FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

Trabajo de grado para optar por el título de
Ingeniero en Automática Industrial

Plataforma software para la manipulación de objetos 3D para un robot UR

Juan David Ruiz Flórez

Juan Sebastián Montenegro Bravo

Popayán, Junio, 2023

Nota de Aceptación

_____

_____

_____

_____

_____

_____

Director: _____

Óscar Andrés Vivas Albán

_____

Firma del jurado

_____

Firma del jurado

Popayán, Junio de 2023

«¡Cuán grande riqueza es, aun entre los pobres,
el ser hijo de buen padre!»
- Juan Luis Vives -

En memoria de Julio César Montenegro Rojas.

Agradecimientos

Juan Sebastián Montenegro Bravo

Estoy sinceramente agradecido a Dios por permitirme vivir y guiar cada paso de mi vida. Mi gratitud se extiende a mi familia, quienes siempre me han dado la oportunidad de soñar y han brindado un apoyo incondicional, creyendo en mí en todo momento. También quiero honrar la memoria de mi padre, agradeciéndole por su amor y sabiduría que siguen siendo una fuente de inspiración en mi vida.

Quiero reconocer y agradecer a mis compañeros especial a mi compañero de tesis Juan David Ruiz, quienes han sido una parte fundamental de mi formación. Los momentos compartidos, tanto dentro como fuera de la Universidad, han enriquecido nuestra experiencia y nos han brindado un apoyo mutuo invaluable.

Expreso mi profundo agradecimiento a mi director de tesis, el PhD. Óscar Andrés Vivas Albán, quien ha hecho posible la realización de esta investigación. Sus aportes oportunos, disponibilidad y paciencia han enriquecido significativamente el trabajo realizado. También agradezco a mi tutor, el PhD. Jose María Sabater Navarro, por su apoyo y capacidad para orientar nuestras ideas, llevando este estudio hacia resultados exitosos. No puedo dejar de mencionar al grupo de investigación nBio de la Universidad Miguel Hernández de Elche. España, y a mi compañero Juan David Romero quien contribuyó de manera valiosa durante todo este proceso.

Expreso mi gratitud a la Universidad del Cauca y a mis maestros, quienes han brindado su apoyo, tiempo y dedicación, sentando las bases para mi formación como ingeniero.

Finalmente, agradezco a los evaluadores de este proyecto, cuyo conocimiento y críticas constructivas han contribuido a mejorar la calidad y el alcance de esta investigación.

Juan David Ruiz Flórez

En este momento de profunda gratitud, elevo mi voz en glorificación a la grandeza de Dios y le agradezco por las innumerables bendiciones que he recibido. Quiero expresar mi gratitud por las oportunidades que me ha brindado con este proyecto y por su infinita sabiduría que me ha guiado en cada paso de este camino. Reconozco y agradezco su amor y gracia, los cuales me han sostenido en todo momento y han sido la fuente de mi fortaleza para superar obstáculos y alcanzar mis metas que no serían posibles sin su divina guía.

Asimismo, deseo expresar mi más profundo agradecimiento a mi amada familia, quienes han sido un apoyo constante en mi vida. Han estado a mi lado, brindándome amor, comprensión y apoyo incondicional. Incluso en los momentos más desafiantes, han sido mi motor y me han ofrecido los mejores consejos para tomar decisiones acertadas. Quiero hacer especial mención a mi madre, Amanda Patricia Flórez, quien siempre ha visto y creído en mis capacidades, y ha sido mi apoyo en todo momento.

También quiero expresar mi gratitud a mi compañero de tesis, Juan Sebastián Montenegro, quien compartió conmigo este fascinante camino de investigación. Juntos hemos enfrentado retos y superado obstáculos. Su dedicación y trabajo en equipo han sido esenciales para el éxito de este proyecto. Deseo extender mi agradecimiento a mi tutor y director de tesis, el PhD. Oscar Andrés Vivas. Fue gracias a su apoyo y conocimientos expertos que pudimos llevar a cabo esta investigación de manera exitosa. Sus valiosas sugerencias, comentarios y dedicación fueron fundamentales para en este trabajo.

Por otro lado, quiero expresar mi agradecimiento a mi tutor, el PhD. José María Sabater, de la Universidad Miguel Hernández. Su asesoramiento y contribuciones han sido de gran relevancia para este trabajo. Gracias a su experiencia y visión, he podido ampliar mis horizontes académicos y abordar la investigación desde nuevas perspectivas. También deseo agradecer a los miembros del grupo nBio, quienes generosamente compartieron su tiempo, conocimientos y experiencia. Sus valiosas contribuciones fueron invaluables y jugaron un papel significativo en el éxito de esta investigación. Quiero hacer una mención especial a Juan David Romero, cuya ayuda durante la fase de investigación fue de gran valor y me permitió forjar una nueva amistad.

Por último, deseo expresar mi agradecimiento a la Universidad del Cauca por proporcionar un entorno propicio para mi crecimiento académico. Quiero agradecer a todos los profesores que contribuyeron a mi formación y a la calidad de la educación que recibí. Además, quiero extender mi agradecimiento a mis compañeros de la Universidad del Cauca, quienes compartieron conmigo experiencias y momentos inolvidables a lo largo de mi carrera académica.

Resumen

La evolución de la industria y su enfoque actual en la colaboración entre humanos y robots ha llevado a una transformación en el control de los manipuladores. Por lo tanto, numerosos investigadores están dedicando sus esfuerzos al desarrollo de plataformas que simplifiquen la programación de trayectorias en los robots. En este sentido, esta tesis presenta una revisión del estado del arte actual en diferentes campos como el industrial, médico y educativo.

El proyecto detalla exhaustivamente la construcción de una plataforma software basada en ROS (*Robot Operating System*), que permite la manipulación del robot UR3e mediante la implementación de tres modos de control simples. Asimismo, se destaca la creación de una interfaz gráfica utilizando Unity, que permite al usuario interactuar con el sistema. Para visualizar los movimientos del robot, se ha incorporado un gemelo digital del robot en formato URDF. Se explica detalladamente la comunicación establecida entre los diferentes componentes que integran la plataforma con el robot, utilizando los tópicos y servicios de ROS para la suscripción y publicación de información, así como la adición de paquetes necesarios para garantizar dicha comunicación. También se describe la conexión de la plataforma con el robot real y el simulador URSim. Por último, se desarrolla una pinza de interferencia granular específicamente para llevar a cabo pruebas con objetos 3D.

En este estudio, se describen detalladamente las pruebas llevadas a cabo para la manipulación de objetos utilizando la plataforma específicamente desarrollada. Se documentan los resultados obtenidos a través de pruebas rigurosas, encontrando un desplazamiento exitoso de los objetos. Además, se presentan las conclusiones derivadas del desarrollo de esta plataforma, resaltando su potencial para futuras aplicaciones en el campo de la robótica y la investigación.

Palabras clave: ROS, Unity, robot UR3e, manipulación robótica, URSim, robots colaborativos, pinzas blandas, interfaz gráfica, rutas libres.

Abstract

The evolution of industry and its current focus on human-robot collaboration has led to a transformation in the control of manipulators. Therefore, numerous researchers are devoting their efforts to the development of platforms that simplify the programming of robot trajectories. In this sense, the thesis presents a review of the current state of the art in different fields such as industrial, medical and educational.

The project exhaustively details the construction of a software platform based on ROS (Robot Operating System), which allows the manipulation of the UR3e robot through the implementation of three simple control modes. It also highlights the creation of a graphical interface using Unity, which allows the user to interact with the system. To visualize the robot movements, a digital twin of the robot in URDF format has been incorporated. The communication established between the different components that integrate the platform with the robot is explained in detail, using ROS topics and services for the subscription and publication of information, as well as the addition of packages necessary to guarantee such communication. The connection of the platform with the real robot and the URSim simulator is also described. Finally, a granular interference gripper is developed specifically for performing tests with 3D objects.

In this study, the tests carried out for object manipulation using a specifically developed platform are described in detail. The results obtained through rigorous testing are documented, finding a successful displacement of objects. In addition, conclusions derived from the development of this platform are presented, highlighting its potential for future applications in the field of robotics and research.

Keywords: ROS, Unity, UR3e robot, robotic manipulation, URSim, collaborative robots, soft grippers, graphic interface, free path.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

La interacción humano-robot (HRI) es un concepto en auge acogido en el ámbito de la robótica, que consiste en tener una colaboración segura entre el humano y el sistema robótico, con el fin de llevar a cabo cierto trabajo o tarea en conjunto [1]. Actualmente está interacción es posible gracias a dos componentes esenciales, el primero está relacionado con los denominados robots colaborativos (cobots), estos son diseñados específicamente para operar en el mismo espacio de trabajo que los humanos. El otro componente corresponde a la importancia de las interfaces gráficas que facilitan la comunicación directa entre el manipulador y el operador [2].

Estas interfaces son utilizadas para enviar órdenes de movimiento al autómata y recibir una retroalimentación sobre el seguimiento de la consigna deseada, así como del estado del manipulador. Lo que se destaca de estas plataformas de control corresponde a la adaptabilidad que pueden llegar a ofrecer para las distintas aplicaciones en las que se puede usar el robot.

En el caso de la empresa Universal Robots, los robots colaborativos se controlan mediante un software propio llamado *Polyscope* [3] [4] a través de una tableta digital (*Teach Pendant*). Esta herramienta permite programar tareas sencillas principalmente para el ámbito industrial, que se sustenta ante todo en tareas repetitivas. Sin embargo, para otras aplicaciones como la robótica quirúrgica, las tareas a realizar son diferentes y cambiantes en todo momento, como suturas, incisiones o perforaciones óseas. Por lo anterior, si se desea programar trayectorias sofisticadas con un robot UR, se debe de recurrir a programas diferentes a *Polyscope*. Esta programación es posible mediante el uso de diversos software, por ejemplo Rviz, Moveit, Python, ROS o Matlab [5] [6], no obstante el uso de estas herramientas implica una curva de aprendizaje más lenta para el usuario.

Por lo anterior, este trabajo aborda la construcción de una plataforma software que permite programar trayectorias en el plano tridimensional, sin tener grandes conocimientos en programación, sin invertir mucho tiempo, y sin hacer uso del limitado *Polyscope*. Esto aportará a la línea de la robótica médica para que en el futuro se pueda tener una mayor facilidad en la ejecución de trayectorias quirúrgicas complejas. Esta interfaz fue

desarrollada en la plataforma Unity 3D, que en conjunto con el software ROS y Python, establece la comunicación entre el cobot UR3e y su simulador (URSim).

Gracias al convenio entre la Universidad del Cauca (Popayán, Colombia) y la Universidad Miguel Hernández (Elche, España), se realizó una estancia investigativa de 3 meses en las instalaciones del laboratorio del grupo nBio, donde se llevaron a cabo pruebas de funcionamiento de la plataforma creada con el robot real.

La estructura de esta monografía se divide en seis capítulos donde se detallan las temáticas del proyecto realizado. El capítulo uno (ya abordado) presenta la introducción para contextualizar al lector. El capítulo dos muestra la revisión del estado del arte actual, seguido por el capítulo tres que contiene los conceptos generales sobre los temas que se tratan en el documento, el desarrollo general de la plataforma y la respectiva implementación. Luego en el capítulo 5 se documentan los resultados obtenidos de las pruebas realizadas y finalmente el capítulo 6 presenta las conclusiones, así como los trabajos futuros.

2. Estado del arte

En el caso del control de manipuladores robóticos, es común encontrar que estos incluyen su software propio, pero en ocasiones dicho software presenta limitaciones, pues los desarrolladores enfocan sus plataformas en aspectos básicos o particulares [7]. Además, el lenguaje y forma de programación de cada robot es muy distinta, por ello muchos investigadores se ven forzados a recurrir al uso de otras herramientas para la creación de plataformas más robustas o enfocadas directamente al campo de interés, dependiendo de dónde desean utilizar el robot.

