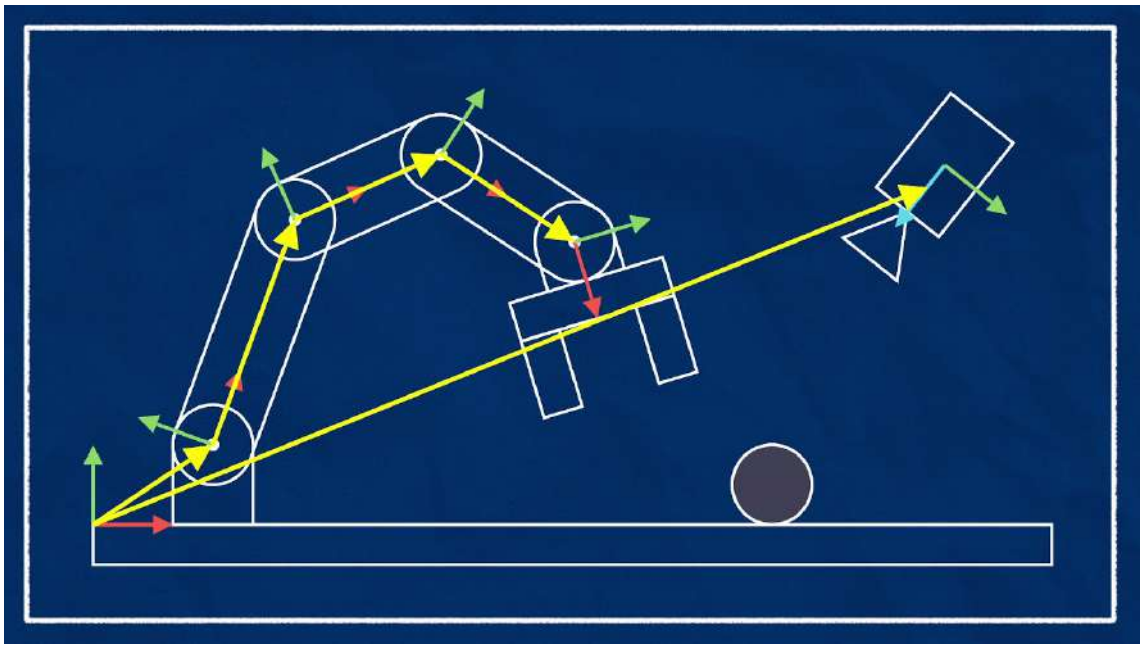

Figura 13: Ejemplo de *frames* y transformaciones en un brazo robótico

Los *frames* definidos en la figura 13 se corresponden con las diferentes articulaciones del brazo, el objeto que debe manejar y el propio espacio de trabajo. Los *frames* están encadenados formando un grafo, donde el primer *frame* definido es el colocado en la esquina inferior izquierda y que a partir de ese "frame" definido se conectan tanto la base del robot como el objeto colocado en la parte superior derecha. A partir de la base del brazo robótico, se irán conectando el resto de articulaciones. Además, a cada una de las articulaciones se les asocia una posición y una orientación (se puede ver por los ejes marcados como flechas verdes y rojas). Estas características se definen y se especifican en el paquete "tf".

2. Paquete "actionlib" [57]: Permite definir las acciones que se pueden realizar en un sistema robótico, ofreciendo herramientas que facilitan la comunicación y coordinación entre los servidores que ejecutan las acciones y los clientes que las solicitan. Además, este paquete permite al usuario recibir retroalimentación del robot y de las acciones que ejecuta en tiempo real, facilitando la monitorización del sistema. Para realizar esto, el paquete ofrece 3 tipos de mensajes:

   a) *Goal*: Permite establecer el objetivo final que debe cumplir la acción definida. En el caso de un brazo robótico, el goal podría ser la configuración del brazo robótico que se desea alcanzar.

   b) *Feedback*: Retroalimentación que devuelve el servidor que realiza la acción según va progresando la acción. Con esto el usuario puede conocer el estado de la acción en todo momento. En el control de un brazo robótico, podría devolver información de la posición del brazo durante toda la ejecución de la trayectoria o incluso choques y errores que se puedan producir durante la misma.

   c) *Result*: El servidor le indica, una vez acabada la acción, el resultado de la misma al cliente. Por ejemplo, en la ejecución de la trayectoria de un brazo robótico, el resultado devuelvo podría ser si la trayectoria se ha completado o no.

   El modo en el que funciona este paquete se puede ver en la figura 14, donde se dispone de un cliente que solicita una acción estableciendo un "goal" y el paquete se encarga de enviar dicha petición a un servidor que se encarga de realizar dicha acción [57]:

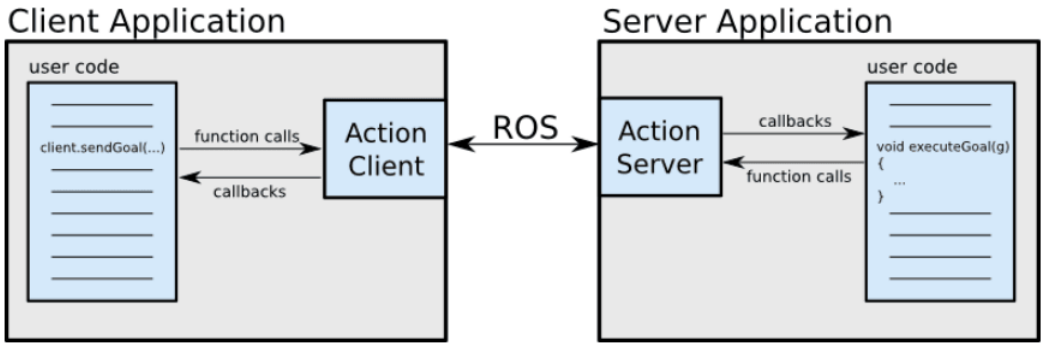

Figura 14: Ejemplo de funcionamiento de actionlib

3. Paquete "common_msgs": Este paquete contiene mensajes que se usan en la gran mayoría de paquetes desarrollados. Algunos de estos mensajes son:

   a) actionlib_msgs: Mensajes con los que trabaja el paquete "actionlib"

   b) geometry_msgs: Utilizados para especificar la geometría y transformaciones de coordenadas o cambios de sistemas de referencia en el espacio.

   c) sensor_msgs: Comunes en la utilización de sensores. Sirven para mostrar la información de diferentes sensores de un robot como cámaras, sensores infrarrojos o incluso el estado de la batería de un robot.

   Este paquete se empleará en otros paquetes necesarios para el control del brazo robótico desde ROS.

4. Paquete "urdf" [58]: Contiene toda la funcionalidad necesaria para interpretar con `C++` los ficheros URDF (Unified Robot Description Format). Estos ficheros siguen el formato XML y sirven para crear una descripción del robot que se empleará en otras herramientas para conocer su estructura física, como por ejemplo, en herramientas de visualización o simulación. Este fichero URDF sirve para definir las articulaciones del robot, los enlaces entre dichas articulaciones, los sensores que contiene u otros elementos presentes en el robot, tratando de simular el robot real.

   En la figura 15 se muestra un ejemplo básico de un fichero URDF donde se especifica la estructura de un brazo robótico con una única articulación denominado "simple_arm":

```
<?xml version="1.0"?>
<robot name="simple_arm">
  <link name="base_link"/>
  <link name="arm_link"/>
  <joint name="arm_joint" type="revolute">
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="arm_link"/>
    <axis xyz="0 0 1"/>
    <limit effort="100" velocity="1.57" lower="-3.14" upper="3.14"/>
  </joint>
</robot>
```

Figura 15: Ejemplo de un fichero URDF

En este fichero URDF se definen una serie de etiquetas XML:

a) Etiqueta *robot*: Es la primera etiqueta y la raíz del fichero URDF. Envuelve toda la descripción del robot y sirve para poner un identificador al robot.

b) Etiqueta *link*: Define un enlace que representa un componente físico del modelo. En este caso hay 2 enlaces que representan la base del robot ("base_link") y el brazo del robot ("arm_link").

c) Etiqueta *joint*: Esta etiqueta representa una articulación que conecta 2 enlaces. En esta caso la articulación es de tipo "revolute" o de giro, permitiendo que dicha articulación pueda rotar. Dentro de esta articulación hay que especificar los enlaces que une y la posición y orientación de la misma.

d) Etiqueta *origin*: Esta etiqueta especifica la posición y orientación iniciales de la articulación dentro del espacio.

e) Etiquetas *parent* y *child*: La etiqueta *parent* indica el enlace principal sobre el que se apoya el siguiente enlace (*child*). Es decir, el robot dispone de una base de la que se extienden articulaciones simulando un cuerpo humano, por lo que el enlace padre sería la componente principal sobre la que se extenderá el robot hacia otro componente. Por ejemplo, en un cuerpo humano, podría indicarse el tronco como *parent*, el brazo como *child* y el hombro como *joint*.

f) Etiqueta *axis*': Sirve para especificar el eje de coordenadas para el enlace hijo desde el enlace padre. Para una articulación de rotación es el eje sobre el que gira. Para una articulación, prismática es el eje a lo largo del que se desplaza. En el fichero ejemplo se especifica que la articulación gira alrededor del eje z.

g) Etiqueta *limit*: Especifica los límites de una articulación. Al igual que en el cuerpo humano una articulación tiene restricciones físicas en su movimiento, en un robot hay que especificar los límites de movimiento de sus componentes. Si no, podría ocurrir que dos componentes se solapasen o hubiera movimientos que en la realidad no puedan ejecutarse. Dispone de varios parámetros en los que se puede configurar la fuerza máxima que puede realizar la articulación ("effort"), la velocidad máxima a la

que puede moverse ("velocity") o los límites del recorrido angular, para una articulación de rotación, o los límites de recorrido lineal, para una articulación prismática.

Únicamente con estas etiquetas se pueden describir desde estas estructuras robóticas muy simples hasta estructuras muy complejas.

Además, este paquete incluye una herramienta para trabajar con ficheros con formato XACRO (XML Macros). Este formato permite crear ficheros URDF modulares y reutilizables a partir de plantillas, por lo que la generación de modelos de robots se facilita permitiendo crear y reutilizar componentes más comoplejos.

5. Paquete "rviz" [59]: RVIZ (ROS Visualization) es una herramienta para visualizar en tiempo real un sistema robótico. Desde RVIZ se puede visualizar un sistema completo junto a un entorno, permitiendo trabajar de forma sencilla y visual con un robot. No solo permite mostrar la estructura del robot, sino que permite visualizar los datos de todo tipo de sensores, el entorno de trabajo o incluso la información de los *topics*. Con esta herramienta se puede depurar el trabajo de un robot dentro de un entorno, comprobar el funcionamiento de sus sensores o incluso configurar de forma rápida todos los componentes.

   RVIZ se utiliza en multitud de paquetes y aplicaciones ya que permite trabajar con modelos URDF y es altamente configurable.

También surge el concepto de librería para clientes. Una librería de ROS podría definirse como un conjunto de paquetes desarrollados para resolver un problema. Normalmente es común desarrollar librerías que resuelvan un problema complejo a partir de paquetes independientes entre sí pero comunicados y que posteriormente se distribuyen como una librería completa. Por ejemplo, para desplegar una serie de nodos, comunicarlos y que pueda desarrollarse una sistema completo entre ellos se disponen de librerías como "roscpp" o "rospy". Estas librerías contienen todo lo necesario para poder desarrollar un programa software simple en lenguajes como `C++` o Python respectivamente, aunque existen muchas otras librerías semejantes que permiten otros lenguajes de programación u otras funcionalidades.

Otro concepto interesante es el espacio de trabajo o *workspace*. Un espacio de trabajo en ROS es un entorno estructurado para organizar, desarrollar, compilar y ejecutar paquetes de ROS con el fin de desarrollar una aplicación completa.

Para trabajar con estos espacios de trabajo existe la herramienta "catkin" [60]. Esta herramienta permite construir y gestionar los paquetes dentro de un espacio de trabajo de forma automática. Se basa en CMake como sistema de compilación, siguiendo una estructura ordenada de directorios donde se almacenarán los paquetes, los ejecutables compilados y los ficheros de configuración.

Todas estas funcionalidades se han empleado en este caso de estudio. Para este proyecto, se ha generado un workspace de ROS con una serie de paquetes donde se encuentra toda la funcionalidad necesaria para controlar el brazo robótico UR3e desde ROS. Estos paquetes contienen el driver necesario para comunicar ROS el

brazo robótico y los controladores necesarios para enviar los comandos que permitan al brazo moverse, así como otras funcionalidades como la visualización del robot usando RVIZ o incluso la posibilidad de usar planificadores de trayectorias.

En este caso, se ha utilizado el paquete Moveit. Este paquete ofrece herramientas para visualizar y planificar trayectorias de un robot de forma rápida e intuitiva. Moveit permite visualizar en tiempo real un robot, a partir de un fichero URDF, y mostrar las trayectorias que sigue este robot.

Moveit [61] cuenta con un asistente que permite generar, mediante una interfaz gráfica y de forma intuitiva el modelo de un robot que posteriormente se almacenará en un fichero XACRO o URDF. Con un modelo válido de un robot, Moveit se encarga de realizar todas las transformaciones necesarias acorde a la planificación de la trayectoria realizada, marcando un estado inicial y un objetivo ("goal") que debe alcanzar.

Para entender mejor como funciona, se puede observar la figura 16, obtenida de la propia documentación de Moveit [62]:

Como se puede observar, el usuario interactuará a través de la terminal, de la interfaz gráfica de RVIZ o incluso enviará comandos directamente al nodo "move_group" de Moveit. Una vez el usuario establezca el movimiento o "goal" que desea alcanzar, Moveit se encarga de todo. Este nodo "move_group" es el encargado de recibir las peticiones del usuario, de cargar toda la escena y el modelo del robot a partir de los URDFs y de cargar toda la configuración necesaria del robot. Una vez ha inicializado todo, el nodo trabaja con una serie de *topics*, como el tópico "joint_states", a los que enviará información sobre los movimientos realizados y el estado del robot o de los que recibirá información como el objetivo a alcanzar o cambios en la configuración de Moveit.

Con todo esto, el nodo se encarga de enviar las peticiones necesarias al nodo de planificación, que será el encargado de planificar la trayectoria del movimiento del robot. Esta planificación se realiza a través de librerías de cálculo de trayectorias que tienen en cuenta las colisiones existentes en la escena y generan la ruta más óptima hasta el objetivo. En particular, se ha utilizado para este proyecto la librería de OMPL (Open Motion Planning Library) que es open source y ofrece una serie de algoritmos para la planificación de los movimientos. Para el cálculo de las colisiones y evitarlas, además de OMPL se utiliza FCL (Flexible Collision Library) que trabaja con modelos compuestos por triángulos para calcular las colisiones en tiempo real.

A su vez, mientras planifica el movimiento y detecta las posibles colisiones, también se dispone de un nodo que controla la trayectoria del robot real (en caso de haberlo y que no sea una simulación). Contiene los controladores necesarios para hacer mover al robot real, por lo que podría decirse que es la parte encargada del movimiento del hardware (el driver).

Por último, falta un concepto útil y necesario para poder ejecutar todos estos programas de forma coordinada, el comando "roslaunch". Este comando junto a los ficheros "launch" permiten al usuario lanzar y configurar nodos de una forma rápida y coordinada, evitando la necesidad de ejecutar cada uno de los paquetes de forma

independiente. En estos ficheros, siguiendo una sintaxis XML al igual que los ficheros URDF, se especifican los nodos o paquetes a ejecutar junto a los parámetros que requieran. Posteriormente, estos ficheros se ejecutan con el comando "roslaunch", lanzando todos los nodos especificados de forma automática en diferentes procesos. Un ejemplo de un fichero launch es el siguiente:

```
<launch>
    <!-- Lanzar el primer nodo -->
    <node name="node1" pkg="my_package" type="node1" output="screen" />

    <!-- Lanzar el segundo nodo -->
    <node name="node2" pkg="my_package" type="node2" output="screen" />
</launch>
```

Este ejemplo básico de un fichero "launch" aparecen 2 nodos, "node1" y "node2", que se ejecutarán al lanzar el fichero "launch". Como se puede observar, existen la etiqueta ´´<launch>" que envuelve a todo el fichero, y la etiqueta "<node>" que sirve para especificar un nodo y donde se pueden cambiar parámetros como el paquete al que pertenece o parámetros de ejecución al igual que si ejecutáramos el nodo por separado. También existen otras etiquetas como "<include>" que sirve para incluir otros ficheros "launch" dentro del actual o "<args>" que se añade dentro de las etiquetas anteriores y sirven para especificar los argumentos a utilizar.

Todos estos concepto, en mayor o menor medida, se han empleado durante la investigación del caso de estudio relacionado con el brazo robótico UR3e.

4.2.3. Conceptos de ROS2

Al igual que cualquier pieza software, ROS1 tenía sus limitaciones y problemas como la seguridad o la fiabilidad en algunos entornos que limitaban o entorpecían el desarrollo de algunas funcionalidades o dificultaban la evolución de sistemas complejos. Es por todo esto que surge en 2014 ROS2 [63]. La nueva versión de ROS continúa con todos los puntos fuertes de ROS1 pero trata de resolver todas las limitaciones que presenta.

La principal diferencia entre ROS2 y ROS1 es su arquitectura, ROS1 seguía una arquitectura centralizada en el nodo maestro el cual era necesario ejecutarlo en alguna máquina para que el sistema funcionara y se seguían comunicaciones punto a punto. En cambio, en ROS2 se sigue una arquitectura distribuida, donde cada nodo funciona de forma independiente en cualquier entorno y se comunica con el resto de nodos a través de los llamados IDs de dominio que permiten identificar qué nodos forman parte de un ecosistema para poder comunicarse. Cuando se arranca un nodo en ROS2, se le establece un "ROS_DOMAIN_ID" por el que se anunciará y por el que el resto el resto de nodos del mismo dominio responderán. Con esto todos los nodos sabrán de la existencia de otros dentro de un dominio, aunque periódicamente se anunciarán de nuevo para seguir avisando de su correcto funcionamiento. Una vez un nodo cae, ya sea por error o por finalización correcta, avisa al resto de nodos del mismo dominio. Esta arquitectura se denomina DDS (Data Distribution Service) y permite comunicaciones más eficientes y confiables.

Además, ROS2 ofrece una compatibilidad multiplataforma mientras que ROS1 solo tenía soporte para Linux. También incluye grandes mejoras en seguridad para autenticación, cifrado o control de acceso, permitiendo comunicaciones entre nodos mucho más seguras. Otra de las mejoras que incluye es el soporte para sistemas de tiempo real con el middleware RTPS (Real-Time Publish-Subscribe) de DDS. Finalmente, otra de las mejoras que presenta ROS2 es un mayor soporte para otros lenguajes de programación o incluso versiones más actuales de los ya utilizados en ROS1.

ROS2 es una evolución significativa de ROS1, pero sigue las misma filosofía. ROS2 trabaja al igual que ROS1 con nodos, mensajes, servicios, acciones y con el mismo tipo de ficheros como los ficheros "msg" para los mensajes o los ficheros "URDF" para los modelos de los sistemas robóticos.

En cambio, los ficheros "launch", aunque tienen la misma funcionalidad la sintaxis cambia. Ahora los ficheros "launch" se especifican mediante un script de Python3. Por ejemplo, este sería un fichero "launch" en ROS2 semejante al ejemplo anterior en ROS1:

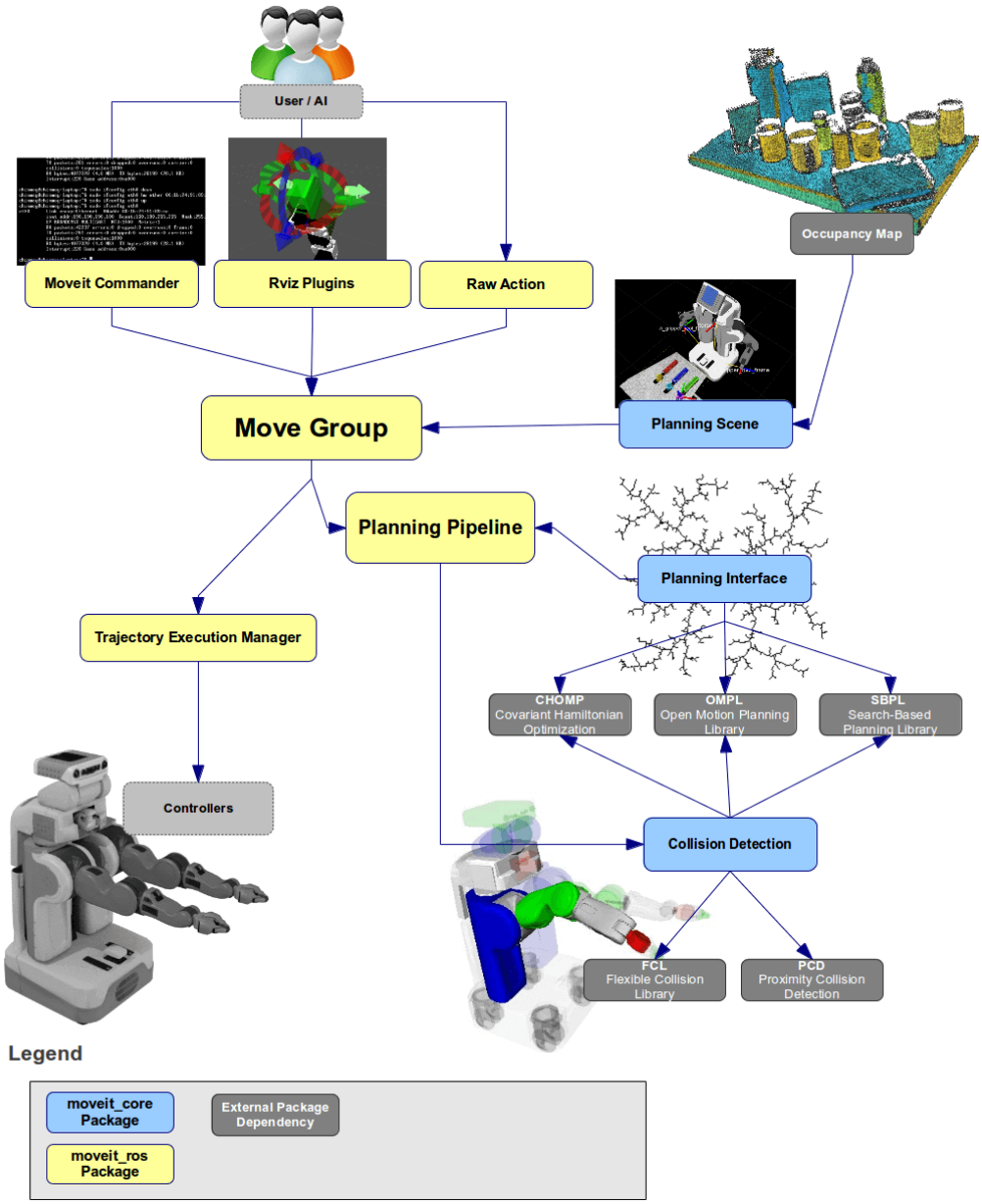

Figura 16: Funcionamiento de Moveit

```
# Ejemplo de archivo launch en ROS 2

# Importar el paquete de lanzamiento de ROS 2
import launch
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    # Crear una descripción de lanzamiento
    ld = LaunchDescription()

    # Lanzar el primer nodo
    node1 = Node(
        package='my_package',
        executable='node1',
        name='node1',
        output='screen'
    )

    # Lanzar el segundo nodo
    node2 = Node(
        package='my_package',
        executable='node2',
        name='node2',
        output='screen'
    )

    # Agregar los nodos a la descripción de lanzamiento
    ld.add_action(node1)
    ld.add_action(node2)

    return ld
```

Como se puede observar, las diferencias entre los ficheros "launch" entre ROS1 y ROS2 son grandes pero a la vez son muy semejantes. En ROS2, la función "generate_launch_description" actúa de forma similar a la etiqueta "<launch>" en ROS1, envolviendo todos los nodos especificados. Dentro de esta función, se establecen todos los nodos. Para ello se genera una descripción "LaunchDescription" y se registran los 2 nodos al igual que en el ejemplo anterior, "node1" y "node2", para finalmente devolver dicha descripción.

Las principales diferencias entre los ficheros ´´launch" de ROS1 y ROS2 son:

1. Sintaxis: Mientras que en ROS1 se utiliza XML, en ROS2 se utiliza YAML que es más legible y fácil de entender.

2. Argumentos y comandos: en ROS2 se pueden utilizar los comandos y argumentos que se proporcionan al ejecutar el fichero launch, permitiendo una integración más sencilla con otras herramientas.

3. Grupos de nodos: en ROS2 se permiten configurar grupos de nodos para poder trabajar con ellos de forma conjunta, ofreciendo una mayor modularidad y flexibilidad.

Otra de las diferencias en ROS2 es en los "workspace". La carpeta "devel" ahora pasa a ser la carpeta "install", y la herramienta de compilación "catkin" ahora pasa a

ser la herramienta “colcon”. Aunque estas herramientas ofrecen la misma funcionalidad, la herramienta colcon ofrece una mayor flexibilidad y características mejoradas, como la construcción modular y la gestión de dependencias.

El resto de paquetes como Moveit o RVIZ, aunque están adaptados a la sintaxis y componentes de ROS2, ofrecen las mismas funcionalidades en ROS2.

4.2.4. Polyscope

Polyscope es una interfaz de programación y visualización para robots colaborativos (cobots) fabricados por Universal Robots. Es una herramienta que permite programar, controlar y supervisar los robots UR de forma intuitiva y sencilla. Proporciona una interfaz gráfica fácil de usar que permite a los usuarios interactuar con el robot a través de una pantalla táctil. Esta pantalla permite controlar todo el robot e incluso dispone de mecanismos de seguridad para liberar el control del movimiento del robot y poder moverlo a mano o incluso detenerlo por completo mediante una seta de emergencia. La pantalla táctil puede verse en la figura 17.

Figura 17: Tablet de control con Polyscope

Algunas características y funcionalidades de Polyscope son las siguientes:

1. Programación por flujo: Polyscope utiliza un enfoque de programación por flujo, donde los usuarios pueden crear programas mediante una serie de bloques de función visualmente representados. Esto facilita la creación y edición de programas sin necesidad de escribir código manualmente.

2. Control del robot: La interfaz de Polyscope permite controlar el robot en tiempo real, permitiendo iniciar y detener movimientos, cambiar parámetros y configuraciones, y supervisar el estado del robot durante la ejecución en tiempo real.

3. Configuración y calibración: Polyscope permite configurar y calibrar el robot de manera sencilla. Los usuarios pueden definir zonas de trabajo, ajustar los límites de velocidad y fuerza, y realizar otras configuraciones específicas del robot.

4. Programación avanzada: Además de la programación por flujo, Polyscope también permite la programación avanzada utilizando un lenguaje de script llamado URScript. Los usuarios pueden escribir scripts personalizados para realizar tareas complejas y personalizadas.

5. Monitorizar y diagnóstico: Polyscope proporciona herramientas de monitorizar y diagnóstico que permiten a los usuarios verificar el estado del robot, supervisar las señales y variables del programa, y solucionar problemas en caso de error.

6. Extensible y modular: Polyscope permite, mediante el uso de los denominados URCaps (Universal Robots Capsules), extender las capacidades del robot. Los URCaps permiten a los desarrolladores incluir nuevas funcionalidades al robot sin apenas esfuerzo. Por ejemplo, un URCap utilizado en este caso de estudio es el "External Control" que permite controlar el brazo robótico desde otra máquina.

Polyscope también ofrece una serie de plantillas que permiten realizar tareas configurables. Estas tareas pueden ser desde movimientos sencillos en un eje o secuencias simples, hasta tareas complejas con secuencias de movimiento que permitan realizar trabajos de manipulación, como por ejemplo, el paletizado. Para estas plantillas se ofrece un tutorial paso a paso del funcionamiento de la plantilla y cómo configurar cada movimiento del robot para realizar tareas. Por ejemplo, una de las plantillas ofrecidas permite al robot generar una secuencia de movimiento a partir de un código GCODE, lo que permite al robot poder realizar tareas de fresado como si de una CNC se tratase (aunque habría que acoplarle la herramienta oportuna para ello al robot) o pintado.

5. Implementación y resultados obtenidos

En esta sección se mostrarán los pasos seguidos y los resultados alcanzados para las diferentes versiones de ROS donde se ha tratado de integrar el driver del brazo robótico UR3e y de los programas desarrollados para Polyscope.

5.1. ROS1

Ya que las versiones de ROS1 llevan varios años, son estables y la mayoría no tienen más desarrollo, los drivers y paquetes necesarios para el correcto funcionamiento del robot en estos entornos están completos y funcionan de forma correcta. Es por esto que solo se mostrará el desarrollo para la versión de ROS Melodic Morenia.

5.1.1. ROS Melodic Morenia

En primer lugar, y para poder trabajar con todos los paquetes necesarios, se ha creado un workspace. En este workspace se almacenarán y compilarán todos los paquetes encargados de mover el brazo robótico utilizando ROS. Estos paquetes son:

1. Paquete **"Universal Robots ROS Driver"** de Universal Robots [64], que proporciona el driver adaptado para los robots de Universal Robots junto a un paquete de calibración
2. Paquete **"universal_robots"** de ros_industrial [65], que proporciona drivers, configuraciones y herramientas para robots industriales, teniendo un paquete específico con herramientas para los cobots de Universal Robots.

Una vez descargados y compilados los paquetes en el workspace, se ejecuta la calibración para obtener la información del robot como la posición o los parámetros que emplea. Esto se puede hacer con los siguientes comandos, teniendo en cuenta que en estas pruebas la IP del robot es la 212.128.172.172.

```
$ source /opt/ros/melodic/setup.bash
$ source ~/workspace/devel/setup.bash
$ roslaunch ur_calibration calibration_correction.launch robot_ip:=212.128.172.172 \
target_filename:="${HOME}/robot_calibration.yaml"
```

Los primeros comandos son necesarios ejecutarlos en cualquier terminal nueva ya que se encargan de cargar variables de configuración tanto de la propia instalación de ROS global como la configuración del workspace con el que trabaja. Ejecutando la calibración se obtiene un fichero en formato YAML. En este fichero se obtiene a posición y orientación de cada una de las articulaciones en ese momento.

Con esto, ya se puede ejecutar el driver del robot encargado de conectar el software de ROS al software de Polyscope para poder controlar el robot. Para ello hay que seguir 2 pasos:

1. Ejecutar el driver en ROS.
2. Descargar la URCap de "External Control" en Polyscope que será la encargada de conectarse al driver para el control del robot.

El comando para ejecutar el driver es el siguiente:

```
$ roslaunch ur_robot_driver ur3e_bringup.launch robot_ip:=212.128.172.172 \
kinematics_config:="${HOME}/robot_calibration.yaml"
```

Con esto el driver cargará la configuración creada en la calibración y esperará a que el robot se conecte desde Polyscope. Una vez el driver está arrancado, se ejecuta el URCap del robot y este se conectará al driver y este indicará que el robot está conectado correctamente o no hay conexión.

A partir de este punto, el driver está correctamente arrancado y aparecen una serie de nodos y *topics* que permiten controlar el robot estableciendo las posiciones de cada articulación o devuelven información del estado del robot. Estos nodos y *topics* se puede visualizar con la herramienta "rqt_graph" como se puede ver en la figura 18.