2.1. Campo industrial

La evolución de la industria y su enfoque actual en la colaboración entre humanos y robots ha llevado a una transformación en el control de los manipuladores. Por lo tanto, numerosos investigadores están dedicando sus esfuerzos al desarrollo de plataformas que simplifiquen la programación de trayectorias en los robots, con el objetivo de mejorar el rendimiento en los procesos de producción.

En el estudio [8], se presenta una interfaz basada en el movimiento por imitación para el control de robots industriales. Este software se ha implementado en un robot Scara de Mitsubishi que está equipado con un sensor Kinect de Microsoft. Dicho sensor tiene la capacidad de identificar los gestos corporales de una persona, los cuales se codifican en Matlab para así poder mover el manipulador. Sin embargo, la interfaz presenta ciertas dificultades en la detección del cuerpo en lo que respecta al rango de seguimiento del Kinect. En distancias muy cercanas o muy lejanas, el campo de visión de la cámara comienza a fallar. Para abordar este problema, se ha implementado una solución consistente en mantener el cuerpo a una distancia intermedia del sensor.

En el documento [9] se destaca la importancia de las interfaces hombre-robot en el diseño de sistemas robóticos seguros y eficientes, especialmente en escenarios peligrosos y no planificados que requieren el uso de múltiples robots. Se describe el desarrollo de una interfaz hombre-robot modular y multimodal para la intervención con un equipo cooperativo de robots en el complejo de aceleradores del CERN (Figura 2.1). Esta

interfaz permite el control homogéneo de robots heterogéneos, brindando funcionalidades de secuencias de comandos en vivo que se adaptan en tiempo real. Se enfatiza en enfoques de desarrollo de software para garantizar la modularidad y seguridad del sistema. Los resultados muestran un alto nivel de usabilidad, capacidad de aprendizaje y seguridad, incluso para operadores no expertos. La interfaz ha sido utilizada con éxito en intervenciones reales en entornos industriales, demostrando su precisión y seguridad.

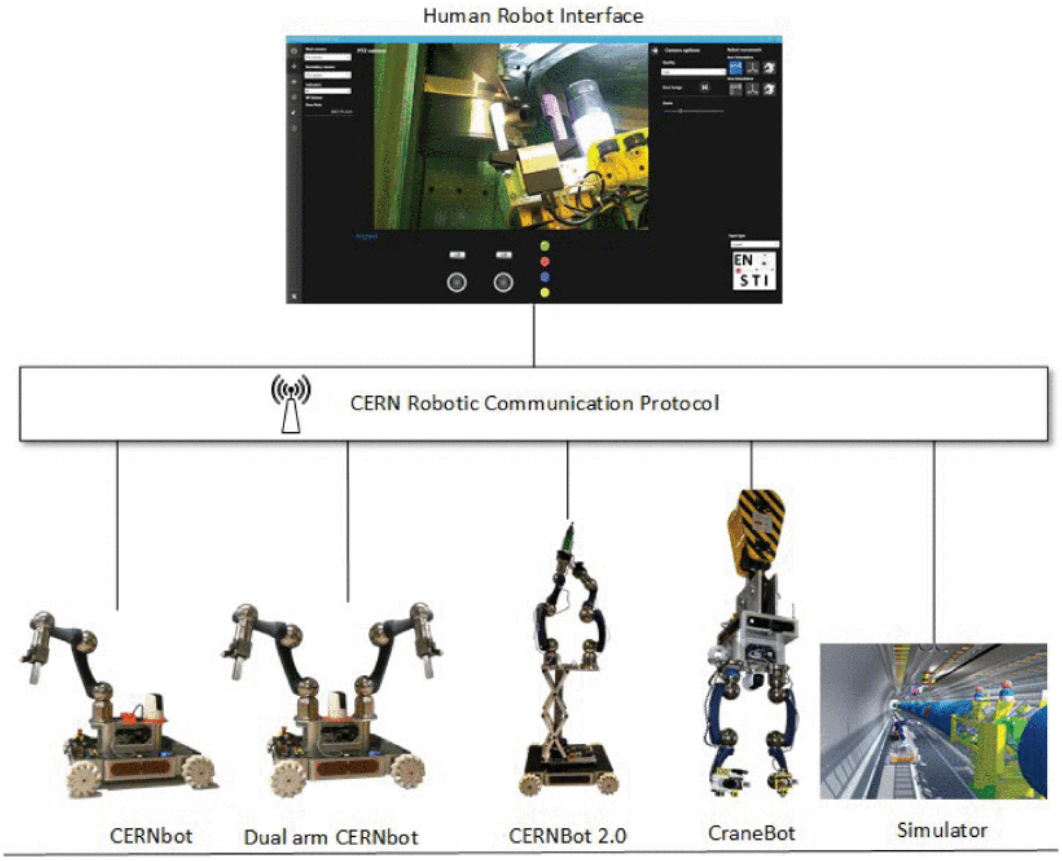
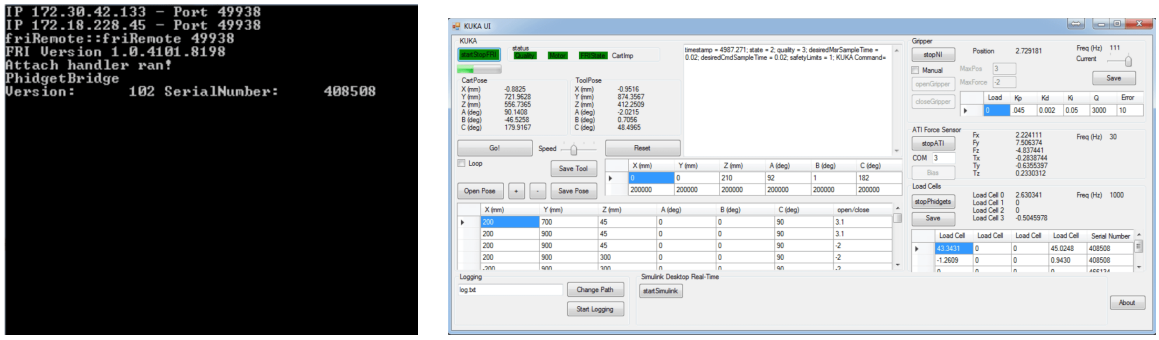


Figura 2.1.: Interfaz unificada multimodal humano-robot para interactuar con el equipo heterogéneo de robots [9].

Por otro lado, en el estudio [10], se presenta una plataforma capaz de enviar comandos en tiempo real para controlar un robot Kuka mediante una conexión a Internet, utilizando el protocolo UDP. La necesidad de desarrollar esta interfaz surge del hecho de que el control del robot Kuka se realiza mediante comandos desde una consola (Figura 2.2(a)), lo cual a veces dificulta su manipulación. Con el propósito de simplificar las tareas complejas requeridas para manejar este robot y reducir los tiempos de latencia en la comunicación, se ha creado esta plataforma. La programación del robot se lleva a cabo

utilizando el lenguaje KRL y el control de trayectorias se realiza mediante el toolbox KTC de Matlab. Además, esta plataforma permite la integración de dispositivos externos, como sensores y actuadores. El entorno de la interfaz está basado en el framework .NET, ya que se argumenta que proporciona aplicaciones sólidas y confiables. En la Figura 2.2(b), se muestra la interfaz que permite el envío de posiciones y orientaciones al robot, así como la opción de enviar consignas predefinidas en el plano cartesiano. Los resultados obtenidos demuestran una interfaz de usuario intuitiva y, a través de tres casos de estudio, se demuestra la posibilidad de manipular el robot en tiempo real utilizando la plataforma propuesta.



(a) Consola de programación (b) Interfaz de control

Figura 2.2.: Control del robot Kuka por consola y por interfaz [10].

Con el fin de enriquecer la colaboración entre humanos y robots, se ha explorado el concepto de teleoperación como un valor añadido. Un ejemplo destacado es el trabajo realizado por [11], que ha desarrollado una interfaz que permite la teleoperación mediante un espacio compartido. Esta propuesta presenta una interfaz que crea un entorno virtual del manipulador UR5 (Figura 2.1) y permite el control a través de una computadora y un mando de juego. Además, se aborda la solución del marco de referencia adaptable a la orientación relativa del usuario, lo cual resulta especialmente útil para entornos dinámicos y cambiantes.

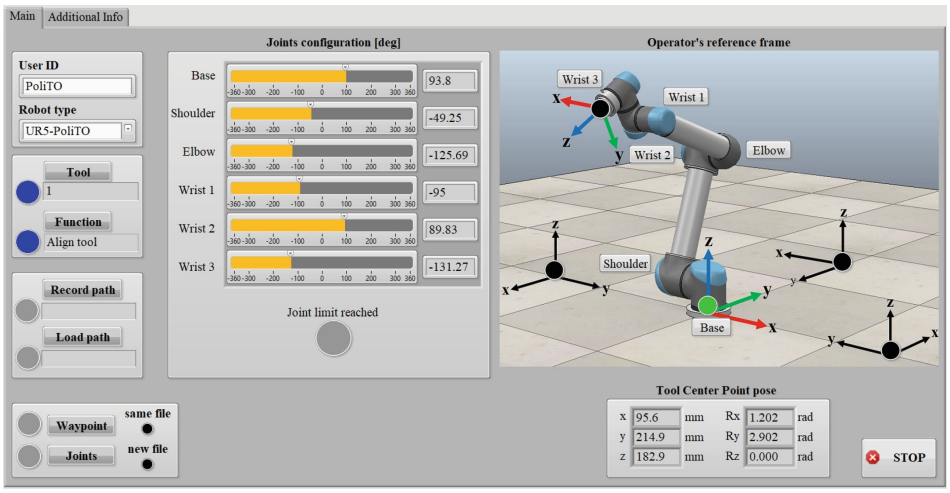


Figura 2.3.: Interfaz gráfica usada para el telecontrol del robot colaborativo UR5 [11].

2.2. Campo médico

En el ámbito médico, se observa el avance de software alternativo que busca crear sistemas robóticos para cirugías que sean completamente autónomos [12]. Varios de estos estudios presentan plataformas simples para tareas quirúrgicas básicas, mientras que otros son más complejos y se centran en un tipo particular de tratamiento u operación.

El artículo [13] se centra en desarrollar un conjunto de soluciones tanto de hardware como de software para llevar a cabo acciones quirúrgicas simples. La propuesta incluye el uso de un robot comercial UR, un robot diseñado específicamente para cirugías y una serie de algoritmos para la planificación y ejecución de las tareas. Todo esto se gestiona a través de una interfaz gráfica de usuario (GUI) diseñada a partir de encuestas realizadas a profesionales de la salud, con el objetivo de obtener una interfaz eficiente y centrada en el usuario. Además, el sistema cuenta con un modo de teleoperación para permitir la manipulación directa por parte del cirujano. Las pruebas se llevaron a cabo en un maniquí destinado a operaciones, donde se evaluaron la inserción de una aguja y la crioablación de un tumor renal. Sin embargo, aún hay mucho por explorar en este proyecto para mejorar su robustez, como la inclusión de otras tareas como cortes o suturas.

En el artículo [14], se propone una plataforma de robótica junto con un algoritmo de planificación de rutas para la bioimpresión in situ. Esta plataforma se construye utilizando Linux CNC, un software de código abierto, y se emplea la herramienta Matlab para la planificación de rutas, como se muestra en la Figura 2.4. Permite realizar

impresiones en superficies o piezas irregulares previamente diseñadas. El algoritmo calcula ángulos para mantener la perpendicularidad del efector final en cada punto de bioimpresión. Aunque se presentan imprecisiones en los ejes de dirección horizontal y vertical, los resultados obtenidos son aceptables. Sin embargo, el enfoque se dirige hacia aplicaciones médicas, específicamente en la restauración de piel, cartílago y huesos, por lo que es esencial mejorar la precisión de la plataforma.