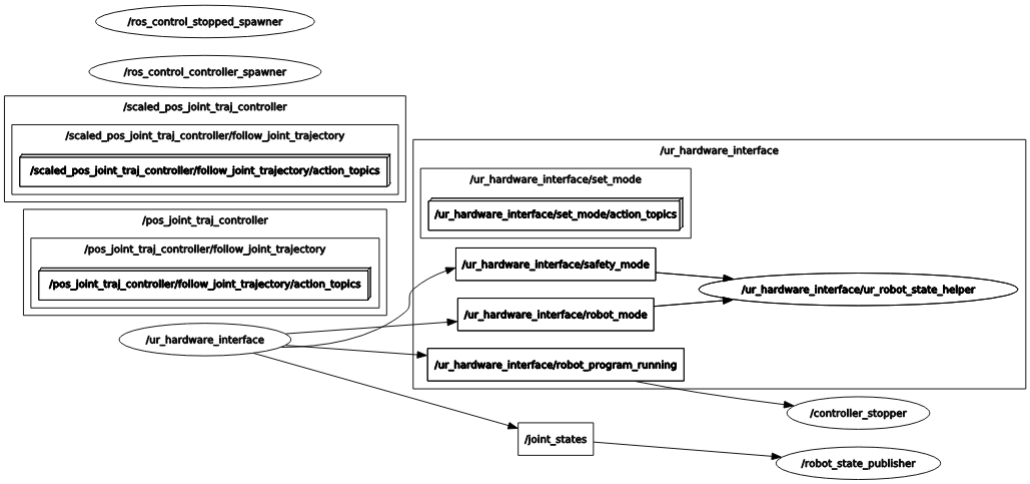

Figura 18: *topics* y nodos creados con el driver

Como se puede ver en el grafo se inician varios *topics* referentes al robot. Los *topics* referentes al control del robot y los encargados de transmitir la información necesaria entre el robot y ROS son los siguientes:

1. Los *subtopics* dentro de "pos_joint_traj_controller" sirven para generar y realizar trayectorias de posición para las articulaciones del robot. Como se tratan de acciones a realizar, hay *topics* dentro de este conjunto para establecer el goal o recibir el feedback y el resultado, así como ver el estado de la acción.
2. Los *subtopics* dentro de "scaled_pos_joint_traj_controller" sirven para escalar los valores enviados a los *topics* anteriores para adaptarlos a las necesidades de cada controlador.

3. Los *subtopics* dentro de "ur_hardware_interface" son los encargados de la comunicación entre ROS y Polyscope. Dentro de este conjunto de *topics* hay *topics* para ver el estado de la E/S del robot, el programa que se está ejecutando, el modo del robot (en caso de estar parado del todo, parado por emergencia, arrancado, etc), si está activado o no el mecanismo de seguridad o *topics* que realizan una acción para cambiar el modo del robot desde ROS.

4. Tópico "joint_states" sirve para comunicar información referente a la posición, velocidad y esfuerzo de cada una de las articulaciones del robot. Con este tópico los planificadores o controladores pueden obtener información en tiempo real del estado de las articulaciones del robot.

5. Otros *topics* adicionales son "robot_state_publisher" que se encarga de publicar la información referente al tópico de "joint_states"; el tópico "controller_stopper" que se encarga de detener el controlador en el caso de que el programa que se ejecuta en el robot se detenga; el tópico "tf" encargado de todas las transformaciones de cada componente del robot; y otros *topics* que se ejecutan al arrancar .

Todos los *topics* activos pueden consultarse con el comando "rostopic list" como se muestra en la figura 19:

```
/joint_group_vel_controller/command
/joint_states
/pos_joint_traj_controller/command
/pos_joint_traj_controller/follow_joint_trajectory/cancel
/pos_joint_traj_controller/follow_joint_trajectory/feedback
/pos_joint_traj_controller/follow_joint_trajectory/goal
/pos_joint_traj_controller/follow_joint_trajectory/result
/pos_joint_traj_controller/follow_joint_trajectory/status
/pos_joint_traj_controller/state
/rosout
/rosout_agg
/scaled_pos_joint_traj_controller/command
/scaled_pos_joint_traj_controller/follow_joint_trajectory/cancel
/scaled_pos_joint_traj_controller/follow_joint_trajectory/feedback
/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
/scaled_pos_joint_traj_controller/follow_joint_trajectory/result
/scaled_pos_joint_traj_controller/follow_joint_trajectory/status
/scaled_pos_joint_traj_controller/state
/speed_scaling_factor
/tf
/tf_static
/ur_hardware_interface/io_states
/ur_hardware_interface/robot_mode
/ur_hardware_interface/robot_program_running
/ur_hardware_interface/safety_mode
/ur_hardware_interface/script_command
/ur_hardware_interface/set_mode/cancel
/ur_hardware_interface/set_mode/feedback
/ur_hardware_interface/set_mode/goal
/ur_hardware_interface/set_mode/result
/ur_hardware_interface/set_mode/status
/ur_hardware_interface/tool_data
/wrench
```

Figura 19: *Topics* arrancados en ROS1 al ejecutar el driver

Con todo esto el usuario ya puede mover el robot enviando la información necesaria a dichos *topics*. Sin embargo, para realizar planificación o facilitar estas tareas, surgen los controladores o planificadores.

Se dispone de un controlador denominado "rqt_joint_trajectory_controller" que permite trabajar directamente con los *topics* de posición escalada del robot. El controlador se encarga de proporcionar una interfaz gráfica muy sencilla con unas barras donde el usuario podrá interactuar con cada articulación del robot por separado. La interfaz proporcionada es la mostrada en la figura 20.

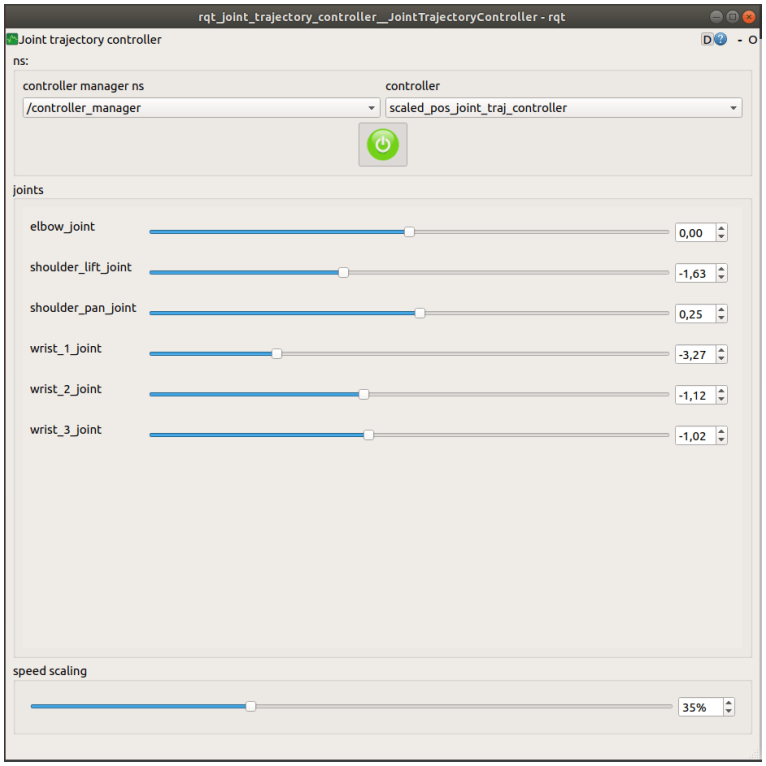

Figura 20: Interfaz gráfica del controlador de articulaciones de RQT

Este controlador permite al usuario comprobar si el robot está correctamente conectado y mover cada articulación de forma individual, pero no permite realizar ningún tipo de planificación.

Para poder trabajar de forma visual y poder realizar planificación de las trayectorias del robot, se ha optado por el uso de Moveit. Para lanzar Moveit se utilizan un comando para cargar la configuración de UR3e de Moveit y otro para lanzar la herramienta de visualización en RVIZ:

```
$ roslaunch ur3e_moveit_config moveit_planning_execution.launch
$ roslaunch ur3e_moveit_config moveit_rviz.launch
```

5.1.2. Pruebas en ROS1

Con esto ya podemos visualizar el robot en RVIZ y planificar movimientos de forma rápida y sencilla. Para comprobar que el driver del cobot funciona con los

planificadores de Moveit! se han diseñado 3 pruebas. La primera prueba sirve para comprobar que el robot se mueve correctamente desde una configuración final a una objetivo y en las otras dos pruebas se ha trabajado con varios escenarios con objetos para comprobar que la planificación de caminos es la correcta.

Para estas pruebas, se ha utilizado:

1. ROS1 Melodic como entorno de trabajo.
2. Driver de Universal Robots para ROS1 para conectar los controladores de ROS con Polyscope y permitirles el control del robot.
3. Se ha empleado el URCap "External Control" de Polyscope para conectar el robot al driver.
4. Moveit! como controlador y planificador de trayectorias.Se ha empleado el planificador OMPL que ofrece algoritmos para el cálculo de trayectorias.
5. Otros controladores como "rqt" que permiten controlar de forma manual el robot y comprobar su correcto funcionamiento.
6. Se ha utilizado el fichero URDF del robot completo para su visualización en RVIZ.

Todo esto genera los nodos, *topics* y funcionalidad necesaria

La primera prueba consistió un movimiento simple, donde se trata de colocar el robot en posición vertical. Para ello se inicia la acción con el robot en una posición aleatoria y se le ordena mediante el planificador de Moveit! que se coloque en posición vertical. En las figuras 21 y 22 se pueden observar tanto la posición inicial (en verde) como la posición final (en naranja) y la trayectoria seguida para realizar el movimiento establecido.

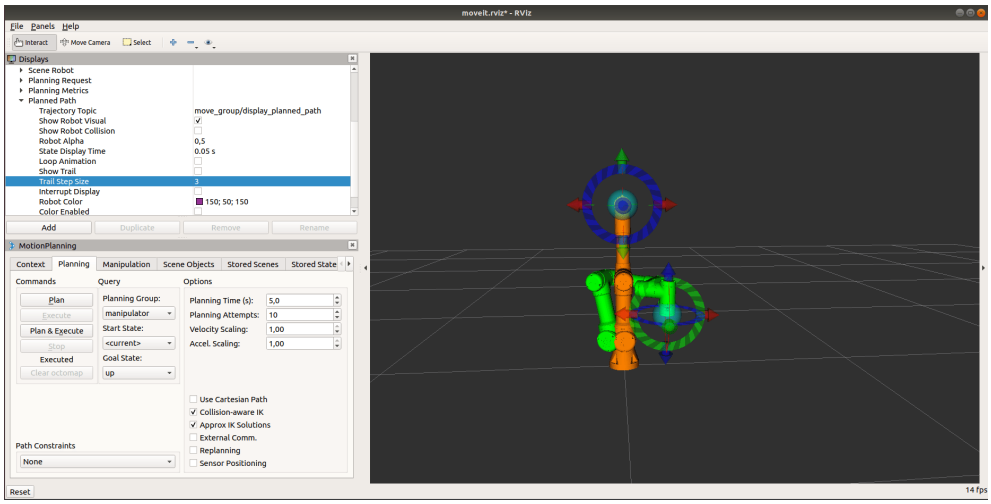

Figura 21: Movimiento simple con UR3e

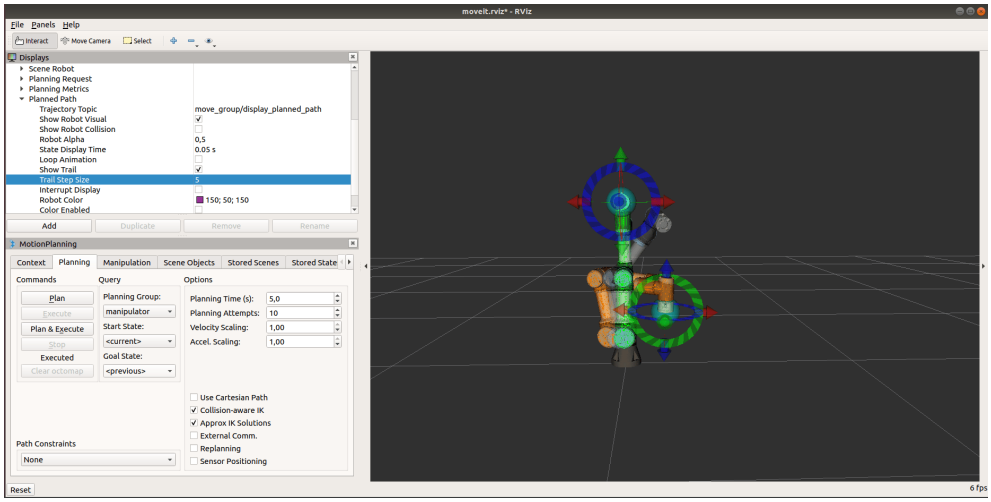

Figura 22: Trayectoria seguida en el movimiento simple

Aunque el visualizador da una imagen del movimiento del robot, el planificador se encarga de transformar esos movimientos en mensajes para los *topics* de control de las articulaciones que posteriormente serán interpretados por el driver y transformados en comandos que Polyscope sea capaz de procesar. Este movimiento se puede ver también en el robot real como se muestra en las figuras 23 y 24.

La segunda prueba consistió en simular un espacio de trabajo con objetos, donde hay que planificar una ruta libre de colisiones con los objetos del entorno, para que el robot se mueva desde una configuración inicial a otra final. Para simular un entorno real, se colocó un monitor de ordenador al lado del robot para comprobar si era capaz de planificar la ruta

Para la segunda prueba se intentó simular un entorno real colocando un monitor de ordenador al lado del robot, creando el escenario simulado para establecer las colisiones y finalmente comprobando si la planificación era correcta.

El entorno recreado en Moveit! consta de varios objetos que simulan una mesa en la que el robot se apoya junto a una pantalla que el robot debe esquivar. Como el objeto es bastante grande, el planificador tiene problemas al calcular rutas cortas, por lo que el movimiento que permite es bastante corto, levantándose por encima de la pantalla esquivándola. En la figura 25 puede observarse el escenario de Moveit! desarrollado para la prueba, donde se muestra también el brazo en su posición inicial en azul y su posición final en naranja:

Figura 23: Inicio del movimiento simple en el robot real

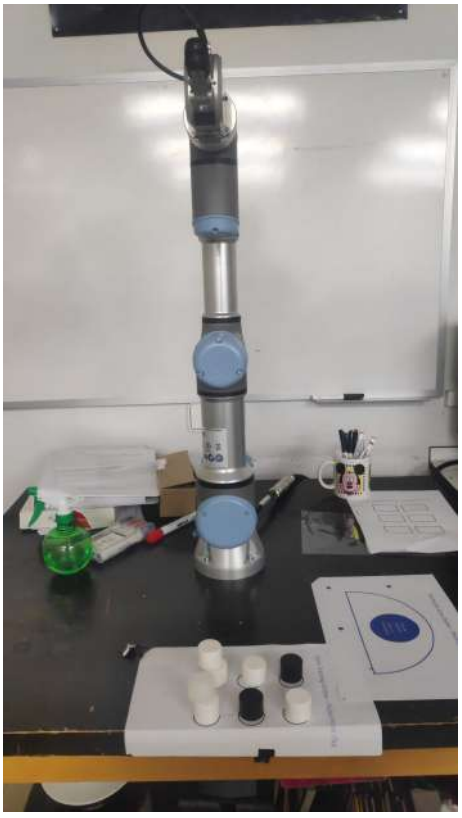

Figura 24: Fin del movimiento simple en el robot real

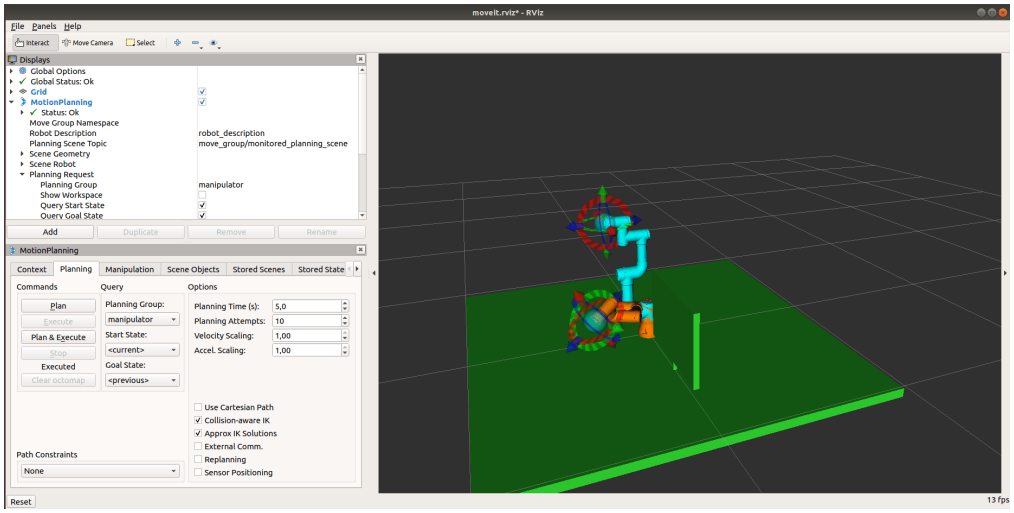

Figura 25: Escena simulada para un entorno real

Una vez ejecutada la trayectoria, se puede observar en las figuras 26, 27 y 28 el movimiento del brazo real desde la posición inicial a la final esquivando la pantalla colocada.

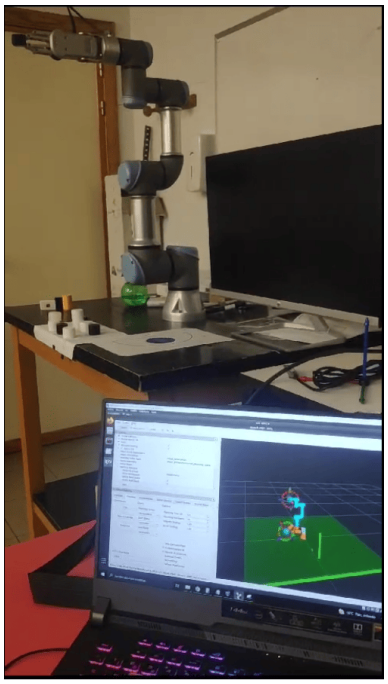

Figura 26: Inicio del movimiento del robot en un escenario real

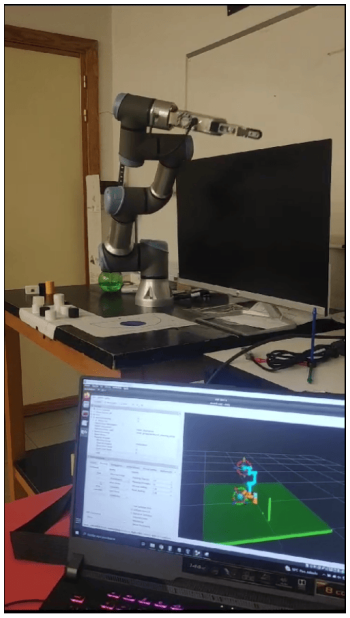

Figura 27: Movimiento intermedio del robot en un escenario real

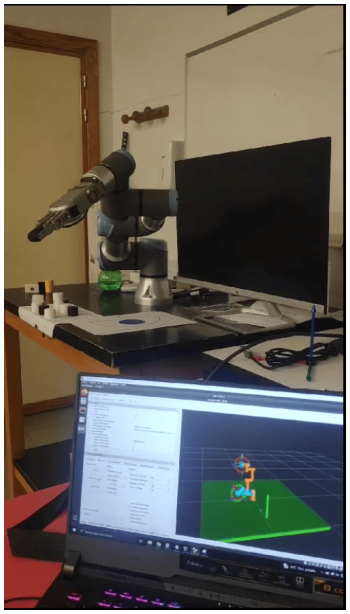

Figura 28: Fin del movimiento del robot en un escenario real

Con esta prueba, aparece un problema porque el robot tiene un espacio de trabajo pequeño y se han colocado objetos grandes: el planificador no es capaz de generar algunas trayectorias. Para tratar de resolver este problema, se decidió crear un tercer escenario con varias cajas flotantes simuladas (aunque no colocadas en el entorno real). Con esto se quería comprobar si, al colocar objetos más pequeños, el planificador era capaz de generar la trayectoria correcta para esquivar dichos objetos. En la figura 29 aparece el escenario simulado en Moveit!, con tres cajas verdes, y la configuración inicial del robot en azul y la configuración final en naranja. En las figuras 30, 31 y 32 se muestran tres configuraciones adoptadas por el robot real cuando ejecuta la trayectoria planificada por Moveit!

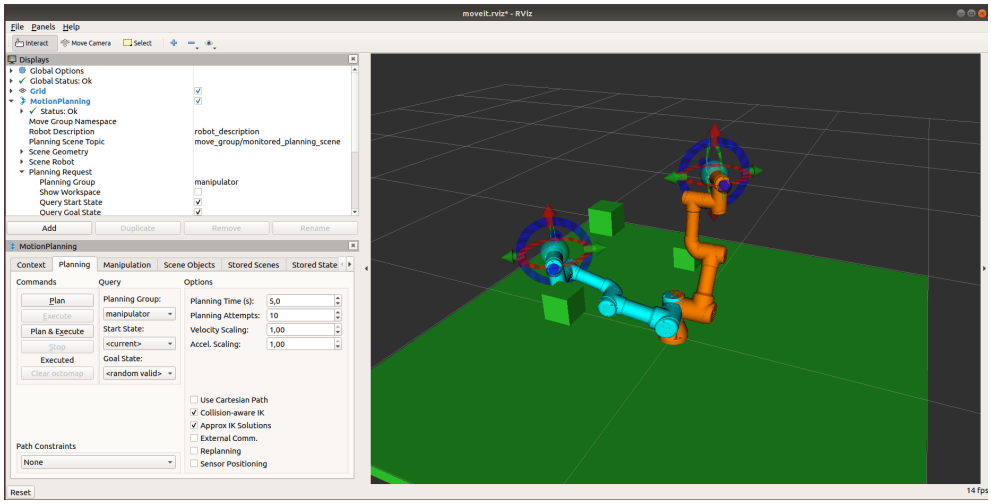

Figura 29: Escena con cajas simuladas en Moveit

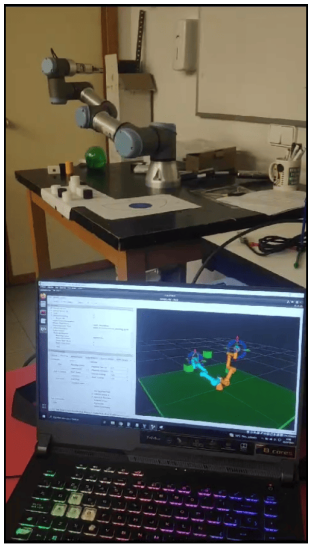

Figura 30: Inicio del movimiento del robot en un escenario con cajas simuladas

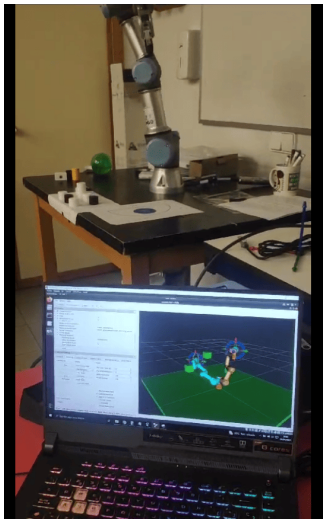

Figura 31: Movimiento intermedio del robot en un escenario con cajas simuladas

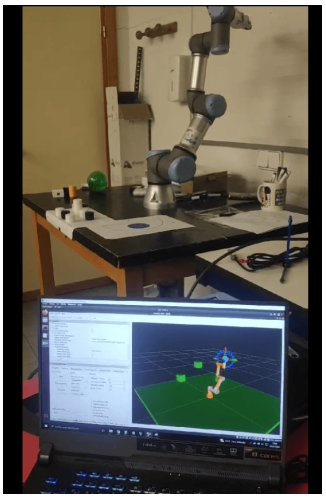

Figura 32: Fin del movimiento del robot en un escenario con cajas simuladas

Con estos ejemplos se ha comprobado que el driver del brazo funciona de forma correcta con los planificadores de Moveit.

El siguiente objetivo planteado sería añadir el funcionamiento la pinza para manipular objetos con el cobot, generando trayectorias de forma automática.

Para ello, se han investigado drivers que fueran compatibles para la pinza RG2 para ROS1 y que contengan un modelo URDF de la pinza. Solo hay disponibles de 2 drivers, ambos basados en el mismo driver de "ros_industrial" llamado "robotiq" [66].

Tras comprobar el funcionamiento de estos drivers y sabiendo que la pinza solo

funciona en Polyscope añadiendo el URCap necesario, no se ha conseguido ningún resultado positivo. Este driver trabaja mediante el protocolo MODBUS, que se trata de un protocolo de comunicación para el intercambio de información con arquitectura maestro-esclavo (Polyscope sería el maestro y el driver de la pinza el esclavo). Arrancando el driver de la pinza nunca se ha conseguido que conecte con Polyscope, por lo que la pinza real del brazo nunca se abre. Además, el driver está pensado para trabajar con la pinza únicamente, por lo que sería necesario extender el driver del brazo junto al driver de la pinza, todo ello unido, al igual que los ficheros URDF para que los planificadores puedan trabajar de forma correcta. El desarrollo del driver completo para generar la funcionalidad de detección de la fuerza empleada, el movimiento de la pinza y el movimiento completo del brazo resulta una tarea larga tratándose de ROS, por lo que este sistema se ha propuesto como la principal línea de trabajo futura.

5.2. ROS2

En este apartado se tratará de realizar las mismas tareas que en el apartado anterior pero trabajando con el driver de ROS2 en distintas versiones: Ros Foxy Fitzroy, ROS Humble Hawksbill y ROS Rolling Rodley. El principal problema, que se verá en cada apartado, es el "poco" tiempo de desarrollo de la arquitectura de ROS2 y de este driver en ella. Esto se ve agravado por el poco interés de la comunidad en migrar sus proyectos desde ROS1, por lo que la mayoría de desarrolladores siguen trabajando en ROS1. En ROS2 las versiones cambian de forma muy rápida, por lo que el desarrollo de paquetes se ve ralentizado por las adaptaciones que necesitan para cada versión y problemas que puedan surgir. Es por esto que se ha procedido a trabajar con varias versiones de ROS2 para comprobar el funcionamiento del driver de Universal Robots entre versiones.

5.2.1. ROS Foxy Fitzroy

Esta es una de las primeras versiones en las que se puede implementar el driver de Universal Robots para ROS2. Esta versión ya no dispone de soporte ya que como se ha comentado, el cambio de versiones es muy rápido y las versiones antiguas dejan de tener soporte a cambio de un mejor desarrollo en las nuevas.

Al igual que en ROS1, el punto inicial es el de crear un workspace de forma similar a como se hacía en dicha versión y descargar en él los paquetes necesarios para el robot. Los dos paquetes principales son: "Universal Robots ROS2 Driver" y "Universal Robots ROS2 Client". Estos dos paquetes contienen el driver y las diferentes funciones de calibración y conexión del robot para ROS2. Aunque estos dos paquetes son los principales, se dispone también de otros paquetes como RVIZ o Moveit que, a diferencia de ROS1, ahora se descargan de forma automática en el workspace.

Una vez compilados los paquetes con la herramienta Colcon, se pueden ejecutar tanto la función de calibración como el driver y Moveit! al igual que en ROS1 con

los siguientes comandos:

```
$ source /opt/ros/foxy/setup.bash
$ source ~/ros2_foxy/install/setup.bash
$ ros2 launch ur_calibration calibration_correction.launch.py robot_ip:=212.128.172.172 \
target_filename:="${HOME}/my_robot_calibration.yaml"
$ ros2 launch ur_bringup ur_control.launch.py ur_type:=ur3e robot_ip:=212.128.172.172 \
use_fake_hardware:=false launch_rviz:=true
$ ros2 launch ur_bringup ur_moveit.launch.py ur_type:=ur3e robot_ip:="212.128.172.172" \
use_fake_hardware:=false launch_rviz:=true
```

Con estos comandos se realiza la calibración del robot y se inician el driver y el planificador de Moveit! respectivamente. Los parámetros indicados corresponden al tipo del robot, su IP y si se quiere una simulación o controlar el robot real.

Aunque los paquetes funcionan correctamente, surge un error en esta versión de ROS2, tanto en el control del robot real como en la simulación. El problema en esta versión de ROS y que se extiende a otras versiones es la carga del modelo del robot. A la hora de arrancar el driver, y disponiendo de un modelo URDF correcto del robot con una calibración a medida, el driver no es capaz de cargarlo, lanzando un error indicando que el modelo es inválido (figuras 33 y 34).

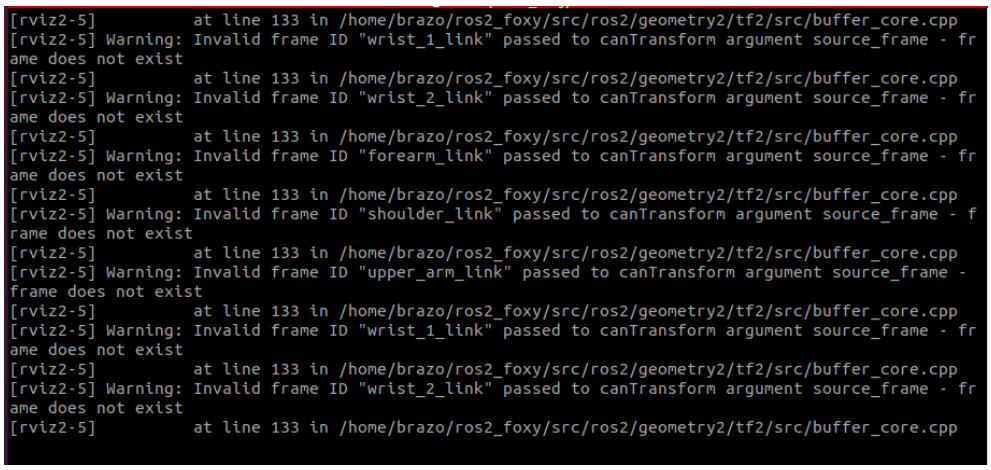

Figura 33: Error en la carga del modelo en el driver en ROS Foxy

Este error ocasiona que ni siquiera el driver funcione, por lo que no se puede realizar un control del robot en esta versión de ROS.

Una posible solución que se trató de implementar fue la de revisar el fichero URDF buscando aquellas articulaciones que puedan tener algún error en su descripción o incluso fallos de sintaxis, pero sin ningún éxito. Es un error ya que se trata de un error de la versión que se ha tratado de solucionar en otras versiones, pero que aún así se extiende.

Por todo ello, se decidió trabajar directamente con la versión ROS Humble, una versión mucho más moderna y con soporte actual.

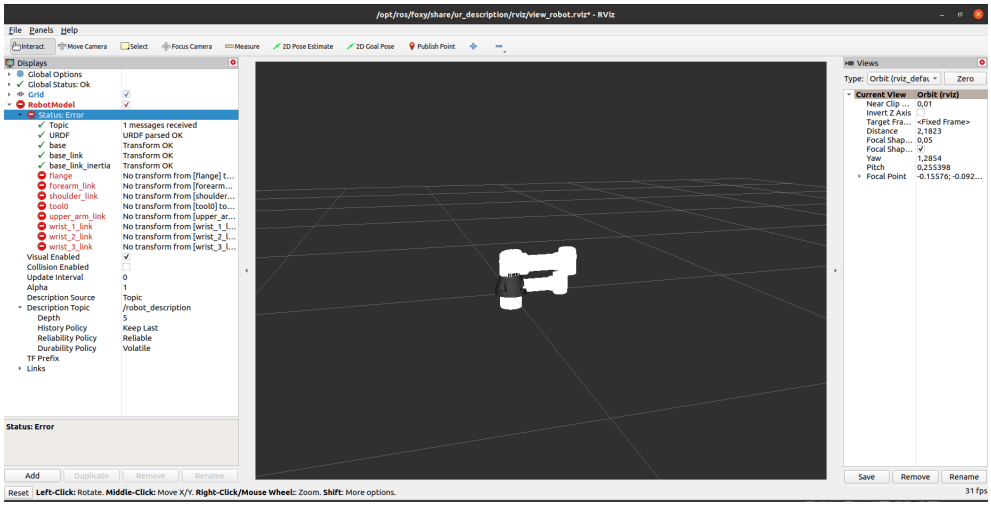

Figura 34: Error en la carga del modelo en RVIZ en ROS Foxy

5.2.2. ROS Humble Hawksbill

Esta versión de ROS2 es una de las últimas versiones lanzadas y dispone de soporte, tanto de ROS como del driver de Universal Robots.

En esta versión, el driver ha sido mejorado y los paquetes se han desarrollado de forma más modular, separando el paquete "ur_bringup", que era el principal encargado de lanzar toda la funcionalidad de otros paquetes en versiones anteriores, en otros paquetes como "ur_robot_driver", que contiene únicamente la funcionalidad del driver o "ur_calibration" con la funcionalidad de la calibración del robot. Aún así, la estructura y comandos de las versiones anteriores siguen funcionando en esta versión, pero el paquete está obsoleto.