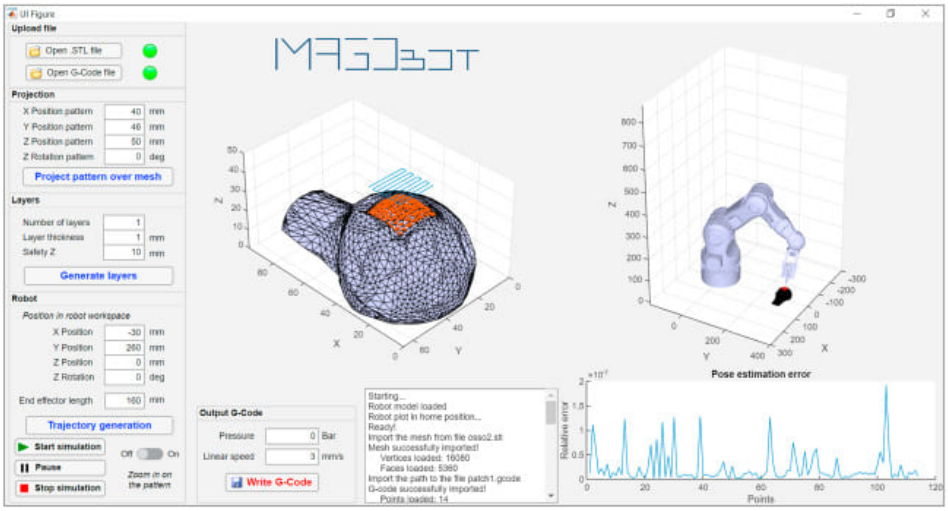


Figura 2.4.: Interfaz para la gestión y simulación de las rutas de impresión [14].

En [15] se introduce una plataforma quirúrgica llamada MOPS, la cual se destaca por su aprovechamiento de la comunicación ofrecida por ROS. Esta característica posibilita la creación de módulos y la integración de diversos tipos de hardware, como dos manipuladores UR, dos dispositivos hápticos y cámaras Basler AG, que se conectan entre sí en la plataforma para permitir la realización de teleoperaciones quirúrgicas. El objetivo principal de este proyecto consiste en desarrollar una plataforma abierta y adaptable a componentes comerciales para crear un sistema quirúrgico completo que pueda ser utilizado por los cirujanos. Sin embargo, es importante tener en cuenta que la modularidad de la plataforma presenta limitaciones que dependen del hardware utilizado. En este caso particular, se ha observado que al emplear dos robots de la serie UR pero de distintos modelos (UR5-D y UR5-e), uno de ellos muestra una menor precisión en comparación con el otro. Asimismo, se han detectado fallos en la detección de los componentes del sistema quirúrgico, la cual se basa en el uso de marcadores.

Este estudio de caso [16] describe una configuración basada en ROS (*Robot Operating System*) para la cirugía asistida por robots en el ámbito de la cirugía mínimamente invasiva. El sistema incluye componentes de percepción como sensores Kinect, cámaras

de tiempo de vuelo, cámaras endoscópicas, rastreadores basados en marcadores y ultrasonido, así como dispositivos de entrada háptica de fuerza, robots como KUKA LWR y Universal Robots UR5, soporte de endoscopio ViKY, instrumentos quirúrgicos y pantallas de realidad aumentada. Se han desarrollado subsistemas basados en combinaciones de estos componentes, como un telemanipulador bimanual, seguimiento de múltiples personas con Kinect, guía de endoscopio basada en el conocimiento y tomografía por ultrasonido. Esta configuración no es un proyecto de investigación en sí mismo, sino una infraestructura básica utilizada para varios proyectos de investigación en robótica. El objetivo principal es mostrar cómo construir una plataforma robótica completa y flexible utilizando ROS (Figura 2.5), ejecutándose en la versión de ROS Indigo y el sistema operativo Ubuntu Trusty (14.04).

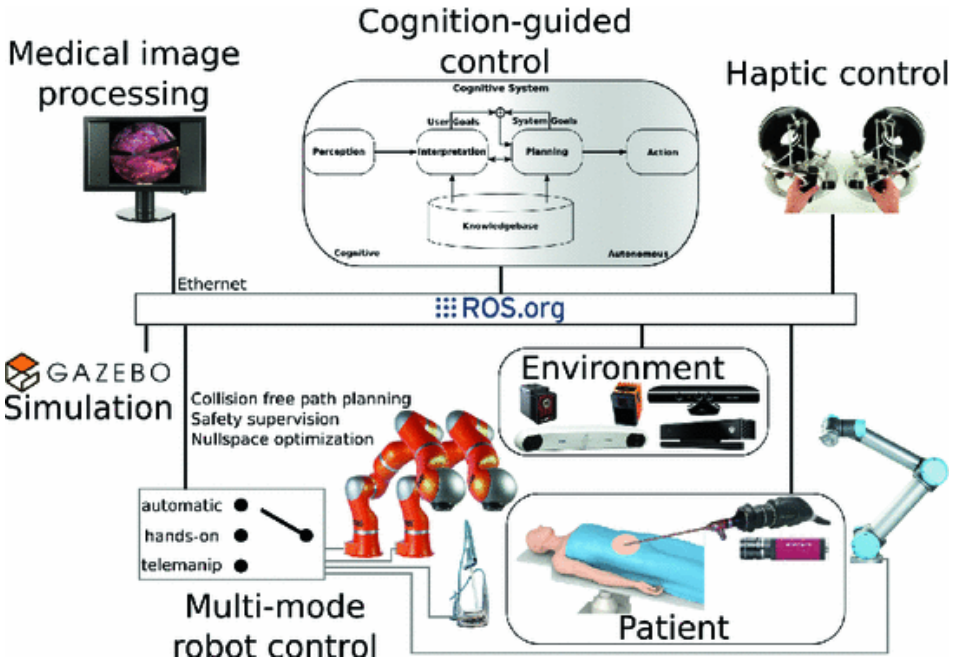


Figura 2.5.: Plataforma modular basada en ROS para cirugía mínimamente invasiva asistida por robot [16].

En el ámbito de la cirugía, también se llevan a cabo investigaciones enfocadas en procedimientos específicos, como se evidencia en el caso de la plataforma Sylvius [17]. Esta plataforma multidisciplinaria está diseñada para abordar la cirugía de epilepsia. Una de sus funciones destacadas es la capacidad de generar un modelo tridimensional del cerebro a partir de imágenes de tomografía computarizada. Este modelo tridimensional se utiliza para planificar las rutas que el robot debe seguir al colocar electrodos profundos. Siguiendo una línea similar, se encuentra el estudio Tactics [18], el cual surge en respuesta

a las limitaciones observadas en las aplicaciones comerciales para la cirugía estereotáctica. El objetivo de Tactics es desarrollar una plataforma que permita la planificación y validación de las rutas de inserción de electrodos cerebrales, utilizando las imágenes de tomografía computarizada del paciente. Como resultado de esta investigación, se lograron planificar trayectorias efectivas en cuestión de minutos para numerosos casos. No obstante, se identifican algunas limitaciones significativas, como la deficiencia en el flujo de trabajo y la falta de comparación con otras plataformas similares. En el primer caso, la plataforma solo permite llevar a cabo dos procedimientos clínicos simultáneamente, y en el segundo caso, se requiere un mayor análisis comparativo para evaluar su verdadero potencial.

2.3. Campo educativo

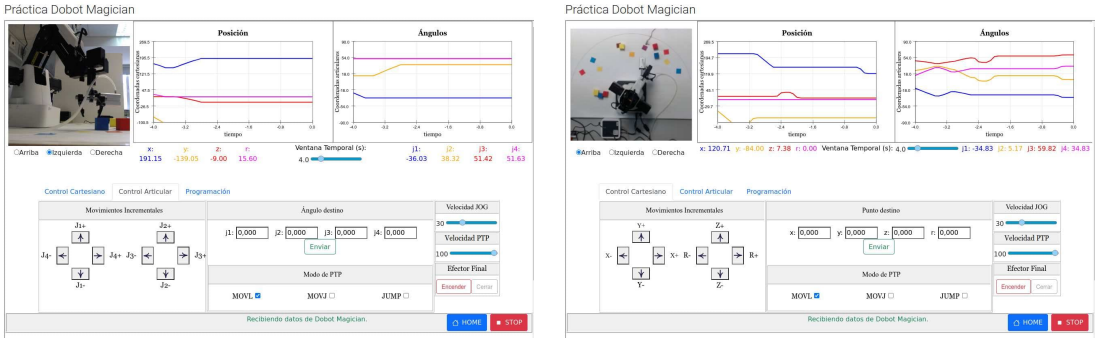
En el ámbito educativo, se ha empleado el uso de plataformas robóticas con el propósito de despertar interés y motivación [19], así como facilitar la transmisión de conocimientos de forma más accesible.

En el estudio realizado [20], se investigó la efectividad de enseñar robótica a estudiantes de ingeniería industrial utilizando herramientas de simulación como RobotScene y SGRobot. Estos programas permiten la construcción y programación de robots a través de una interfaz visual, donde los parámetros cinemáticos pueden ser configurados y los errores se reflejan en tiempo real en la pantalla. Además, ofrecen simulaciones de trayectorias con características diversas para crear tareas y escenarios virtuales. Esta metodología fomenta el desarrollo de habilidades y destrezas en los estudiantes. Los resultados estadísticos obtenidos en este estudio demostraron que el uso de estas herramientas resulta efectivo en el entorno educativo. No solo refuerzan los conocimientos teóricos adquiridos en clase, sino que también proporcionan una experiencia dinámica que motiva a los estudiantes a seguir explorando el campo de la robótica.

En los últimos años, la realidad virtual ha experimentado un notable crecimiento, pero gran parte de su aplicación no está directamente relacionada con sistemas robóticos. No obstante [21] destaca por su contribución al desarrollar un entorno inmersivo a través de una interfaz de realidad virtual en Unity3D que se conecta a un robot industrial. Esta innovadora solución permite la simulación, capacitación y control de robots, y se distingue por su bajo costo y los resultados positivos que ha mostrado en el aumento de la eficiencia de los procesos de capacitación y simulación. La integración de la realidad virtual y los sistemas robóticos ofrece ventajas significativas, ya que proporciona una experiencia inmersiva realista para interactuar con un entorno virtual y facilita la

práctica y perfeccionamiento de habilidades sin riesgos. Esta aplicación concreta ha demostrado su capacidad para reducir los tiempos de aprendizaje, minimizar errores y optimizar el rendimiento en tareas específicas, posicionándose como una herramienta prometedora en la mejora de la formación y el control de robots en diversos sectores industriales.

En un enfoque similar, se destaca el trabajo presentado en [22], el cual aborda el desarrollo de un laboratorio remoto con objetivos educativos. Este laboratorio consiste en un software en línea que se conecta a través de servidores para proporcionar acceso al laboratorio de la Universidad Complutense de Madrid, permitiendo a los usuarios interactuar con el robot educativo Dobot Magician. El software cuenta con una interfaz web que permite controlar el robot tanto en modos articulares como cartesianos. Además, se incluye una visualización del robot y gráficas que representan el estado del manipulador (Figura 2.6). Esta solución, de bajo costo y con fines educativos, permite a los estudiantes conectarse de forma remota y realizar prácticas que promueven su aprendizaje. Como resultados se llevaron a cabo pruebas con un grupo de 5 docentes y 5 estudiantes, quienes respondieron positivamente a la implementación del software. Sin embargo, se destacó el problema relacionado con el ángulo de visión en la interfaz como una limitación a considerar.



(a) Interfaz web en el modo de control articular (b) Interfaz web en el modo de control cartesiano

Figura 2.6.: Interfaz web del laboratorio remoto [22].

3. Materiales y métodos

3.1. Descripción del hardware

En esta sección, se proporcionará una descripción detallada del hardware utilizado en el proyecto, que incluye los cobots y el robot UR3e. Estos componentes desempeñan un papel crucial en la implementación de tareas de manipulación y automatización.

3.1.1. Cobot

La Organización Internacional para la Estandarización ha definido en la ISO 10218-2 punto 3.2, a los robots colaborativos como aquellos que han sido diseñados para realizar un apoyo conjunto con la parte humana dentro de un área de trabajo establecida.

Los robots colaborativos o también llamados "cobots" por su abreviatura [23], se caracterizan por 4 aspectos fundamentales [24].

- No trabajan a altas velocidades y su carga útil es comparable con el brazo de una persona.
- Se utilizan en tareas cooperativas, es decir que no reemplazan el trabajo humano.
- Son simples para controlar y programar.
- Son seguros y pueden compartir espacios de trabajo con las personas.

Una de las empresas líderes en la fabricación de robots colaborativos es Universal Robots, que lanzó su primer modelo, el UR5, en 2008. Debido a su éxito, la compañía desarrolló otras versiones como el UR3, UR10 y UR16 [25]. Estos robots tienen una estructura similar a la de un brazo humano y cuentan con una pantalla digital, conocida como *Teach Pendant*, que permite al usuario interactuar y controlar el robot de manera sencilla.

3.1.2. Robot UR3e

El brazo robótico de tipo colaborativo utilizado en este proyecto es el *UR3e*. Destaca por ser el robot de menor tamaño de la línea del fabricante, lo que lo hace ideal para entornos de trabajo reducidos. Cuenta con un rango de operación de hasta 50 cm desde su base, una carga útil de 3 kg, un peso total de 11 kg y una precisión de 0.1 mm. Posee seis grados de libertad, donde todas las articulaciones de la muñeca giran 360°. A diferencia de su homólogo *UR3-CB*, el *UR3e* ofrece una mayor frecuencia de intercambio de datos, alcanzando los 500 hz [26] [27].

El robot cuenta con una tableta de enseñanza *Teah Pendant*, que incluye una interfaz de visualización gráfica 3D llamada *Polyscope*. Esta interfaz permite a un usuario no experimentado ver y controlar el comportamiento del robot. Aunque esta tableta tiene limitaciones para la generación de trayectorias complejas, es posible incorporar diversas aplicaciones como Unity, ROS, Coppelia, entre otras, lo que posibilita la codificación de algoritmos propios para manipular el robot y realizar tareas con alto grado de operabilidad.

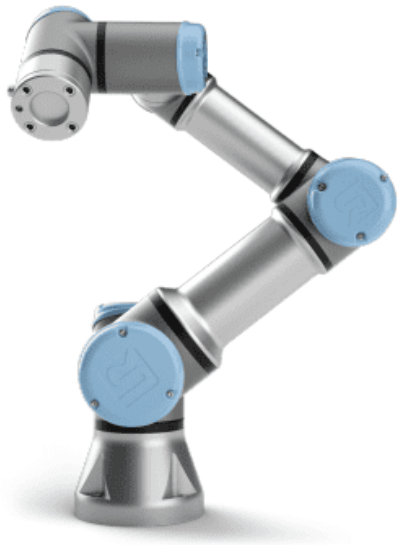


Figura 3.1.: Robot *UR3e* [28].

Figura 3.2.: *Teach Pendant* [28].

3.2. Descripción del software

En esta sección, se menciona el uso de varias herramientas y *frameworks* de software altamente especializados para lograr un desarrollo eficiente del proyecto. Entre los principales componentes utilizados se encuentran Unity, una poderosa plataforma de desarrollo de juegos y simulación, ROS (*Robot Operating System*), un sistema flexible y ampliamente utilizado en la robótica, Websocket, un protocolo de comunicación bidireccional, Rosbridge, una biblioteca que facilita la integración de ROS con otros sistemas, RosSharp, una biblioteca de comunicación para ROS en C#, URSim, un simulador de los robots de Universal Robots, y ExternalControl, una interfaz de control externo. La combinación de estas tecnologías ha permitido crear una solución completa para el proyecto en cuestión.

3.2.1. Unity

Unity es un motor de desarrollo inicialmente diseñado para crear videojuegos en dos y tres dimensiones, realidad aumentada y/o realidad virtual. Los proyectos desarrollados en Unity son compatibles con diferentes dispositivos (Web, *smartphones*, consolas, ordenadores, etc.) y diversas plataformas (Windows, Linux, Android, iOS, etc.), lo que permite la creación de un solo proyecto multiplataforma.

Aunque su función principal es la creación de videojuegos, esta herramienta también se utiliza en otros campos, como el cine, la medicina y el sector automotriz, para crear animaciones, escenarios virtuales o simuladores. Esto es posible gracias a sus múltiples

funciones integradas, como los motores gráficos y físicos que simulan las leyes físicas, los efectos de sonido e iluminación, y la posibilidad de programación en C# [29].

En la Figura 3.3 se pueden observar los componentes básicos necesarios para trabajar en el editor de Unity. El primer componente es la ventana de jerarquía, la cual enumera todos los objetos (visibles o no) que conforman el mundo que se observa en la ventana central. Esta ventana ofrece dos modos de vista: el modo escena, que es editable y permite llevar a cabo el desarrollo del proyecto, y el modo juego, que muestra el resultado final o la simulación del proyecto. En la parte derecha de la pantalla se encuentra la ventana del inspector, que muestra todas las características y componentes asociados a los objetos creados. Por último, en la ventana del proyecto se muestran todos los recursos generados en el proyecto, tales como escenas, *scripts*, *prefabs*, entre otros.

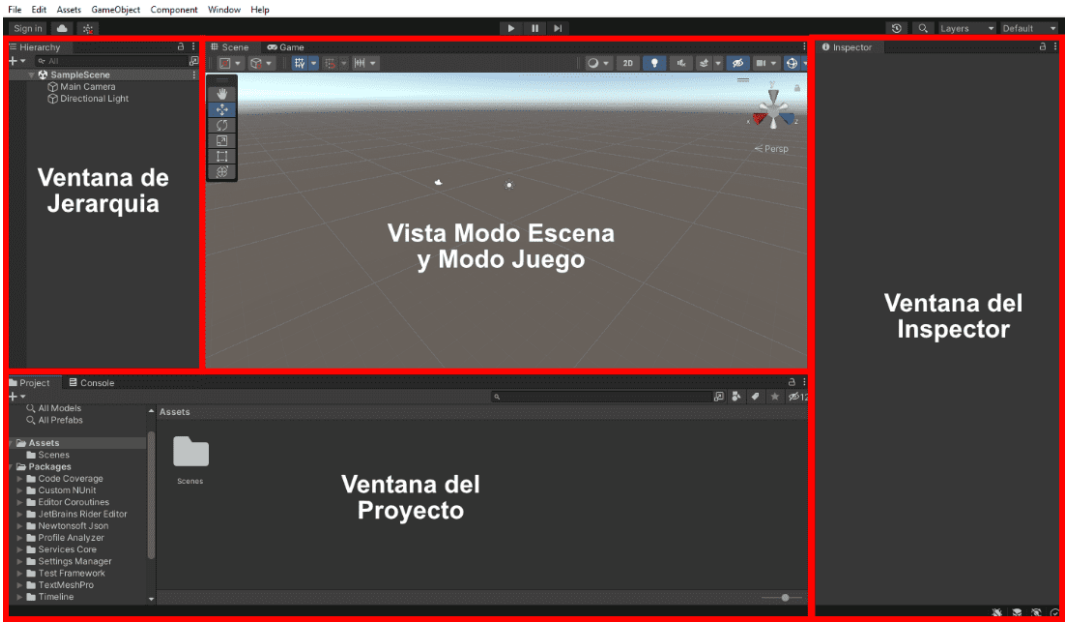


Figura 3.3.: Componentes del editor de Unity.

3.2.2. Robot Operating System (ROS)

ROS es un *middleware* y marco de software especializado en el desarrollo de aplicaciones para robots. Proporciona una plataforma completa compuesta por herramientas, bibliotecas y convenciones que facilitan el desarrollo de software robótico. Su principal función consiste en establecer estándares para el manejo de señales eléctricas y procedimientos en sistemas robóticos, lo que permite la creación de autómatas personalizados con funciones predefinidas reutilizables [30].

Una de las características distintivas de ROS es su arquitectura de comunicación basada en canales conocidos como "tópicos". Estos tópicos permiten la suscripción y publicación de información específica de cada tipo de mensaje, lo que facilita la interoperabilidad entre los distintos componentes de un sistema robótico. Los nodos en ROS son los procesos individuales que se comunican a través de estos tópicos, intercambiando mensajes y datos de manera eficiente (Figura 3.4).

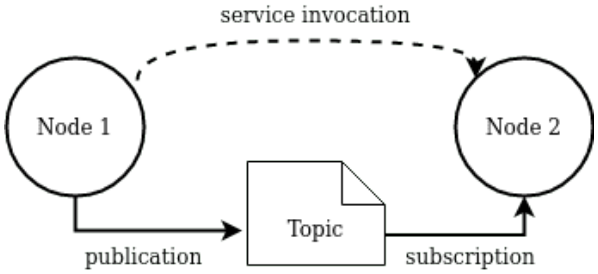


Figura 3.4.: Diagrama de la representación básica de ROS [31].

Además de su arquitectura de comunicación, ROS ofrece una serie de servicios que se asemejan a los ofrecidos por un sistema operativo. Estos servicios incluyen la abstracción del hardware, el control de dispositivos de bajo nivel, la funcionalidad compartida, la comunicación de mensajes entre procesos y la gestión de paquetes. Estas características proporcionan una base sólida para el desarrollo de sistemas robóticos complejos y permiten la integración de diferentes subsistemas, como controladores, sensores y actuadores.

Una ventaja significativa de ROS es su capacidad para trabajar de manera distribuida en varios ordenadores. Esto significa que los desarrolladores pueden dividir el sistema en nodos individuales y ejecutarlos en diferentes computadoras, lo que permite trabajar de forma seccionada con distintos subsistemas. Posteriormente, estos nodos pueden interconectarse de manera eficiente, aprovechando las herramientas y bibliotecas proporcionadas por ROS. Esta capacidad distribuida es especialmente útil para aplicaciones robóticas que requieren un alto grado de escalabilidad y flexibilidad [32] [33].

Dentro de la terminología de ROS se pueden encontrar los siguientes conceptos:

- Nodo: un archivo ejecutable dentro de un paquete de ROS que realiza funciones concretas como leer datos de un sensor o controlar un actuador.
- Tópico: canal de comunicación entre nodos para enviar y recibir un tipo de mensaje en particular.
- Mensaje: estructura de datos que es publicado entre nodos por medio de tópicos.

- Paquete: contenedor de archivos específicos de ROS que incluye nodos.
- Distribución ROS: conjunto de paquetes con cierta versión, dichas distribuciones permiten a los desarrolladores trabajar hasta llegar a una nueva versión estable, por ejemplo, el sistema operativo de Linux y distribución Ubuntu.
- Catkin_ws: espacio de trabajo que contiene paquetes, configuraciones, simulaciones, nodos, servicios entre otros, y es creado para algún proyecto en particular.
- Directorio: ruta donde está ubicada alguna carpeta que contiene archivos.

Mensajes en ROS

Los mensajes son estructuras de datos que permiten la comunicación entre diferentes nodos del sistema. Estos mensajes pueden contener diversos tipos de información, como números, texto, imágenes o información de posición, entre otros. Los nodos de ROS se comunican entre sí mediante la publicación y suscripción a diferentes temas (*topics*), y también pueden enviar mensajes a través de servicios que esperan una respuesta antes de continuar con la ejecución del programa. Los mensajes de ROS están definidos en archivos ".msg"que especifican la estructura y los tipos de datos que contiene el mensaje. Dichos archivos se compilan para generar código fuente en diferentes lenguajes de programación que permiten enviar y recibir mensajes entre nodos de ROS.