Siguiendo los pasos de las versiones anteriores y ejecutando los siguientes comandos podemos realizar la calibración para, posteriormente, comprobar varias opciones tratando de arrancar el driver:

```
$ source /opt/ros/humble/setup.bash
$ source ~/ros2_ws/install/setup.bash
$ros2 launch ur_calibration calibration_correction.launch.py robot_ip:=212.128.172.172 \
target_filename:="${HOME}/my_robot_calibration.yaml"
```

Una vez la calibración está completa tenemos varias opciones:

1. Ejecutar el driver desde el paquete "ur_bringup" al igual que se hacía en ROS Foxy.
2. Ejecutar el driver desde el launch "ur3e.launch.py" con Moveit.
3. Ejecutar el driver desde el nuevo paquete "ur_robot_driver".

Estas 3 opciones pueden ejecutarse con los siguientes comandos:

```
- Ejecución desde el paquete ur_bringup  con el launch ur_control.launch.py
$ ros2 launch ur_bringup ur_control.launch.py ur_type:=ur3e robot_ip:=212.128.172.172 \
use_fake_hardware:=false launch_rviz:=true
$ ros2 launch ur_moveit_config ur_moveit.launch.py ur_type:=ur5e launch_rviz:=true

- Ejecución desde el paquete ur_bringup con el launch ur3e.launch.py
$ ros2 launch ur_bringup ur3e.launch.py robot_ip:=212.128.172.172 use_fake_hardware:=false
$ ros2 launch ur_moveit_config ur_moveit.launch.py ur_type:=ur5e launch_rviz:=true

- Ejecución desde el paquete ur_robot_driver
$ ros2 launch ur_robot_driver ur3e.launch.py robot_ip:=212.128.172.172
$ ros2 launch ur_moveit_config ur_moveit.launch.py ur_type:=ur5e launch_rviz:=true
```

Con cualquiera de estas opciones se obtienen resultados similares a la versión ROS Foxy como se puede observar en la figura 35.

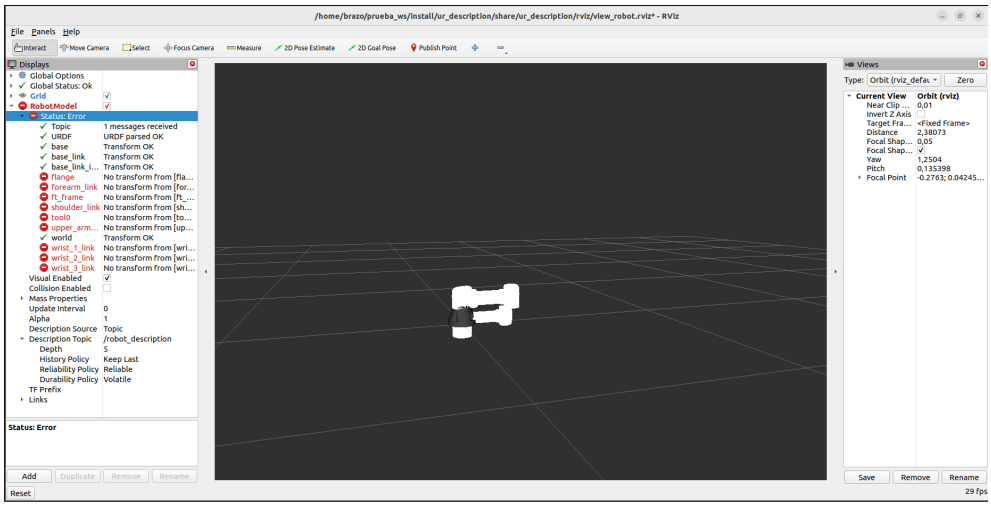

Figura 35: Error en la carga del modelo en el driver en ROS Humble

El modelo del robot no se carga correctamente y no permite arrancar ni el driver ni el planificador. Este error ya ha sido reportado por otros desarrolladores en el propio repositorio del driver: UniversalRobots. "Issues with IKFast Plugin in ROS 2 driver". Recuperado de `https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver/issues/724`.

Finalmente, como última prueba e investigando una versión en la que se tenga certeza de que el driver funciona, se decidió trabajar con la versión ROS Rolling.

5.2.3. ROS Rolling Ridley

Esta versión sirve como campo de pruebas para el desarrollo de ROS2. En esta versión de ROS se implementan todos los nuevos cambios que se implementarán en las versiones "oficiales" de ROS, por lo que no es recomendable trabajar en proyectos grandes con esta versión por posibles errores que puedan surgir en su desarrollo. Sin embargo, para este proyecto, es la última opción para conseguir mover el robot desde ROS2, pues es la versión con mejores y las últimas actualizaciones.

Al igual que en las versiones anteriores, creando el workspace, descargando y compilando los paquetes podemos ejecutarlos con los siguientes comandos:

```
$ source  /opt/ros/rolling/setup.bash
$ source ~/ros_ws/install/setup.bash
$ ros2 launch ur_calibration calibration_correction.launch.py robot_ip:=212.128.172.172 \
target_filename:="${HOME}/my_robot_calibration.yaml"
$ ros2 launch ur_robot_driver ur3e.launch.py robot_ip:=212.128.172.172
$ ros2 launch ur_moveit_config ur_moveit.launch.py ur_type:=ur3e
```

A diferencia de las otras versiones de ROS2, esta vez el driver sí funciona y arranca correctamente, cargando el modelo del robot en RVIZ y permitiendo el movimiento del robot. En las figuras 36 y 37 se muestra tanto la conexión correcta del robot con el driver como el modelo URDF cargado de forma correcta en RVIZ.

```
[spawner-8] [INFO] [1688748482.611661017] [spawner_io_and_status_controller]: Configured and activated io_and_status_controller
[ur_ros2_control_node-1] [INFO] [1688748482.613139181] [io_and_status_controller]: Configure UR gpio controller with tf_prefix:
[INFO] [spawner-10]: process has finished cleanly [pid 3638]
[INFO] [spawner-11]: process has finished cleanly [pid 3642]
[INFO] [spawner-7]: process has finished cleanly [pid 3632]
[INFO] [spawner-6]: process has finished cleanly [pid 3630]
[INFO] [spawner-9]: process has finished cleanly [pid 3636]
[INFO] [spawner-8]: process has finished cleanly [pid 3634]
[ur_ros2_control_node-1] [INFO] [1688748529.462106107] [UR_Client_Library]: Robot requested program
[ur_ros2_control_node-1] [INFO] [1688748529.462230131] [UR_Client_Library]: Sent program to robot
[ur_ros2_control_node-1] [INFO] [1688748529.629213835] [UR_Client_Library]: Robot connected to reverse interface. Ready to receive control commands.
[ur_ros2_control_node-1] [INFO] [1688748529.633965365] [io_and_status_controller]: Configure UR gpio controller with tf_prefix:
```

Figura 36: Conexión del robot con el driver en ROS2

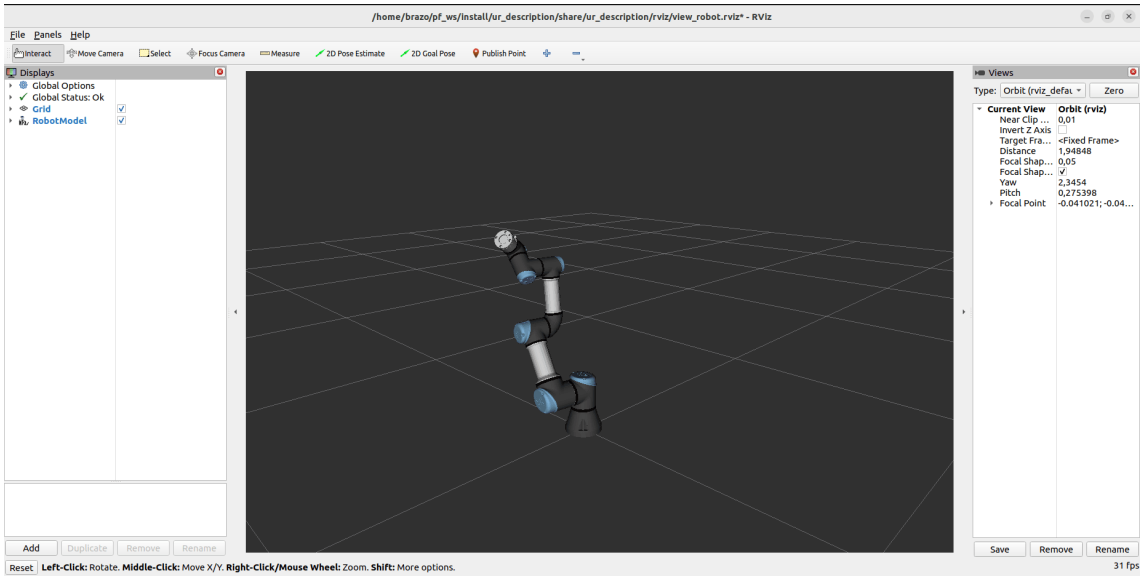

Figura 37: Modelo del robot en RVIZ en ROS2

Con esto, el robot está conectado y listo para ser controlado. En los comandos anteriores, también se arrancaba, a la vez que el driver, el planificador de Moveit. Como ya se ha comentado, las versiones de ROS2 son mucho más inestables que las de ROS1 y tienen multitud de problemas. En la prueba hecha con ROS Rolling, el error de carga de los ficheros URDFs está solucionado ya que era un error debido a un bug existente en ROS2 y no en los propios paquetes. A la vez que este error se ha solucionado han surgido otros como un problema en el arranque de Moveit!! (figuras 38 y 39).

En esta versión de ROS, aunque el driver funciona, los paquetes son muy inestables. A la hora de realizar las pruebas, el paquete de Moveit no era capaz de cargar

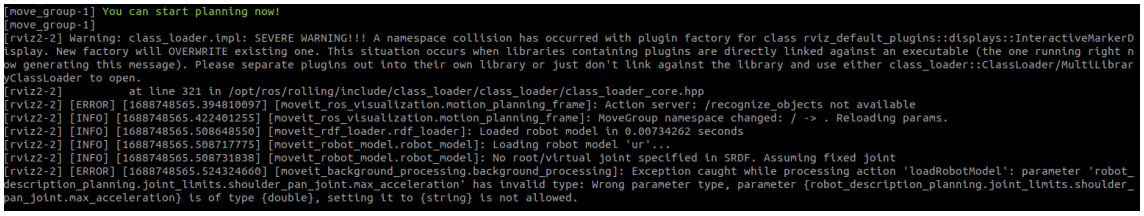

Figura 38: Error en Moveit en ROS2 Rolling

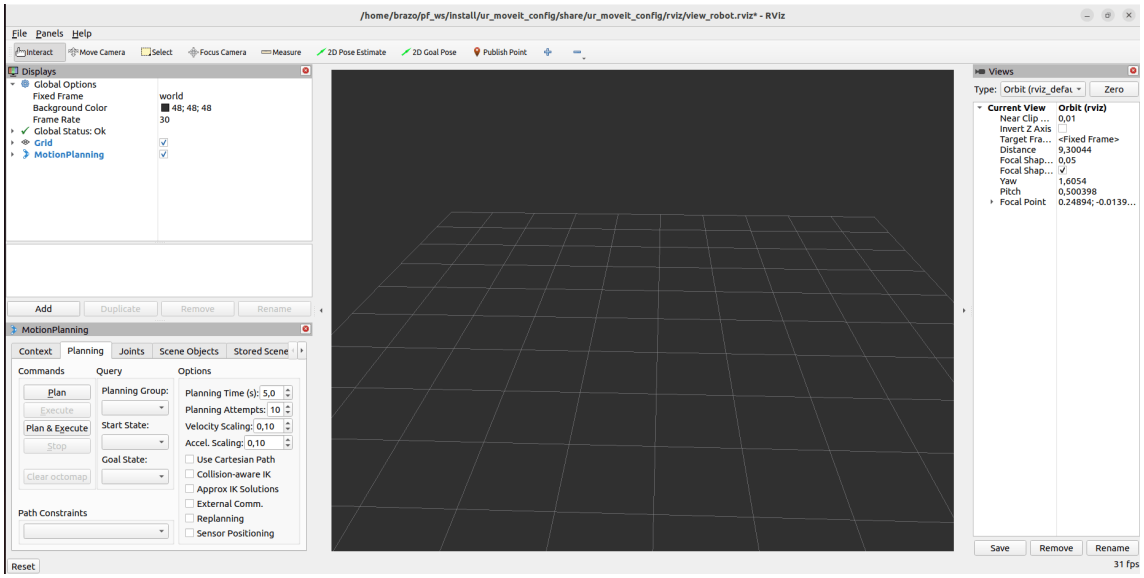

Figura 39: Moveit RVIZ error en ROS2 Rolling

el modelo del robot, indicando que uno de los ficheros de descripción del robot donde se establece un parámetro de la máxima aceleración para una articulación no es del tipo correcto. Para solucionarlo se trató de modificar el tipo de dato e incluso modificar la descripción completa del robot, pero el error seguía apareciendo.

Aún así, aunque el planificador Moveit! no funcionara, el robot podía moverse. Para ello, se utilizó el controlador de rqt, tratando de controlar el robot de forma manual y comprobar que el driver estaba funcionando correctamente. Al igual que sucedía en ROS1, enviando la información correcta de movimiento a los *topics* a través de un controlador muy sencillo se podía controlar el robot. Este controlador se puede ejecutar con el siguiente comando:

```
$ rqt --standalone rqt_joint_trajectory_controller
```

Una vez arrancado el driver, se despliegan todos los nodos y *topics* necesarios para el control del robot (figuras 40 y 41).

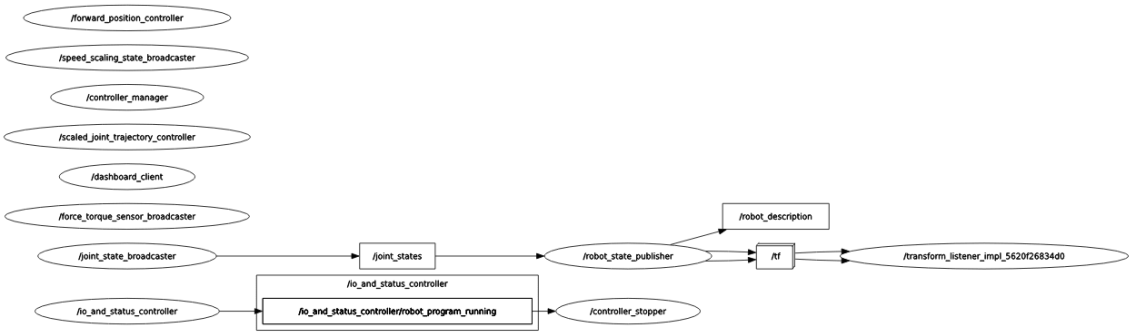

Figura 40: Grafo de nodos y *topics* en ROS2

```
brazo@brazoTFM:~$ ros2 topic list
/clicked_point
/diagnostics
/dynamic_joint_states
/force_torque_sensor_broadcaster/transition_event
/force_torque_sensor_broadcaster/wrench
/forward_position_controller/commands
/forward_position_controller/transition_event
/goal_pose
/initialpose
/io_and_status_controller/io_states
/io_and_status_controller/robot_mode
/io_and_status_controller/robot_program_running
/io_and_status_controller/safety_mode
/io_and_status_controller/tool_data
/io_and_status_controller/transition_event
/joint_state_broadcaster/transition_event
/joint_states
/parameter_events
/robot_description
/rosout
/scaled_joint_trajectory_controller/controller_state
/scaled_joint_trajectory_controller/joint_trajectory
/scaled_joint_trajectory_controller/transition_event
/speed_scaling_state_broadcaster/speed_scaling
/speed_scaling_state_broadcaster/transition_event
/tf
/tf_static
```

Figura 41: *topics* en ROS2

Como el planificador no arranca, disponemos de muchos menos *topics* asociados al mismo. Los *topics* desplegados son los siguientes:

1. Los *topics* "joint_state_broadcaster", "joint_states", "robot_state_publisher" y "robot_description" se asemejan a los *topics* de ROS1. Trabajan con la información sobre el estado de las articulaciones y del estado del modelo del robot (posiciones de las articulaciones en cada momento) y mostrar las diferentes posiciones de cada articulación en el RVIZ, permitiendo configurarlas en tiempo real.

2. Los *topics* de la rama de "io_and_status_controller" se corresponden a los *topics* de la rama "ur_hardware_interface" en ROS1. Todos los *topics* correspondientes a estas ramas se encargan de la comunicación entre ROS y Polyscope, obteniendo y enviando toda la información relativa al estado del robot, el modo activo del mismo, si hay algún programa en ejecución en el cobot, etc.