En ROS, existen varios paquetes de mensajes utilizados en diferentes situaciones y aplicaciones. Algunos de estos paquetes incluyen *std_msgs*, que contiene los tipos de mensajes más básicos, como cadenas de texto y números; *trajectory_msgs* describe trayectorias en el espacio de juntas o en el espacio de tareas de un robot; *geometry_msgs*, que describe geometrías en diferentes formatos; *sensor_msgs*, que describe información de sensores como imágenes y nubes de puntos; *nav_msgs*, utilizado en la navegación de robots para obtener información de posición y mapa, y *tf2_msgs*, que se utiliza para definir la posición y orientación de los robots y sus componentes. También está *actionlib_msgs*, que se utiliza en la ejecución de acciones, como la retroalimentación y resultados de esas acciones. La elección del paquete de mensajes dependerá de la aplicación específica y los datos que se necesiten comunicar entre los nodos del sistema.

WebSocket

Es un tipo de tecnología sobre un *socket* TCP que permite el flujo constante de datos bidirecional de alta velocidad definida por la frecuencia, usualmente se establece conexión entre dos dispositivos cliente/servidor por medio de una dirección IP local y remota [26].

Rosbridge

El flujo de mensajes en ROS sigue un protocolo basado en *sockets* TCP/IP que asegura un transporte eficiente de datos entre nodos asincrónicos de ROS. Aunque es posible crear comunicaciones directas, esto requiere un conocimiento más profundo de la infraestructura de ROS, especialmente en interfaces de teleoperación, y puede haber problemas relacionados con los retardos y el manejo de los *frames* de Unity. Para evitar estos problemas, se puede utilizar el *framework* de Rosbridge.

Rosbridge permite la comunicación directa entre programas externos, como Unity, y los nodos de ROS. Funciona convirtiendo los mensajes estándar de ROS en mensajes JSON y viceversa, enviándolos a través de una interfaz web. Además, Rosbridge utiliza WebSockets para establecer una capa de comunicación que permite a los agentes externos acceder al entorno de ROS localmente (Figura 3.5) .

La comunicación entre ROS y Unity se realiza mediante la transmisión de mensajes codificados en cadenas YAML a través de WebSockets. Estos mensajes se traducen en objetos instanciados en el entorno de Unity utilizando nodos JSON. La comunicación es bidireccional, lo que significa que Rosbridge recibe mensajes de control desde Unity. Estos mensajes contienen comandos de movimiento y señales para controlar la ejecución de la misión [34] [35].

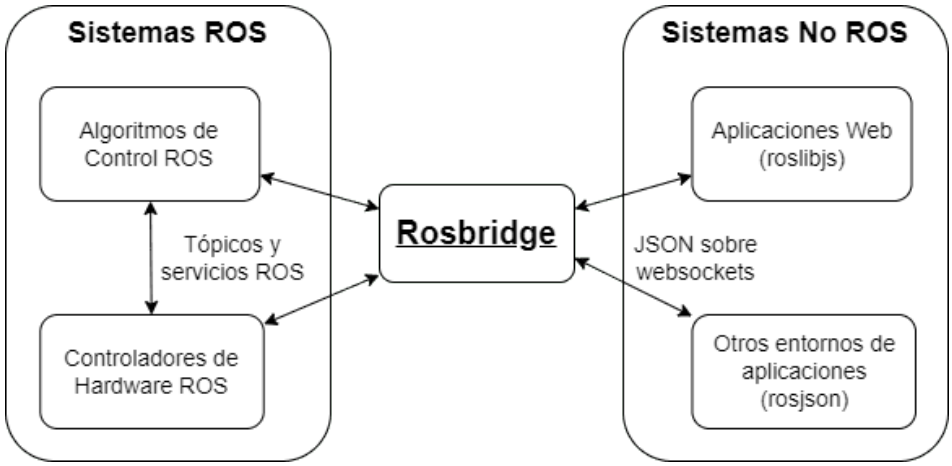


Figura 3.5.: Implementación de Rosbridge.

RosSharp

RosSharp o ROS#, es un conjunto de librerías y bibliotecas de código abierto diseñado para implementar protocolos de comunicación en lenguaje C#. Esto facilita la

comunicación entre los servicios ROS y las aplicaciones desarrolladas en .NET, como Unity [36].

Además de esto, ROS# contiene ficheros en C# predefinidos para distintos tipos de mensajes ROS, lo que permite reutilizarlos y codificarlos de forma estandarizada. Cabe mencionar que el paquete RosSharp requiere la ayuda de Rosbridge para establecer la comunicación entre ROS y Unity. La (Figura 3.6) ilustra cómo se utiliza RosSharp para la comunicación entre ROS y Unity.

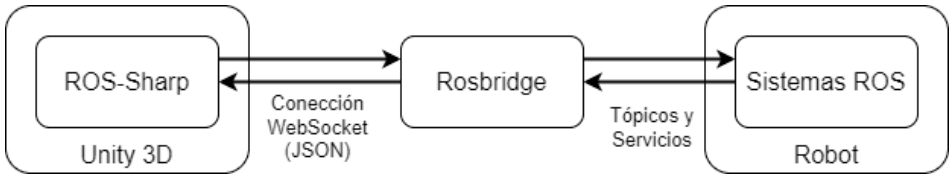


Figura 3.6.: Flujo de datos entre ROS y Unity.

Por otro lado, este paquete también posibilita la importación de sistemas robóticos en formato URDF, lo que resulta muy útil en el desarrollo de simulaciones de robots complejos.

3.2.3. URSim

Una herramienta útil para la programación *offline* de robots UR es URSim (Figura 3.7), un software o gemelo digital de simulación proporcionado por Universal Robots. Este programa, que funciona en el sistema operativo Linux, tiene la capacidad de representar gran parte del comportamiento del robot UR, aunque presenta limitaciones en el cálculo de fuerzas y colisiones. Se pueden simular robots UR tanto de la serie CB como de la serie-e. URSim es especialmente útil para probar trayectorias antes de enviarlas al robot real, lo que permite evitar movimientos indeseados que podrían provocar colisiones y dañar los sistemas internos del manipulador [37] [38].

En la literatura se menciona que URSim es una herramienta útil. No obstante, muchos investigadores prefieren trabajar con una máquina virtual debido a posibles problemas de compatibilidad entre versiones o la falta de conocimiento necesario para llevar a cabo la tarea directamente en el sistema operativo. En el Anexo A.2 se presenta una forma detallada de instalación que facilita el uso directo de URSim en el sistema operativo, sin usar máquina virtual. Al hacerlo, se mejora el consumo de recursos y se evitan posibles problemas de latencia en el envío y recepción de datos.

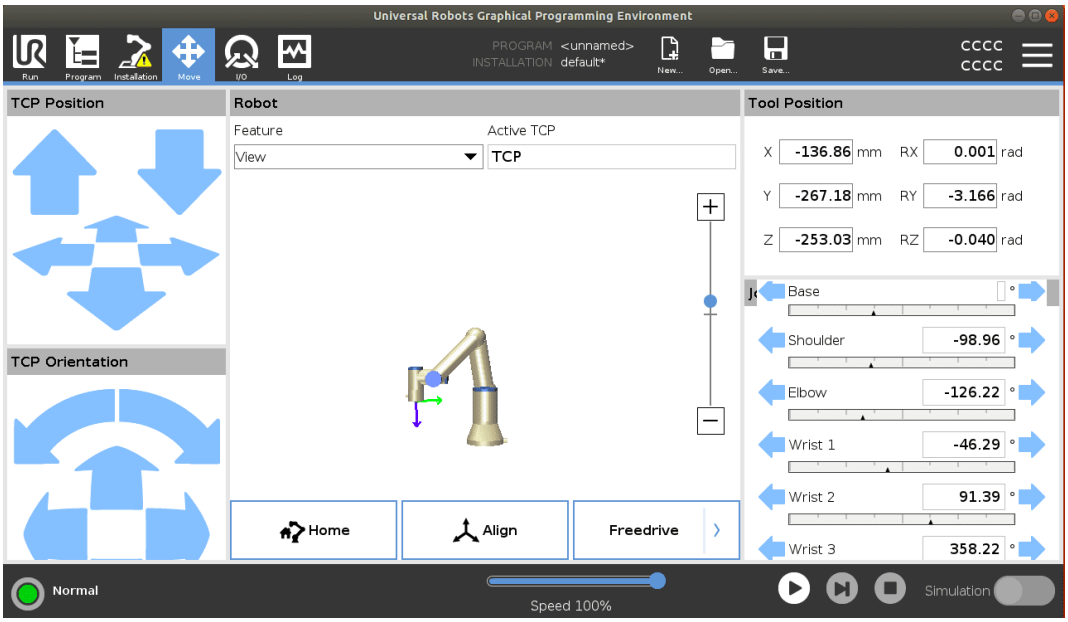


Figura 3.7.: Simulador URSim.

URCap External Control

External Control es una extensión de software URCaps que se incorpora en *Polyscope*, tanto en el simulador URSim como en el robot real. Esta herramienta permite la comunicación entre robots UR y otros dispositivos externos [39]. Con External Control, el robot físico puede conectarse a una red Ethernet y establecer una comunicación con el *socket* del equipo externo encargado de controlar el robot UR. Los manipuladores de Universal Robots interpretan programas desarrollados en lenguaje propio, URScript. Para evitar las limitaciones de *Polyscope*, existen paquetes modulares denominados URCaps que permiten el uso de lenguajes de alto nivel para proporcionar comportamientos complejos a los sistemas robóticos y construir interfaces amigables para el usuario final [40].

3.3. Desarrollo de la plataforma

En esta sección se describe el proceso de construcción de la plataforma que permite la manipulación de objetos con un robot UR3e. Se presenta la creación del escenario virtual en Unity, que incluye un gemelo digital del robot y páneles con los modos de operación para su control. Además, se detallan las herramientas empleadas para establecer la comunicación mediante el paso de información por tópicos de ROS.

3.3.1. Composición del espacio de trabajo (workspace)

Este directorio es el lugar donde se alojan todos los paquetes, configuraciones, nodos y servicios del sistema robótico utilizado. Debido a que es un espacio de trabajo muy sensible a fallos, se recomienda trabajar de manera ordenada y limpia para evitar errores en los archivos que podrían impedir la compilación y ejecución de los programas que ahí están contenidos.

Como estándar de ROS, este directorio debe crearse con el nombre de *catkin_ws*, por ejemplo: *home/UR3e/catkin_ws*. Es importante tener en cuenta que cada robot que se vaya a utilizar debe tener su propio directorio *catkin_ws*, y no se deben combinar paquetes de otros sistemas en un mismo directorio, ya que esto podría generar conflictos entre archivos.

Este directorio se crea a través de una terminal de comandos en el sistema de Ubuntu utilizando la siguiente sintaxis 3.1:

Código 3.1: Composición catkin ws.

```
#crea un directorio
$ mkdir nombre_proyecto
$ cd /UR3e mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
# compila el directorio
$ catkin_make
```

Los comandos anteriores generan el directorio del proyecto *UR3e*, que alberga el espacio de trabajo *catkin_ws* y los subdirectorios llamados *built*, *debel* y *src*, siendo este último el que incorpora todos los paquetes y ficheros necesarios para poner en marcha el robot.

En este espacio de trabajo también se incluyen los paquetes *ur_robot_driver* [41] y *ros_industrial* [42], clonados de sus respectivos repositorios. Ambos paquetes son indispensables para el correcto funcionamiento del robot y su implementación en proyectos de robótica industrial.

3.3.2. Modelo digital del robot UR3e

El propósito de agregar el gemelo digital *UR3e* a Unity es permitir una representación precisa de los movimientos del manipulador real en un entorno virtual, que puede ser visualizado desde una pantalla de computadora. Para lograr este objetivo, se utiliza la herramienta RosSharp y el paqute de *ros_industrial*.