3. Los *topics* 'initial_pose" y "goal_pose" sirven para especificar en RVIZ la posición inicial del robot a la hora de cargar el modelo o para acciones del controlador para indicarle la posición que se desea alcanzar.

4. Los *topics* "scaled_joint_trajectory_controller", "speed_scaling_state_broadcaster", "force_torque_sensor_broadcaster", "dashboard_client" y "controller_manager" proporcionan información sobre el robot, sobre el control de trayectoria y velocidad, la fuerza que están detectando sus sensores o información de control del robot.

5. Existen otros *topics* que se inician de forma automática por ROS como "ros_out" y que son *topics* con los que trabaja ROS y ofrecen información de depuración.

En resumen, los *topics* generados por el driver de ROS2 son equivalentes a los *topics* generados en ROS1. Ambos framework deben proporcionar la información correcta para que Polyscope la interprete. Es por esto que los *topics* generados al lanzar el driver son equivalentes aunque con distintos nombres.

El principal problema con ROS2, como ya se ha comentado, es su alta variabilidad. Las versiones de ROS2 son muy cambiantes y aparecen nuevas versiones de forma muy rápida, por lo que continuamente aparecen errores que de una forma u otra ralentizan el desarrollo de las aplicaciones.

El resultado obtenido en ROS2 no ha sido concluyente, por lo que se decidió no solo comparar las versiones de ROS1 y ROS2, sino también el software Polyscope de Universal Robots para comprobar la diferencia entre un software libre y que sirve para multitud de aplicaciones frente a un software propietario y específico para los cobots de UR.

5.3. UR Polyscope

Polyscope es una interfaz gráfica basada en iconos y menús desplegables, diseñada para facilitar la programación y el control del cobot UR3e.

Para tratar de comparar la forma en la que Polyscope facilita el desarrollo de tareas, se han desarrollado 2 programas:

1. Un programa de paletizado, donde se le proporciona al cobot diferentes piezas en un lugar concreto y las coloca ordenadas siguiendo un patrón de paletización. Se trata de un pick and place pero con el propósito de colocar de forma ordenada objetos. Como ejemplo de aplicación industrial, podrían llegar desde una cinta transportadora y el robot se encargaría de colocarlos para el transporte.

2. Un programa de pintado, donde el robot con un rotulador sujeto con la pinza es capaz de pintar cualquier tipo de dibujo en una pizarra. Esto se realiza a través de una URCap, donde dado un trazo específico como una trayectoria en GCODE (lenguaje de programación en CNCs), el robot es capaz de reproducirlo. Como ejemplo de aplicación industrial, este mismo programa podría utilizarse a modo de CNC en el fresado y corte de pieza mecanizadas.

5.3.1. Paletizado

Para el paletizado se dispone de unas marcas a modo de rejilla y unas piezas con forma cilíndrica que el cobot tendrá que mover y colocar. Esta rejilla está hecha con marcas en un papel como se muestra en la figura 42.

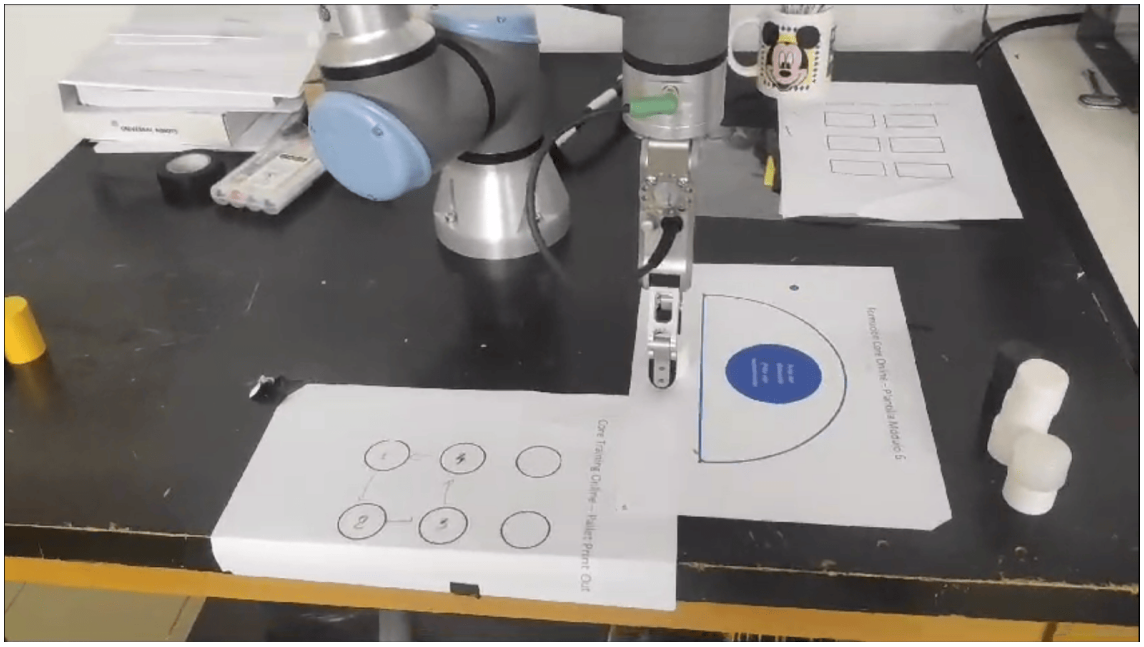

Figura 42: Rejilla para el paletizado

Esta rejilla tiene 6 posiciones y se dispone de 8 figuras, por lo que para el paletizado se establecerán estas 6 posiciones y varias capas en altura para colocar de 6 en 6 piezas por capa (en este caso, colocará 6 piezas en la primera capa y 2 en la segunda capa). El robot coge piezas desde la posición que se puede observar en la figura 42, por lo que el primer paso será desplazar el robot hasta la posición de surtido de piezas. Una vez el robot coja la pieza, se encargará de realizar el movimiento hasta la siguiente posición libre de la rejilla (o de la capa correspondiente). Estos movimientos se repetirán hasta que el usuario que colabora con el robot coloque todas las piezas.

Todo esto se ha realizado en un programa de paletizado que se muestra en las figuras 43 y 44

Figura 43: Parte 1 del programa de paletizad

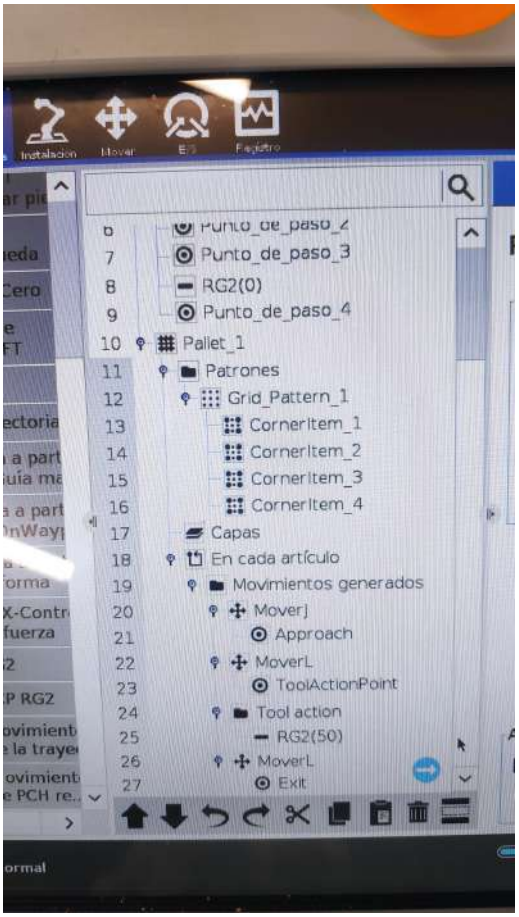

Figura 44: Parte 2 del programa de paletizado

Siguiendo este programa paso a paso se pueden ver las funcionalidades que permite Polyscope. En primer lugar, el robot tiene que alcanzar el punto de recogida de las piezas. Para ello se emplea un URCap denominado "RG2" que permite controlar la pinza (*líneas 2, 8 y 25 del programa*, abriéndola antes de conseguir alcanzar la pieza. Este URCap permite abrir y cerrar la pinza y controlar la fuerza que ejerce (figura 45). Para mover y colocar el robot en una posición determinada también se dispone de funciones para controlar dónde va a moverse el robot o incluso poder moverlo libremente con la mano y grabar la posición que se quiere alcanzar (líneas 4,6,7,9,21,23 y 27 del programa). Estos movimientos se pueden ver en la figura 43, donde se abre la pinza, se coloca el robot en la posición de la pieza a recoger y se cierra la pinza. Los puntos de paso son puntos establecidos de forma manual con el movimiento libre donde se dirige el robot, aunque hay un panel de control para establecer estas posiciones (figura 46).

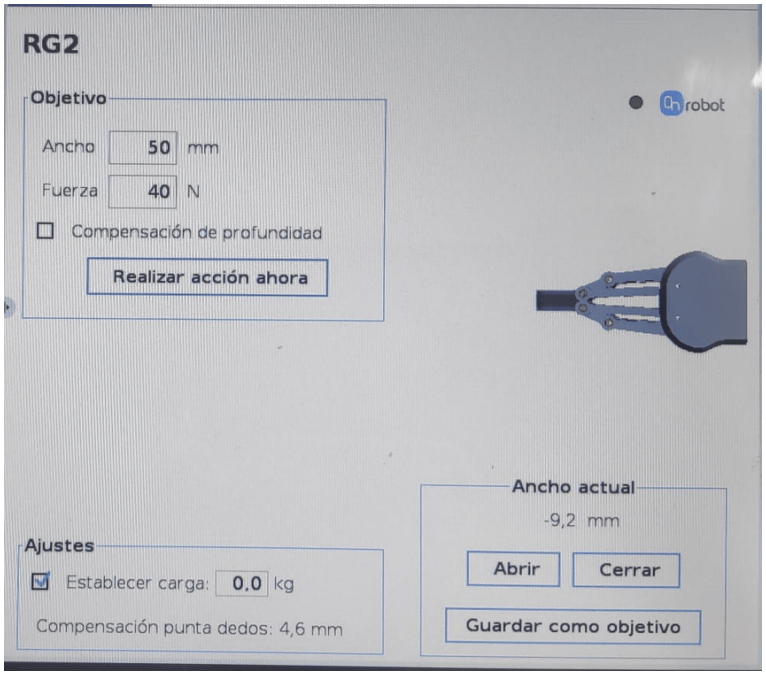

Figura 45: URCap para controlar la pinza RG2

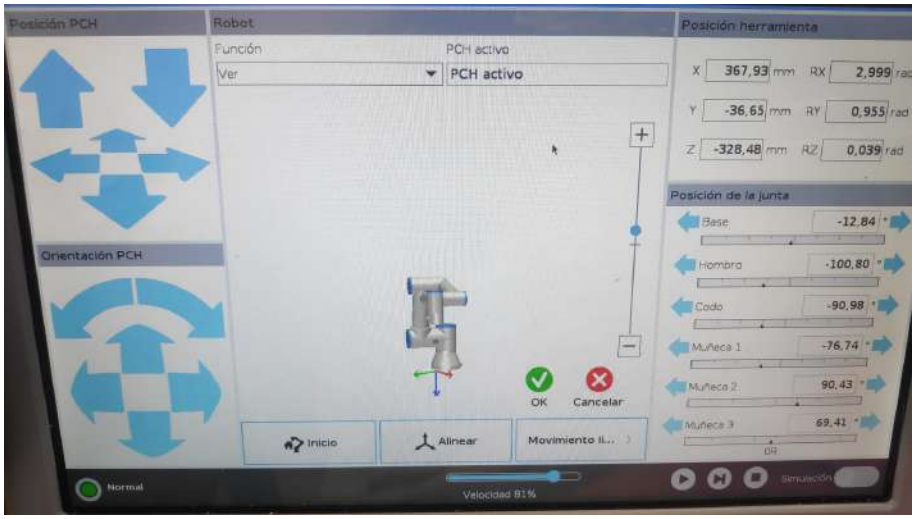

Figura 46: Panel para establecer posición del cobot

Al finalizar estos pasos, el robot ya ha cogido la pieza con la pinza. Es ahora cuando realmente comienza el paletizado.

Para el paletizado hay que establecer los siguientes conceptos que se utilizarán para colocar las piezas en los lugares establecidos:

1. En primer lugar hay que establecer el tipo de "grid" que se utiliza para colocar las piezas, establecer el número de filas y columnas y colocar el robot en las 4 esquinas del grid (definir las esquinas es la misma acción que establecer un punto de paso). Estos pasos se corresponden a las líneas 11 a 16 del programa.

Con esto, el software del cobot se encargará de realizar los cálculos necesarios para obtener todas las posiciones de la rejilla. En este caso, es una cuadrícula de 2 filas y 3 columnas. Estos parámetros se pueden establecer como se indican la figura 47.

2. En segundo lugar hay que establecer el número de capas que va a tener y si todas son iguales o son de distinto tipo (línea 17 del programa). En este caso, habrá dos capas siguiendo la misma estructura de cuadrícula establecida (figura 48)

3. Ahora, para cada pieza de la rejilla hay que establecer 3 movimientos: el movimiento de entrada a la posición, la acción a realizar y la salida. El movimiento de entrada y salida se establece de forma semejante a los puntos de paso descritos en el primer punto (líneas 18 a 27 del programa). Respecto a la acción,. hay que abrir la pinza liberando el objeto.

4. Finalmente, el cobot regresará a la posición inicial para comenzar con la siguiente pieza.

Al completar la programación de todos los pasos previos, el programa está listo para ser ejecutado. Una muestra del proceso y del resultado final se puede ver en las figuras 49 y 50.

5.3.2. Pintado

En el proceso de pintado hay que definir una trayectoria que realice el cobot, al que se le ha colocado en la pinza un rotulador o, en una aplicación industrial, un herramienta de pintura. Estas trayectorias, en otros campos como las impresoras 3D, se realizan mediante los comandos especificados en ficheros GCODE, que indican los movimientos que debe seguir.

Los robots de Universal Robots disponen de un URCap denominado “Remote RCP & Toolpath” que permite cargar un GCODE generado para un CNC y reproducir esos movimientos.

Los pasos a seguir son:

1. En primer lugar,se exporta el GCODE de un trazado del dibujo, utilizando un programa como Fusion 360, y se carga mediante un USB en Polyscope (figura 52). En este caso, se ha cargado el fichero GCODE para pintar un mandala.

2. Una vez cargado, se genera un programa únicamente con el URCap. Este URCap permite seleccionar la trayectoria que hemos importado y un plano donde se va a realizar (figura 53).

3. Se define el plano, en este caso, vertical, se va a pintar sobre una pizarra en la pared (figura 54).

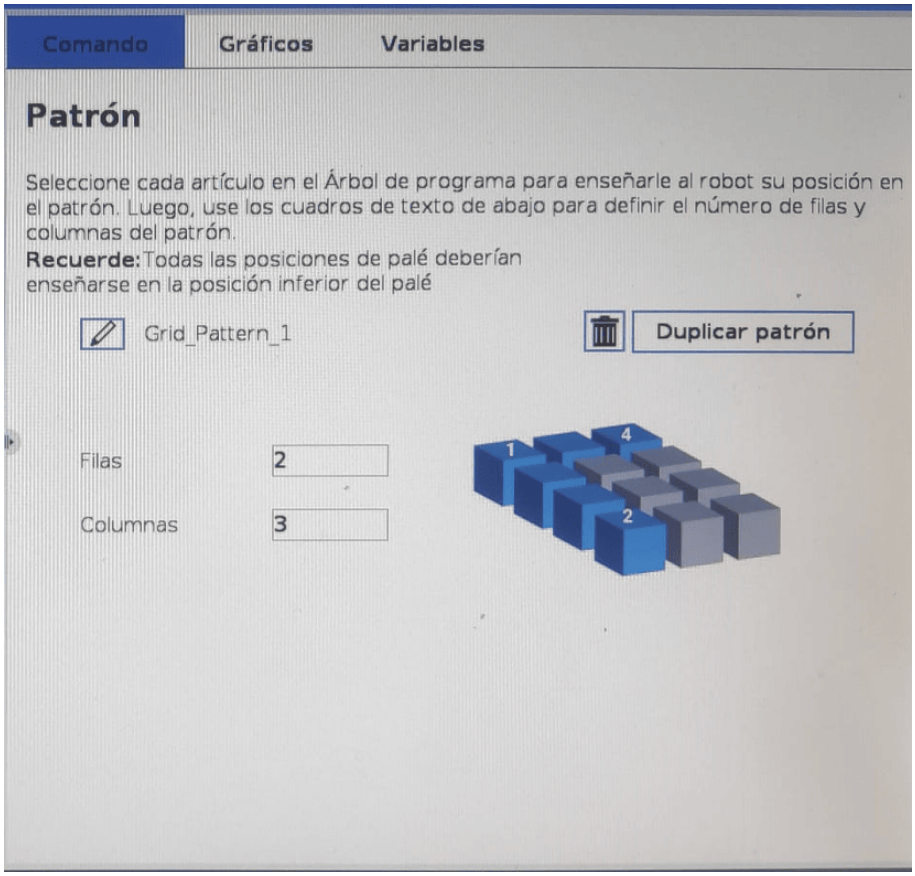

Figura 47: Cuadrícula para el paletizado

Cuando se ejecuta, el resultado final puede verse en la figura 55.Gracias a las herramientas proporcionadas por Polyscope, mediante una interfaz de alto nivel para el usuario que controla el robot, se facilita la programación del cobot sin necesidad de tener conocimientos avanzados.

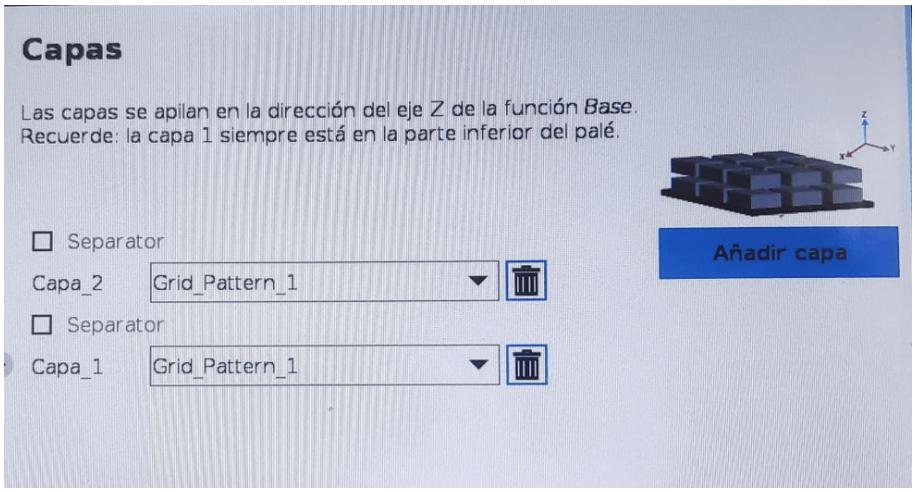

Figura 48: Capas del paletizado

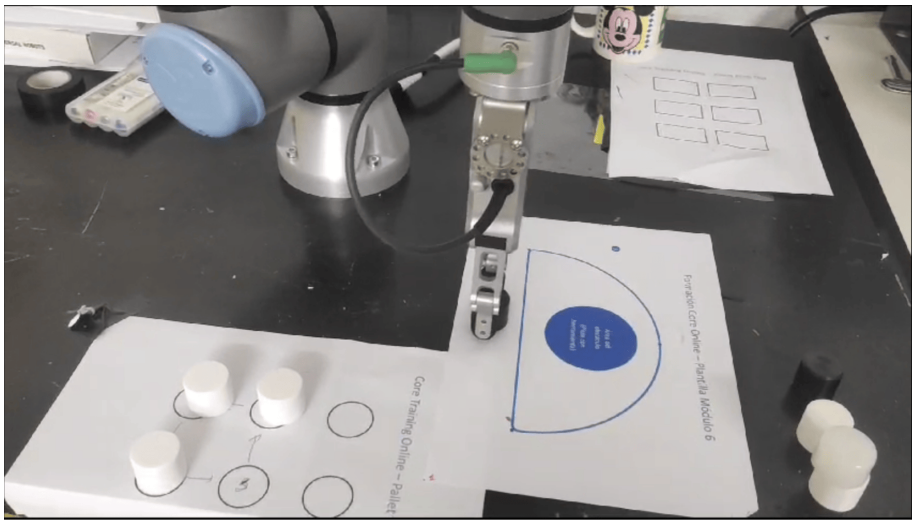

Figura 49: Cobot recogiendo la 4a pieza del paletizado

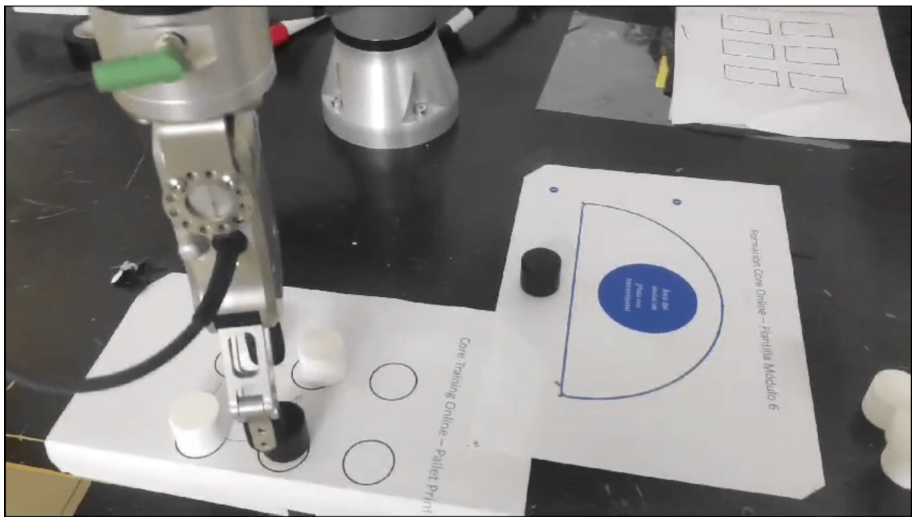

Figura 50: Cobot colocando la 4a pieza del paletizado

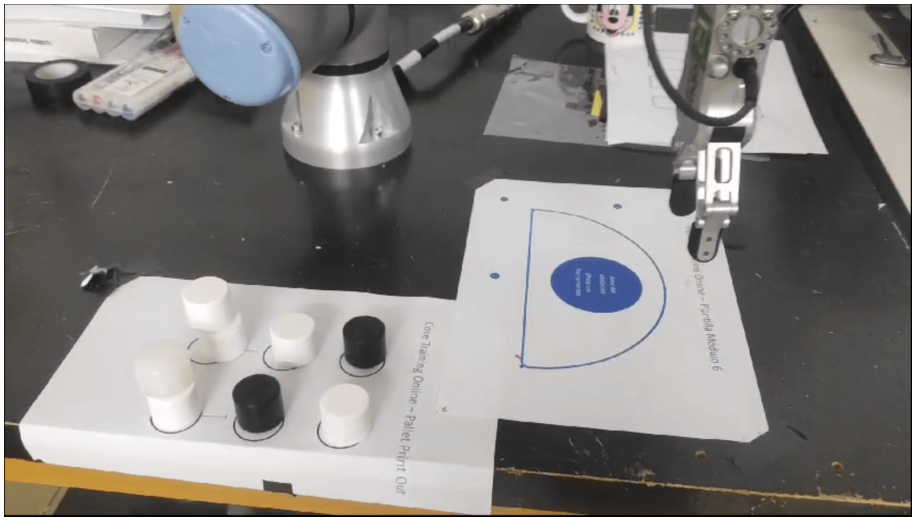

Figura 51: Resultado final del paletizado con 2 capas

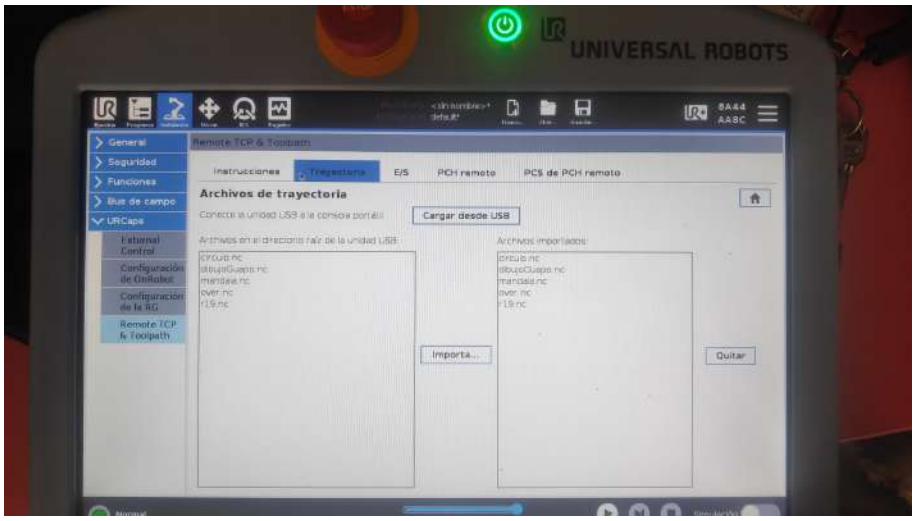

Figura 52: Carga del fichero GCODE en Polyscope

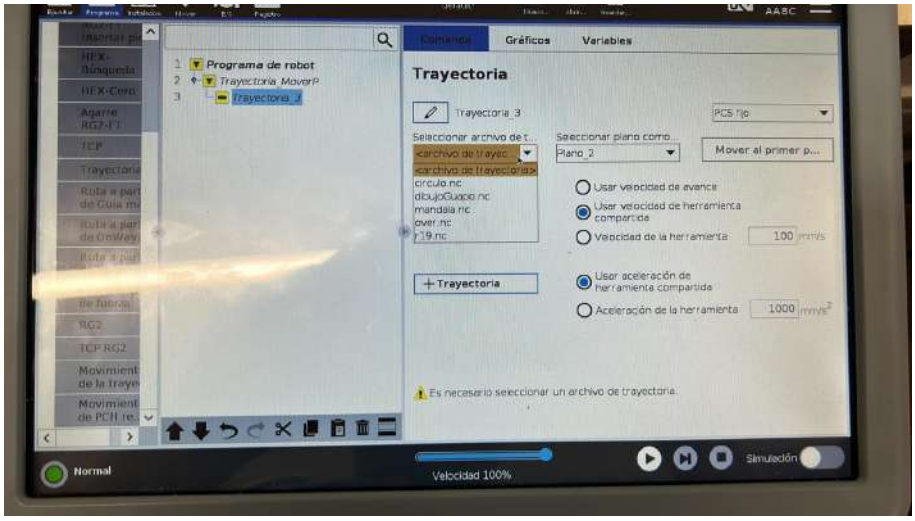

Figura 53: Programa de pintado

Figura 54: Definición del plano de pintado

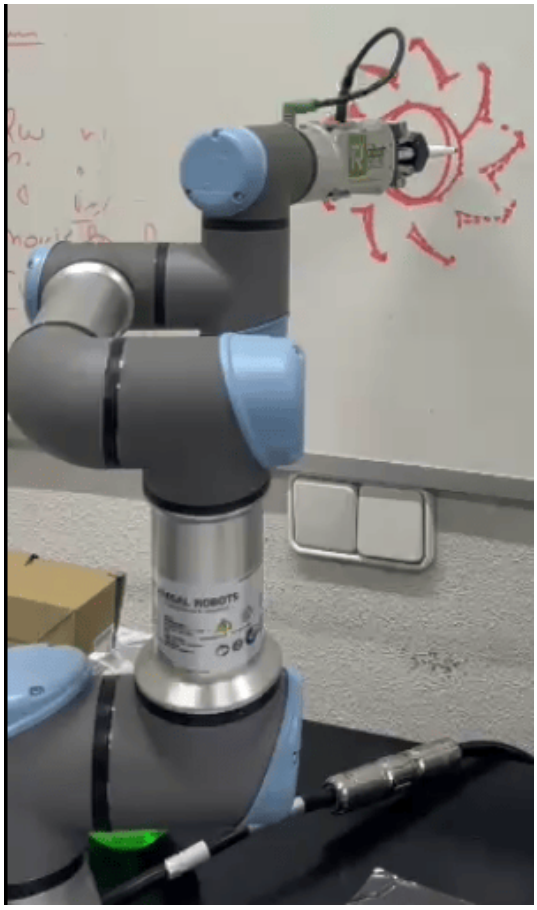

Figura 55: Resultado final del pintado del mandala

6. Conclusiones

Con el desarrollo de este trabajo de investigación se ha llevado a cabo la integración del cobot UR3e de Universal Robots en el entorno de ROS. Con esto se ha permitido conocer de primera mano la dificultad que conlleva realizar esta integración y los resultados que se pueden obtener con las versiones actuales del software. Esto ha permitido no solo aprender sobre cómo integrar un robot colaborativo dentro de un entorno OpenSource, sino que ha permitido investigar a fondo el funcionamiento de un framework robótico con elevada importancia en el sector industrial. También se ha conseguido adquirir los conocimientos para integrar robots de construcción propia o customizados para desarrollar otros proyectos.

Durante el desarrollo de este trabajo de investigación, se han realizado estudios en simulación e integración de un cobot real UR3e dentro del entorno ROS, tanto ROS1 como ROS2, así como la posibilidad de realizar expansiones a estos proyectos incorporando escenarios u otras funcionalidades. También se ha tratado de integrar una pinza robótica en un cobot articulado, que, aunque el resultado no ha sido exitoso, ha permitido adquirir conocimientos sobre el funcionamiento de este tipo de instrumentos de manipulación de objetos e incluso de herramientas software para su control e integración en otros robots.

También, el trabajo se ha centrado en estudiar el software Polyscope, también se ha centrado la investigación en el software Polyscope. La idea de disponer de un brazo robótico con un software tan elaborado y con tantas funcionalidades ha permitido poder comparar las funcionalidades entre un software genérico y opensource como es ROS frente a un software creado por una empresa exclusivamente para este tipo de actuadores. Con este software se han creado 2 programas que permiten realizar un pick-and-place usando el cobot UR3e junto a la pinza RG2. El propósito del primer programa es paletizar materiales, que tiene una aplicación directa en el sector industrial para el almacenaje y la logística. El segundo programa permite cargar y realizar secuencias de movimientos a partir de una trayectoria dada como GCODE, lo que da pie a multitud de aplicaciones industriales, como el pintado o el fresado de objetos.

Con este proyecto, se ha conseguido integrar el brazo UR3e en un entorno ROS y estudiar su funcionamiento. En particular, la integración en ROS 1 ha resultado muy satisfactoria, permitiendo realizar trayectorias planificadas utilizando Moveit junto al robot real. Por otro lado, la integración en ROS2 ha resultado una experiencia complicada. ROS2 ha presentado multitud de problemas asociados a las distintas versiones de ROS2, por lo que ha sido necesario investigar sobre varios sistemas operativos y varias versiones de ROS para obtener un resultado positivo que es el arranque del driver en ROS Rolling. Se ha conseguido conectar el driver de ROS2 a Polyscope, permitiendo realizar movimientos manuales con un controlador sencillo, pero rno se ha conseguido integran en los planificadores de Moveit!. Esto asegura que, aún habiendo conseguido algunos resultados positivos, se necesita un mejor desarrollo y más tiempo para que los drivers y los controladores funciones de una forma más controlada y segura.

En cuanto a la aplicación de la pinza RG2 de OnRobot para esta integración, se ha conseguido observar que muchas de pruebas realizadas en otros artículos como el artículo [67] son sobre simulaciones sin aplicar un driver real que controle al cobot junto a la pinza.

Para finalizar este proyecto, se pueden consultar los vídeos de demostración en siguiente enlace [68].

En resumen, esta investigación ha permitido profundizar sobre el sistema operativo ROS en sus distintas versiones, adquiriendo conocimientos sobre su funcionamiento y permitiendo estudiar las diferencias y similitudes entre ROS1 y ROS2 y el funcionamiento de sus sistemas de paquetes reutilizables que permiten realizar tareas complejas de forma mucho más sencilla. También ha permitido trabajar con un software no libre como es Polyscope, estudiar su funcionamiento y poderlo comparar con ROS como entorno contrario a Polyscope. Finalmente, ha ayudado a adquirir conocimientos sobre los robots articulados y las pinzas robóticas, así como los costes necesarios para trabajar con ellos y la dificultad que presentan para integrarlos en entornos más genéricos.