Integración de ROS Sharp

Para comenzar el proceso de integración de RosSharp en Unity, es necesario descargar el paquete RosSharp.unitypackage desde el repositorio de Github [36] como se muestra en la Figura 3.8. Después de la descarga, se debe arrastrar el archivo al proyecto creado en Unity y situarlo en la sección de *Assest*. Este paso mostrará un cuadro de diálogo (Figura 3.9) en el que se deben seleccionar los componentes necesarios para trabajar con ROS. Estos componentes incluyen los tipos de mensajes, servicios y la herramienta *URDF* para importar los modelos digitales.

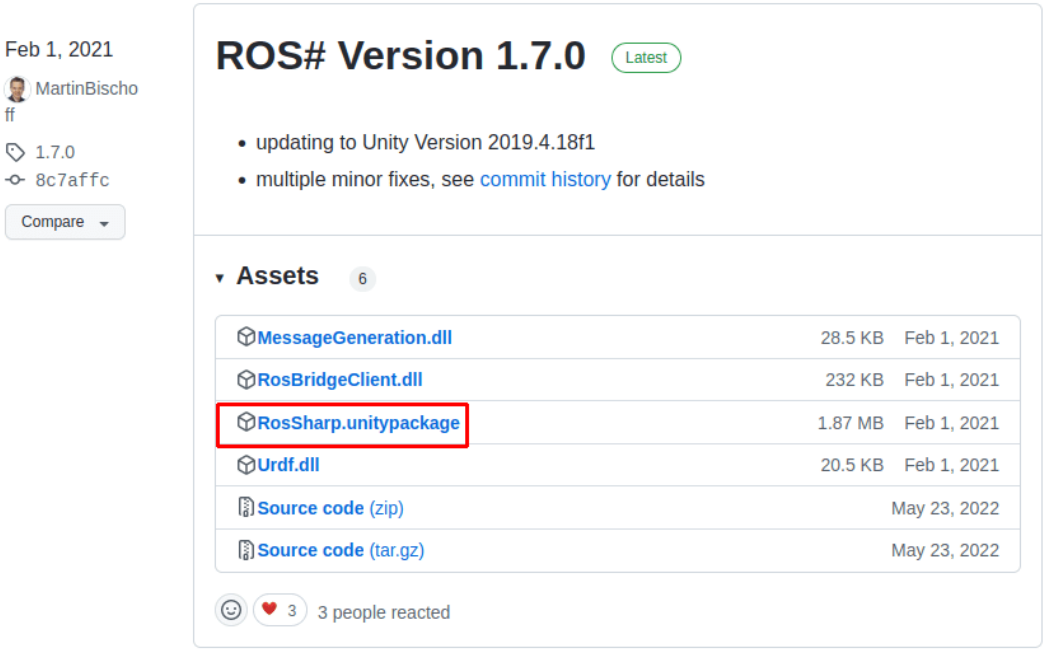


Figura 3.8.: Repositorio de RosSharp.

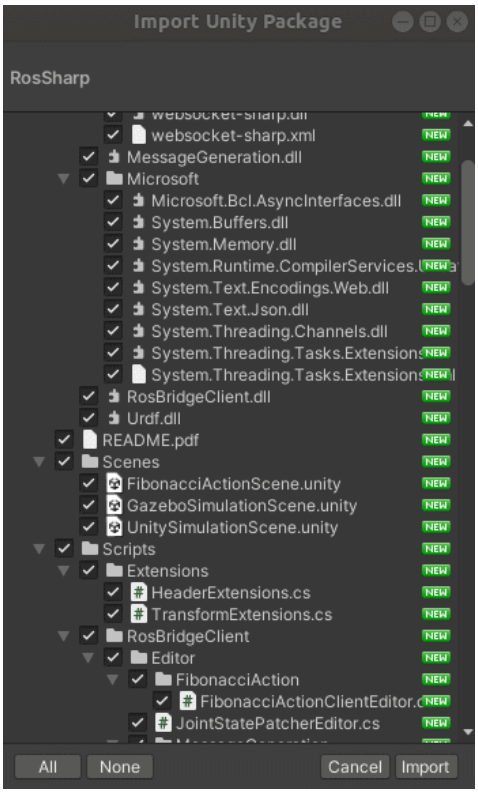


Figura 3.9.: Incorporación del paquete de RosSharp en Unity 3D.

Una vez que se ha completado la instalación de RosSharp en Unity, se verifica si la herramienta ha sido correctamente instalada. Para esto, se debe asegurar que en la escena de Unity se muestra la pestaña *RosBridgeClient* en la parte superior de la pantalla (Figura 3.10). La presencia de esta pestaña indica que la instalación ha sido exitosa y que RosSharp se encuentra correctamente integrado en Unity.

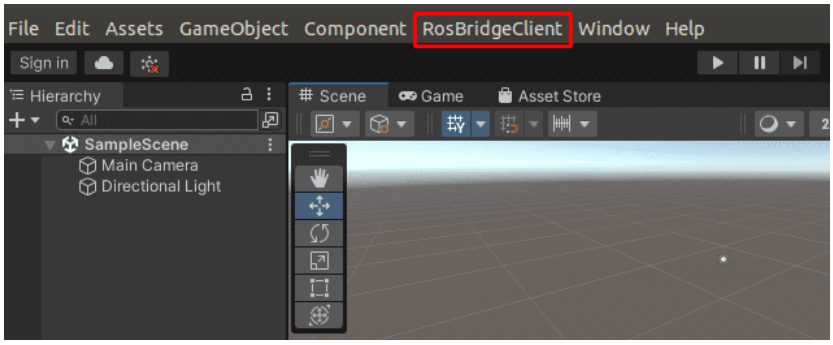


Figura 3.10.: Importación del paquete de RosSharp para Unity 3D.

ROS Industrial

ROS-Industrial alberga interfaces diseñadas para el manejo de manipuladores industriales comunes, pinzas, sensores y redes de dispositivos. En el repositorio de la comunidad Universal Robots en GitHub [42], se encuentra la descripción del modelo de sus robots en formato .xarco (XML Macros). Este formato permite generar los archivos URDF (*Unified Robot Description Format*) que describen el robot. La función del xacro es simplificar documentos XML largos permitiendo la reutilización de elementos comunes para describir el robot y, posteriormente, reutilizarlos.

El formato URDF es especialmente útil para simulaciones, ya que describe la cinemática, la geometría y otros aspectos importantes de los sistemas robóticos, permitiendo la compatibilidad con ROS [43]. Es por esto que se toma el modelo del robot en formato URDF desde el repositorio mencionado anteriormente para ser incorporado en Unity. Para lograrlo, se descomprime o se convierte el archivo que describe al manipulador *UR3e*. Para ello, debe seguirse la siguiente instrucción por terminal (Código 3.2) ubicar la ruta **/home/ur/UR3e/catkin_ws/src/universal_robot/ur_description/urdf** y proceder con la conversión de formatos como se indica en la (Figura 3.11) .

Código 3.2: Conversión URDF.

```
$ cd /UR3e/catkin_ws/src/universal_robot/ur_description/urdf
# Cambio de .xacro a .urdf
$ rosrun xacro xacro ur3e.xacro > ur3e.urdf
```

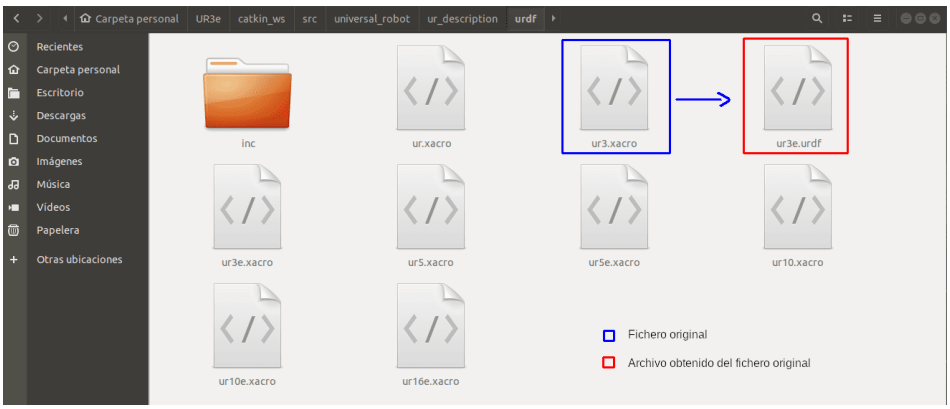


Figura 3.11.: Conversión a la extensión URDF.

Tanto el fichero obtenido como el directorio *ur_description* son copiados a Unity en el apartado *Assest* del proyecto.

Incorporación del robot en Unity

Después de seleccionar el archivo, como se muestra en la Figura 3.12, se puede hacer clic derecho sobre él para encontrar la opción *Import Robot from URDF*. Al seleccionar esta opción, el robot digital aparecerá tanto en la ventana de jerarquía como en la escena en sí. Al desplegar el objeto de la ventana de jerarquía, se pueden observar las diferentes articulaciones del robot, las cuales tienen asociadas una serie de componentes que aparecen en la ventana del inspector. Estos componentes describen algunas características y limitaciones físicas de cada articulación.

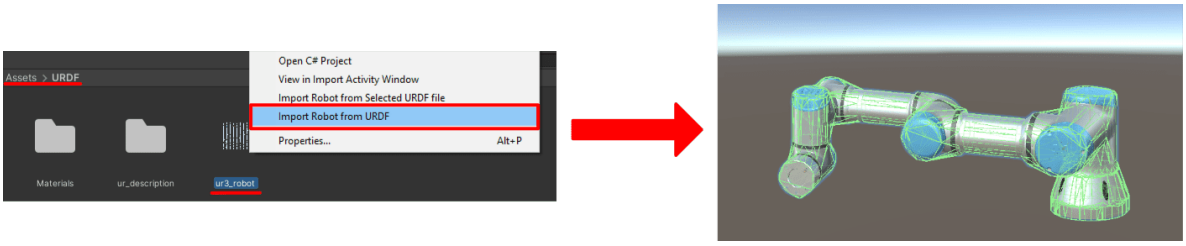


Figura 3.12.: Importación del robot a la escena de Unity.

Dentro de los componentes más fundamentales se tiene: el *rigidbody*, que describe las características físicas de la articulación; el *Urdf Joint Revolute*, que especifica el tipo de articulación, para el caso del robot *UR3e* son articulaciones rotoides; y, finalmente, el componente *Hinge Joint*, que especifica las características cinemáticas, incluyendo el elemento al que está conectada la articulación y el eje de rotación (Figura 3.13).

Adicionalmente, es necesario agregar un archivo llamado "*JointStateWrite*", el cual es proporcionado por RosSharp. Este archivo permite que la articulación pueda ser controlada posteriormente mediante un mensaje de ROS. Es importante destacar que este archivo debe ser agregado a cada una de las articulaciones del robot para que pueda moverse adecuadamente.

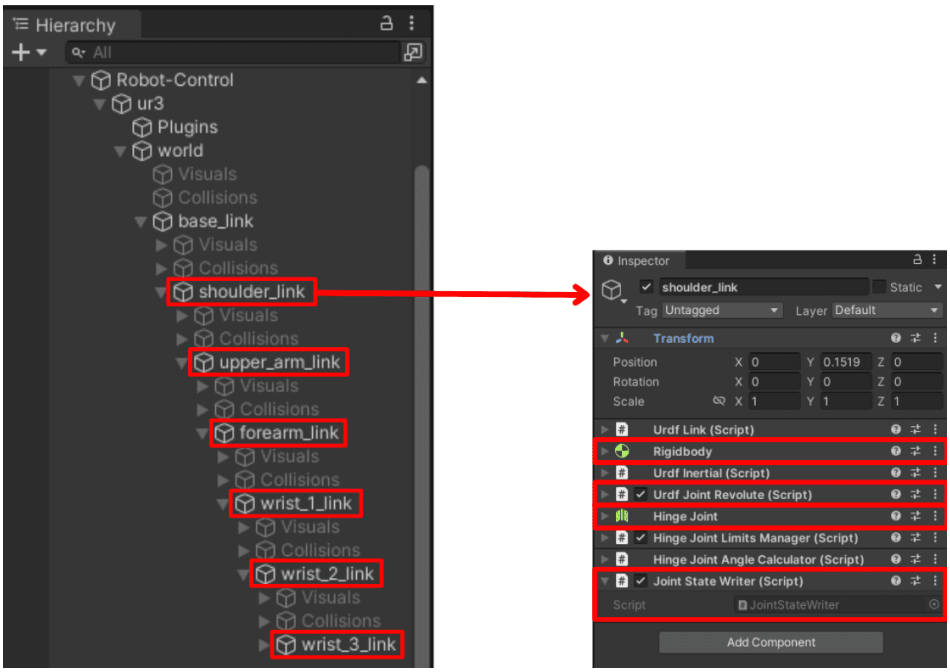


Figura 3.13.: Componentes del robot URDF.