7. Líneas de trabajo futuras

En base a los resultados y conclusiones obtenidos con este proyecto, se proponen algunas líneas de trabajo futuras que permitan mejorar el desarrollo y adquirir nuevos conocimientos en esta área de investigación:

1. Desarrollo del driver completo de la pinza RG2 para ROS. Se propone desarrollar un driver propio que permita trabajar con la pinza RG2 desde ROS a través de *topics* específicos para ello, permitiendo agarrar objetos o incluso actuar con control de fuerza.

2. Integrar el driver de la pinza RG2 anterior junto con el driver del cobot UR3e para Polyscope. Con esto se conseguiría realizar planificación de movimientos al igual que se ha conseguido hacer durante esta investigación pero esta vez con un actuador final que permitiera trabajar sobre escenarios reales o simulados.

3. Realizar un estudio sobre los diferentes planificadores de Moveit con el driver completo dentro de ROS1 y ROS2. Esto permitiría conocer mejor los algoritmos empleados en la planificación de movimientos y poder estudiar las diferencias entre ellos, adquiriendo conocimientos sobre su uso en diferentes aplicaciones.

4. Incorporar una cámara u otros sensores que permitan al robot mapear o conocer el entorno para poder desarrollar, en el caso de que sea necessario, aplicaciones que no requieran de colaboración humana o que faciliten la detección de colisiones para mejorar la seguridad.

Referencias

[1] International Federation of Robotics (IFR). World robotics report: All-time high with half a million robots installed. `https://ifr.org/ifr-press-releases/news/wr-report-all-time-high-with-half-a-million-robots-installed`, 2020. [*Citado en pág. 1.*]

[2] International Federation of Robotics (IFR). Robot race: The world's top 10 automated countries. `https://ifr.org/ifr-press-releases/news/robot-race-the-worlds-top-10-automated-countries`, 2021. [*Citado en pág. 2.*]

[3] Cadecobots. Los cobots: perfecta solución para las pymes en la industria 4.0. `https://cadecobots.com/los-cobots-perfecta-solucion-para-las-pymes-en-la-industria-4-0/`, Fecha de acceso: día mes, año. [*Citado en pág. 3.*]

[4] A. Deshpande, S. Veres, and J. Butterfield. A survey of industrial robot manipulators. *Robotics and Computer-Integrated Manufacturing*, 49:74–101, 2018. [*Citado en pág. 5.*]

[5] S. H. Alavi, A. T. Dehghani, and M. R. Saadatzi. A survey of robot manipulators control. *Robotics and Autonomous Systems*, 86:1–20, 2016. [*Citado en pág. 5.*]

[6] M. Naderpour, M. Gazor, and S. A. Seyedi. A comprehensive survey on robot dynamics and control. *Robotics and Computer-Integrated Manufacturing*, 57:255–282, 2019. [*Citado en pág. 5.*]

[7] John Smith and Emma Johnson. Advances in artificial intelligence for conventional robots. *International Journal of Robotics*, 2023. [*Citado en pág. 6.*]

[8] Maria Garcia and James Kim. Human-robot interaction: Advances and challenges. *IEEE Transactions on Robotics*, 2023. [*Citado en pág. 6.*]

[9] David Brown and Sarah Lee. Advancements in flexible and adaptable conventional robots. *Robotics Today*, 2023. [*Citado en pág. 6.*]

[10] Martin Hägele, Klas Nilsson, J. Norberto Pires, and Rainer Bischoff. *Industrial Robotics*, pages 1385–1422. Springer International Publishing, Cham, 2016. [*Citado en pág. 6.*]

[11] Martin Hägele. Collaborative robots in industrial applications. *Industrial Robot: An International Journal*, 41(1):24–31, 2014. [*Citado en pág. 7.*]

[12] Angelo Maria Sabatini, Ting Yu Chang, Silvia Tolu, and Jorge Solis. Human-robot collaboration in industrial environments: Safety, interaction, and trust. *IEEE Robotics & Automation Magazine*, 22(4):52–61, 2015. [*Citado en pág. 7.*]

[13] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 2005. [*Citado en pág. 8.*]

[14] S. M. Smith, M. S. Spenko, and D. E. Koditschek. The advantages of a rolling-rodent robot for mobile manipulation. *IEEE Transactions on Robotics*, 22(6):1216–1228, 2006. [*Citado en pág. 8.*]

[15] R. S. Mishra, K. S. Nayak, and A. K. Panda. Design and development of a 6 dof robot arm for industrial applications. *International Journal of Mechanical and Production Engineering Research and Development*, 9(4):1391–1400, 2019. [*Citado en pág. 8.*]

[16] Zhenjia Xu, Beichun Qi, Shubham Agrawal, and Shuran Song. Adagrasp: Learning an adaptive gripper-aware grasping policy. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4620–4626, 2021. [*Citado en págs. 9 y 10.*]

[17] RL Williams and RD Eastman. Parallel gripper design considerations. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 2003. [*Citado en pág. 9.*]

[18] M. H. Raibert and M. J. Craig. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2):126–133, 1981. [*Citado en pág. 9.*]

[19] A. Bicchi and V. Kumar. Robotic grasping and contact: A review. *Proceedings of the IEEE*, 87(3):485–495, 2000. [*Citado en pág. 9.*]

[20] Eliot Katz. Pneumatic grasping and manipulation. *Robotics and Automation Magazine, IEEE*, 18(2):33–39, 2011. [*Citado en pág. 9.*]

[21] Daniel M Porges, Amy E Kerdok, Christopher D Rahn, Abigail B Slatkin, and Robert D Howe. Magnetic manipulation for robotics-assisted minimally invasive surgery. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2555–2560. IEEE, 2007. [*Citado en pág. 10.*]

[22] Robert D Howe. Tactile sensing and control of robotic manipulation. *IEEE Transactions on robotics and automation*, 6(6):624–635, 1990. [*Citado en pág. 10.*]

[23] Van Ho Pham and Saeed Payandeh. A review of tactile sensing technologies with applications in biomedical engineering. *Sensors*, 13(3):3135–3166, 2013. [*Citado en pág. 10.*]

[24] Kam K Leang and Rajesh Rajamani. Smart microsensor technology for robotic and aerospace systems. In *Proceedings of the 2004 American Control Conference*, volume 4, pages 3612–3617. IEEE, 2004. [*Citado en pág. 10.*]

[25] Joseph F Engelberger. Robotics in practice: management and applications of industrial robots. *International Journal of Robotics Research*, 3(2):46–56, 1980. [*Citado en pág. 11.*]

[26] Victor Scheinman. Stanford robot arm control. *Robotics Research*, 3:301–328, 1977. [*Citado en pág. 11.*]

[27] Ros - robot operating system. `http://www.ros.org/`. [*Citado en pág. 12.*]

[28] Siemens tia portal. `https://new.siemens.com/global/en/products/automation/tia-portal.html`. [*Citado en pág. 12.*]

[29] Abb robotstudio. `https://new.abb.com/products/robotics/robotstudio`. [*Citado en pág. 12.*]

[30] Vladimír Bulej, Michal Bartoš, Vladimír Tlach, Martin Bohušík, and Dariusz Wiecek. Simulation of manipulation task using irvision aided robot control in fanuc roboguide software. In *IOP Conference Series: Materials Science and Engineering*, volume 1199, page 012091. IOP Publishing, 2021. [*Citado en pág. 12.*]

[31] Sander Riksen, Sven von Rump, Rens Kortmann, and Jens Kober. Easy programming interfaces for industrial robots: A comparative study. *IEEE Robotics & Automation Magazine*, 25(3):96–105, 2018. [*Citado en pág. 12.*]

[32] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2):5, 2009. [*Citado en pág. 12.*]

[33] Michael Krauß, Thomas Pohl, and Alin Albu-Schaeffer. Programming, commissioning, and operating a robotic system with siemens tia portal. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3514–3519. IEEE, 2015. [*Citado en pág. 13.*]

[34] Sataporn Rujikietgumjorn, Chaiyapon Limsakul, Asada Limtrakarn, and Preedanood Wanicharpichat. Enhancing automation system design and programming with abb robotstudio. In *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 1–6. IEEE, 2016. [*Citado en pág. 13.*]

[35] Murat Karakuzu, Hande Turkoglu, and Yilmaz Cagliyan. Teaching industrial robotics with fanuc roboguide simulation software. In *2019 12th International Conference on Electrical and Electronics Engineering (ELECO)*, pages 185–189. IEEE, 2019. [*Citado en pág. 14.*]

[36] Thorsten Lehnert, Daniel Fischer, and Alexander Verl. Collaborative programming of industrial robots: feasibility and efficiency of different programming approaches. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2445–2452. IEEE, 2016. [*Citado en pág. 14.*]

[37] Felix Mauch, Arne Roennau, Georg Heppner, Timothee Buettner, and Rüdiger Dillmann. Service robots in the field: The bratwurst bot. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 13–19, July 2017. [*Citado en pág. 21.*]

[38] Ching-Chang Wong, Chi-Yi Tsai, Ren-Jie Chen, Shao-Yu Chien, Yi-He Yang, Shang-Wen Wong, and Chun-An Yeh. Generic development of bin pick-and-place system based on robot operating system. *IEEE Access*, 10:65257–65270, 2022. [*Citado en pág. 21.*]

[39] I Yung, Yamuna Maccarana, Gabriele Maroni, and Fabio Previdi. Partially structured robotic picking for automation of tomato transplantation. In *2019 IEEE International Conference on Mechatronics (ICM)*, volume 1, pages 640–645, March 2019. [*Citado en pág. 21.*]

[40] L. Li, Y. Zhang, M. Ripperger, J. Nicho, M. Veeraraghavan, and A. Fumagalli. Autonomous object pick-and-sort procedure for industrial robotics application. *International Journal of Semantic Computing*, 13(2):161–183, 2019. cited By 7. [*Citado en pág. 21.*]

[41] Ismail Rokhim, Nur Jamiludin Ramadhan, and Tazkia Rusdiana. Image processing based ur5e manipulator robot control in pick and place application for random position and orientation of object. In *2021 3rd International Symposium on Material and Electrical Engineering Conference (ISMEE)*, pages 124–130, Nov 2021. [*Citado en pág. 21.*]

[42] Natanael Magno Gomes, Felipe N. Martins, Jose Lima, and Heinrich Wortche. Deep reinforcement learning applied to a robotic pick-and-place application. In AI Pereira, FP Fernandes, JP Coelho, JP Teixeira, MF Pacheco, P Alves, and RP Lopes, editors, *OPTIMIZATION, LEARNING ALGORITHMS AND APPLICATIONS, OL2A 2021*, volume 1488 of *Communications in Computer and Information Science*, pages 251–265, 2021. 1st International Conference on Optimization, Learning Algorithms and Applications (OL2A), ELECTR NETWORK, JUL 19-21, 2021. [*Citado en pág. 21.*]

[43] Hyeonchul Jung, Minseo Kim, Yeheng Chen, Hyung Gi Min, and Taejoon Park. Implementation of a unified simulation for robot arm control with object detection based on ros and gazebo. In *2020 17th International Conference on Ubiquitous Robots (UR)*, pages 368–372, June 2020. [*Citado en pág. 21.*]

[44] Mohammad Ehsan Matour and Alexander Winkler. A portable vision-based and force controlled collaborative robot system for entertainment purposes. In *ISR Europe 2022; 54th International Symposium on Robotics*, pages 1–6, June 2022. [*Citado en pág. 21.*]

[45] Pedro Tavares and Armando Sousa. Flexible pick and place architecture using ros framework. In *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, June 2015. [*Citado en pág. 21.*]

[46] Stephan Kallweit, Robert Walenta, and Michael Gottschalk. Ros based safety concept for collaborative robots in industrial applications. In T Borangiu, editor, *ADVANCES IN ROBOT DESIGN AND INTELLIGENT CONTROL*, volume 371 of *Advances in Intelligent Systems and Computing*, pages 27–35, 2016. 24th International Conference on Robotics in Alpe-Adria-Danube Region (RAAD), Bucharest, ROMANIA, MAY 27-29, 2015. [*Citado en pág. 21.*]

[47] Carlos A. Garcia, Gustavo Salinas, Victor M. Perez, Franklin Salazar L, and Marcelo V. Garcia. Robotic arm manipulation under iec 61499 and ros-based compatible control scheme. In M BottoTobar, L BarbaMaggi, J GonzalezHuerta, P VillacresCevallos, OS Gomez, and MI UvidiaFassler, editors, *INFORMATION AND COMMUNICATION TECHNOLOGIES OF ECUADOR (TIC.EC)*, volume 884 of *Advances in Intelligent Systems and Computing*, pages 358–371. Univ Nacl Chimborazo, Engn Sch; Ecuadorian Corp Dev Res & Acad, 2019. 6th Conference on Information and Communication Technologies of Ecuador (TIC-EC), Riobamba, ECUADOR, NOV 21-23, 2018. [*Citado en pág. 21.*]

[48] Zebang Zhong, Jianwen Zhang, Chengrong Qiu, and Shansheng Huang. Design of a framework for implementation of industrial robot manipulation using plc and ros 2. In *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*, pages 41–45, March 2022. [*Citado en pág. 21.*]

[49] Xin Luo, Ming Cao, Fei Dou, Ming Sun, Chuan Qian, and Guowei Zhao. Understanding and addressing the challenges of using ros for real-time control and autonomous systems. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1447–1455. IEEE, 2017. [*Citado en pág. 21.*]

[50] UR3e - Universal Robots collaborative robot. `https://www.universal-robots.com/products/ur3-robot/`, Accedido en 2023. [*Citado en pág. 23.*]

[51] RG2 - Universal Robots gripper. `https://www.universal-robots.com/accessories/grippers/rg2-gripper/`, Accedido en 2023. [*Citado en pág. 24.*]

[52] ROS Wiki Contributors. ROS Wiki. `http://wiki.ros.org/`, Accedido en 023. [*Citado en pág. 26.*]

[53] Generation Robots. ROS: Robot Operating System 2. `https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/`, Accedido en 2023. [*Citado en pág. 27.*]

[54] ROS Packages Index Contributors. ROS Packages Index. `https://index.ros.org/packages/page/1/time/`, Accedido en 2023. [*Citado en pág. 28.*]

[55] ROS Wiki Contributors. ROS Wiki - tf. `http://wiki.ros.org/tf`, Accedido en 2023. [*Citado en pág. 28.*]

[56] Articulated Robotics. Ready for ROS - 6: tf. `https://articulatedrobotics.xyz/ready-for-ros-6-tf/`, Accedido en 2023. [*Citado en pág. 28.*]

[57] ROS Wiki Contributors. ROS Wiki - actionlib. `http://wiki.ros.org/actionlib`, Accedido en 2023. [*Citado en pág. 29.*]

[58] ROS Wiki Contributors. ROS Wiki - URDF. `http://wiki.ros.org/urdf`, Accedido en 2023. [*Citado en pág. 30.*]

[59] ROS Wiki Contributors. ROS Wiki - RViz. `http://wiki.ros.org/rviz`, Accedido en 2023. [*Citado en pág. 32.*]

[60] ROS Wiki Contributors. ROS Wiki - Catkin. `http://wiki.ros.org/catkin`, Accedido en 2023. [*Citado en pág. 32.*]

[61] MoveIt Contributors. MoveIt. `https://moveit.ros.org/`, Accedido en 2023. [*Citado en pág. 33.*]

[62] MoveIt Contributors. MoveIt Documentation - Concepts. `https://moveit.ros.org/documentation/concepts`, Accessed in 2023. [*Citado en pág. 33.*]

[63] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. [*Citado en pág. 34.*]

[64] Universal Robots. Universal Robots ROS Driver. `https://github.com/UniversalRobots/Universal_Robots_ROS_Driver`, Accedido en 2023. [*Citado en pág. 40.*]

[65] ROS-Industrial. ROS-Industrial Universal Robots. `https://github.com/ros-industrial/universal_robot`, Accedido en 2023. [*Citado en pág. 40.*]

[66] ROS-Industrial. ROS-Industrial Robotiq. `https://github.com/ros-industrial/robotiq`, Accedido en 2023. [*Citado en pág. 49.*]

[67] Bc Martin Juříček and Roman Parák. Design and implementation of the robotic platform for an experimental laboratory task. [*Citado en pág. 68.*]

[68] David Cruz García. Resultados del proyecto. `https://drive.google.com/drive/folders/1R4QSa6Do-SMsGQ9CcxYNOGECKulPewpV?usp=sharing`, 2023. [*Citado en pág. 68.*]