3.3.3. Diseño de la interfaz

En este apartado se aborda la construcción del escenario e interfaz, es decir, los elementos que el usuario podrá visualizar e interactuar. Esto incluye la organización de los paneles, la creación del laboratorio virtual y la configuración de la cámara, incluyendo sus atajos.

Paneles

El panel o tablero es la parte de la interfaz gráfica que permite al usuario interactuar con la plataforma, mediante elementos que brindan información o realizan alguna acción. Para crear la interfaz de usuario en Unity es preciso agregar un lienzo o canvas que contenga los paneles. En este proyecto se ha desarrollado un panel principal, que se divide visualmente en dos sub-paneles. La sección superior del tablero muestra los botones que permiten activar la visualización de los diferentes paneles disponibles, mientras que la sección inferior está destinada a mostrar los paneles de cada modo de control una vez que han sido activados (Figura 3.14).

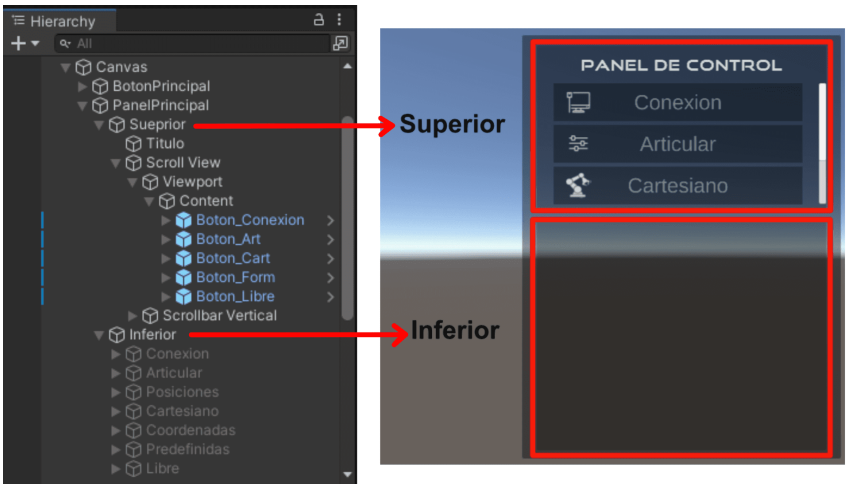


Figura 3.14.: Creación de los tableros para la interfaz.

Escenario

En la etapa de construcción del escenario, se importan algunos objetos adicionales que cumplen la función de proporcionar un ambiente a la escena para hacerla más agradable para el usuario final. Estos objetos están definidos en el formato FBX, el cual es un archivo que contiene datos tridimensionales, sobre la animación, la física, las texturas y otros datos útiles para la creación del contenido 3D.

En la jerarquía del proyecto se crea una entidad contenedora denominada "Laboratorio", que alberga objetos visuales que han sido descargados o diseñados tales como la mesa, la caja controladora del robot [44] y los logos. Además, se construye una habitación utilizando planos de Unity (Figura 3.15). La inclusión de estos detalles a la escena logra dar una mayor inmersión en el entorno virtual, lo que resulta en una experiencia más satisfactoria para el usuario.

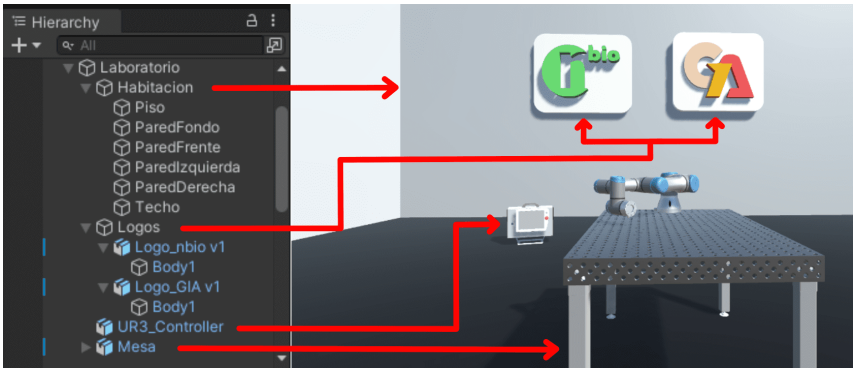


Figura 3.15.: Creación del escenario.

Observador

Otra característica importante de la plataforma es la capacidad que tiene el usuario de moverse a través del laboratorio virtual. Para lograr esto, es necesario configurar los perfiles de entrada desde el menú "*Edit*"→ "*Project Settings*"→ "*Input Manager*"(Figura 3.16). Aquí se definen las acciones que se ejecutarán como respuesta ante las teclas de entrada.

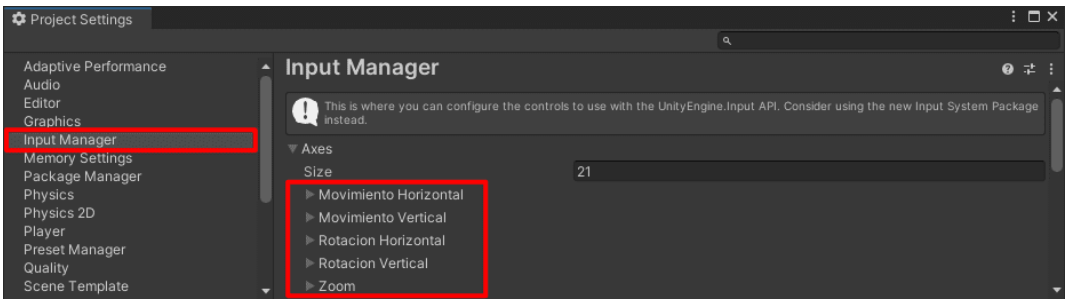


Figura 3.16.: Ventada del *Input Manager*.

Una vez configurados los perfiles de entrada, se crea una entidad llamada "Observador"que incluye la cámara y un *script* (Figura 3.17), que permite al usuario controlar tanto la posición como la rotación de la misma utilizando las teclas previamente configuradas (Anexo A.1.2). También permite alternar entre diferentes vistas que han sido creadas, tales como la vista frontal, trasera, lateral y superior.

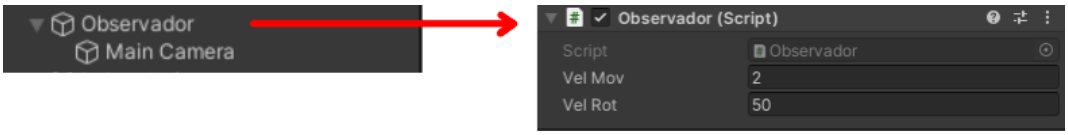


Figura 3.17.: Entidad observador.

Adicionalmente se realiza una modificación en la cámara que permite observar los objetos cercanos y evitar problemas de visualización en la escena. Para lograr esto, se utiliza la propiedad "*Clipping Planes*"del componente "*camera*", que permite establecer la distancia mínima y máxima de visualización. En este caso, se debe configurar el valor de "*Near*" en 0.01 (Figura 3.18).

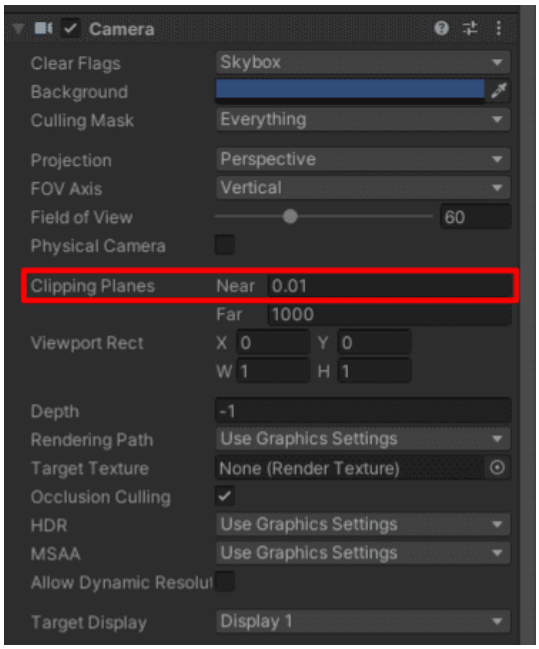


Figura 3.18.: Configuración de la cámara.

3.3.4. Control articular

Se ha diseñado un modo de control que permite mover el robot variando la posición de sus articulaciones en grados. El panel de control articular cuenta con deslizadores que van desde -360 a 360 unidades, representando los grados de cada articulación. Cada deslizador tiene la función de rotar una articulación específica del robot, desplazándola a la posición indicada en grados (RPY). Para lograr esto, se ha agregado a cada deslizador el componente "*Pointer up*", que es un evento que se activa cuando se suelta el clic izquierdo del ratón (Figura 3.19). En este caso, este evento activa la función que envía los valores articulares al robot.

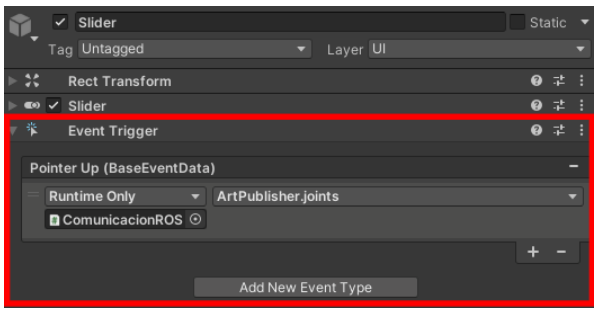


Figura 3.19.: Componente *Pointer Up*.

Para lograr mayor precisión en el movimiento articular, se han incluido botones a los extremos de los deslizadores que permiten incrementar el movimiento en pasos de 0,1 grados. Igualmente, a la derecha de cada deslizador, se han ubicado casillas de texto que indican el valor articular actual en grados. También se han incluido textos en la parte superior de cada deslizador que indican el valor futuro al que se desea que el robot se mueva, lo que facilita la tarea del usuario al tener una referencia visual clara de la posición objetivo (Figura 3.20).

Este modo de control cuenta con una opción para enlistar varias posiciones articulares en un panel ubicado en la parte inferior izquierda de la ventana. Al agregar una posición a la lista, los valores correspondientes son asignados a un arreglo de cuadros de texto construidos como un *prefab*, que es instanciado en el *scrollview* del sub-panel para su visualización.

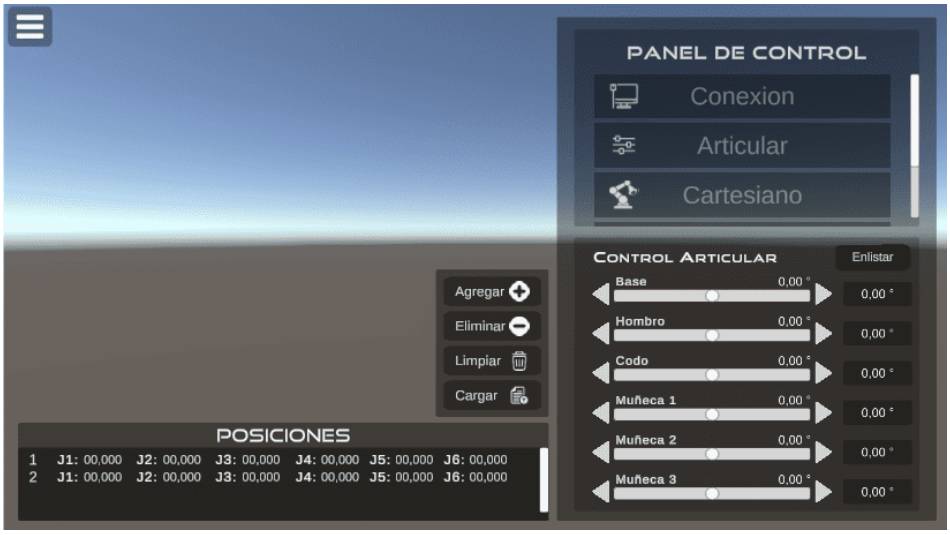


Figura 3.20.: Diseño del control articular.

Por último, es importante destacar que este modo está enlazado con el archivo suscriptor "*JoinStateSuscriber*", que permite mostrar los valores articulares actuales del robot. Así mismo cuenta con un archivo publicador que envía la posición articular a un tópico en ROS en el momento en que se activa por el movimiento de un deslizador o al cargar la lista de posiciones articulares (mayores detalles en la sección 3.3.7).

La lógica de programación de este modo de control está descrita por los diagramas del Anexo A.7.

3.3.5. Control cartesiano

El modo de control cartesiano es una opción para desplazar el robot a una posición tridimensional específica, definiendo la ubicación y orientación del efector final. Para esto, se utiliza un tablero con seis casillas de texto que permiten ingresar los valores de posición y rotación de los tres ejes XYZ. Al lado derecho de cada casilla, se encuentra un cuadro de texto que muestra la posición y orientación actual del efector final. Cabe destacar que las unidades de posición se miden en centímetros y las de orientación en grados RPY. Además, este modo de control cuenta con un deslizador para graduar la velocidad del robot en centímetros por segundo (Figura 3.21).

Figura 3.21.: Diseño del control cartesiano.

Otra característica es la posibilidad de crear una lista de múltiples coordenadas. Al registrar una coordenada, los valores numéricos de las casillas de posición y orientación del panel derecho son copiadas a un arreglo de cuadros de texto predefinido, también conocido como "*prefab*", que se instancia y se muestra en el panel inferior izquierdo.

Por último, es relevante mencionar que este modo de control incluye un archivo suscriptor y un archivo publicador de ROS. El suscriptor obtiene la posición y orientación actual del efector final del robot real, mientras que el publicador envía las posiciones y orientaciones deseadas escritas en el panel o en la lista de coordenadas (mayores detalles en la sección 3.3.7).

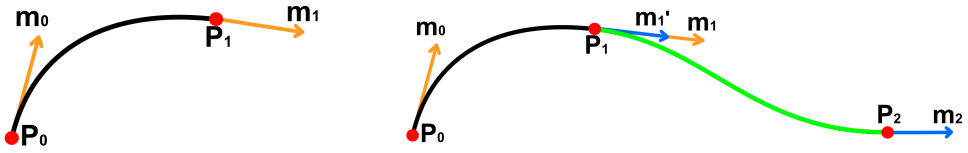
La lógica de programación de este modo de control está descrita por los diagramas del Anexo A.7.

3.3.6. Control por trayectorias libres

El objetivo de este modo de control es permitir que el usuario pueda trazar libremente en el escenario virtual la ruta que seguirá el robot. Para lograr esta idea, se necesitan aplicar ciertas lógicas y utilizar componentes que se describirán a continuación.

Curvas de Hermite

Una curva de Hermite es una representación suave y continua de una curva en el espacio, la cual se construye a partir de dos puntos (P_0 y P_1) y dos vectores tangentes (m_0 y m_1). La línea inicia en el punto P_0 con dirección de la tangente m_0, pero cambia de curso en dirección a la tangente m_1 y termina en el punto P_1 (Figura 3.22(a)). Además, se puede formar una *spline cúbica de Hermite*, conectando varias curvas de Hermite a través de la unión del vector tangente final de una con el vector tangente inicial de la siguiente curva (Figura 3.22(b)). A diferencia de la *spline de Bezier*, las ?*splines de Hermite* pasan por los puntos de control (P_n) [45]. En este proyecto se utiliza la *spline de Hermite* para formar trayectorias adaptativas y permitir la modificación de la ruta del manipulador a través del movimiento de los puntos de control.



(a) Curva cúbica de Hermite (b) Spline cúbica de Hermite

Figura 3.22.: Curva y spline de Hermite.

Line renderer

Es un componente propio de Unity que crea una línea recta continua en el espacio tridimensional a partir de una serie de puntos o coordenadas, que se pueden desplazar por medio de código o de forma manual con el ratón del computador, y que definen la ruta de esta. El componente dispone de parámetros que pueden ser modificados como el color, grosor, textura, entre otros [46]. Es ideal para trazar trayectorias que pueden ir desde una línea recta hasta una geometría más compleja como una espiral.

Para implementar el control por trayectoria adaptativa se ha diseñado un tablero llamado “trayectoria libre”, el cual contiene un botón que activa una función para la creación de una línea recta entre dos esferas situadas en el escenario de la plataforma

(Figura 3.23). Dicha línea es generada a partir del componente "*Line Renderer*", y se le agrega un colisionador de malla para que sea interactiva y pueda detectar el puntero del ratón. Así mismo, los puntos de control (*prefab* de esfera) son interactivos y pueden ser desplazados por la escena para modificar la posición y dirección de la trayectoria. Al presionar sobre algún segmento de la línea, se crea un nuevo punto de control y la línea se transforma en una *spline cúbica de Hermite* definida por la ecuación (3.1).

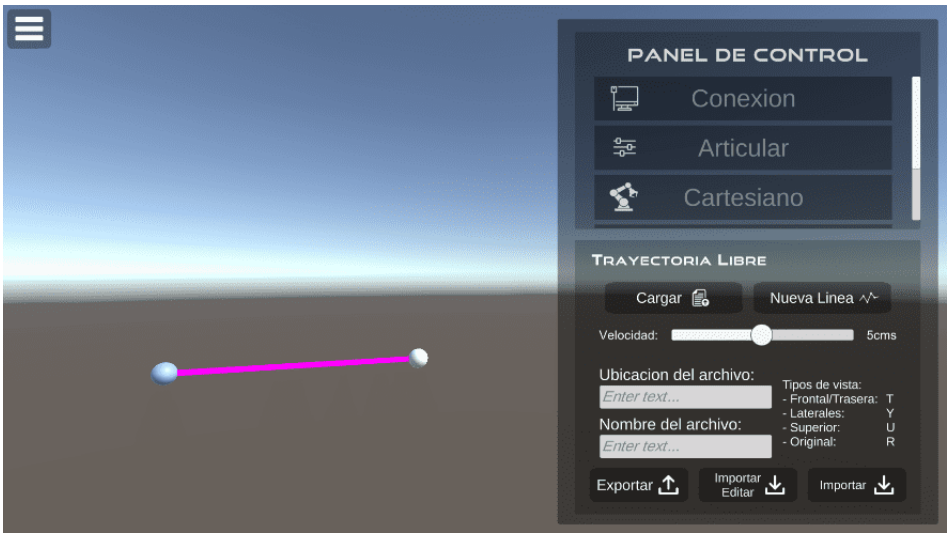


Figura 3.23.: Diseño del control por trayectorias libres.

$$H(t) = (2t^3 - 3t^2 + 1)P_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 - 3t^2)P_1 + (t^3 - t^2)m_1 \quad (3.1)$$

La ecuación previa establece que t es un parámetro de 0 a 1, P_n hace referencia a los puntos de control y m_n a las pendientes correspondientes.

Además de esto, se utiliza una técnica de interpolación *Catmull-Rom* para definir las tangentes en los puntos de control y suavizar la conexión entre los segmentos de la trayectoria. Este proceso se aplica a los puntos intermedios ((3.2)), al punto inicial ((3.3)) y al punto final ((3.4)) de la *spline de Hermite*. En este sentido, es importante mencionar que la variable m_k corresponde a la tangente de los puntos medios y las variables m_{ki} y m_{kf} al punto de inicio y fin de la curva de Hermite, mientras que la variable P_k se refiere al punto de control.

$$m_k = \frac{P_{k+1} - P_{k-1}}{2} \quad (3.2)$$

$$m_{ki} = P_{k+1} - P_k \tag{3.3}$$

$$m_{kf} = P_k - P_{k-1} \tag{3.4}$$

También, se ha incluido una funcionalidad para ajustar la velocidad del robot mediante un deslizador, así como las opciones de importar y exportar localmente las trayectorias que podrán ser posteriormente modificadas por el usuario. Para ello, se ha incorporado una casilla de texto para ingresar el nombre del archivo y otra para definir la ruta donde se desea buscarlo o guardarlo. Es importante destacar que para guardar el archivo en el sistema operativo Ubuntu, se debe elegir una carpeta que tenga permisos de escritura y lectura. Para obtener más detalles al respecto, se puede consultar el Anexo A.4.

En cuanto al proceso de exportación, se toma la matriz de posiciones de los puntos que conforman la trayectoria, se serializa en formato JSON, utilizando la librería *Newtonsoft.Json*. Por otro lado, al importar el archivo, se deserializan los datos utilizando la misma librería, y se crea una matriz con las posiciones de los puntos para luego generar una línea con los datos correspondientes.

Finalmente, este modo de control cuenta con un archivo publicador que permite enviar al robot las posiciones cartesianas de los puntos de la trayectoria creada. Es importante destacar que, antes de enviar las coordenadas, se lleva a cabo una conversión del sistema de referencia, ya que los ejes del marco de Unity difieren de los ejes del sistema de ROS (Tabla 3.1).

ROS	Unity
Eje X	Eje Z
Eje Y	Eje X
Eje Z	Eje Y

Tabla 3.1.: Diferencia del sistema de referencia de ROS y Unity.

La lógica de programación de este modo de control está descrita por los diagramas del Anexo A.7.

3.3.7. Comunicación

Para comunicar la plataforma con el robot *UR3e* o su simulador URSim, se emplea el método de publicación y suscripción de información en tópicos de ROS. Dicha metodología se utiliza para transmitir la información desde la interfaz hacia el robot y viceversa. En las siguientes subsecciones, se detalla el proceso implementado para lograr una comunicación eficiente entre los elementos.

Integración de ROS Bridge

Para lograr la comunicación entre Unity y ROS, se requiere el uso de dos módulos esenciales: RosSharp y Rosbridge. Para integrar Rosbridge, es necesario instalar este complemento desde la terminal de comandos, como se muestran en el Código 3.3.

Código 3.3: Instalacion de Ros Bridge.

```
$ sudo apt-get install ros-<rosdistro>-Rosbridge-server
```

El conjunto de Rosbridge y RosSharp posibilita la comunicación bidireccional y en tiempo real entre Unity y ROS, permitiendo publicar y suscribirse a tópicos, servicios y acciones de ROS. La interacción de estos módulos se describe en el diagrama presentado en la Figura 3.24. Los mensajes generados en Unity son serializados en formato JSON por la librería JSONUtility para poder ser enviados a ROS mediante el puente BridgeServer en el protocolo de comunicación de WebSocket. Una vez publicados, estos datos son deserializados y convertidos desde el formato JSON, a mensajes ROS a través del módulo JSON_Serializer de la librería Rosbridge_Library. Cuando el proceso es inverso, el módulo de Rosbridge convierte los mensajes ROS a un archivo JSON que es enviado mediante el WebSocket para que la biblioteca de JSONUtility los convierta en objetos de C#.

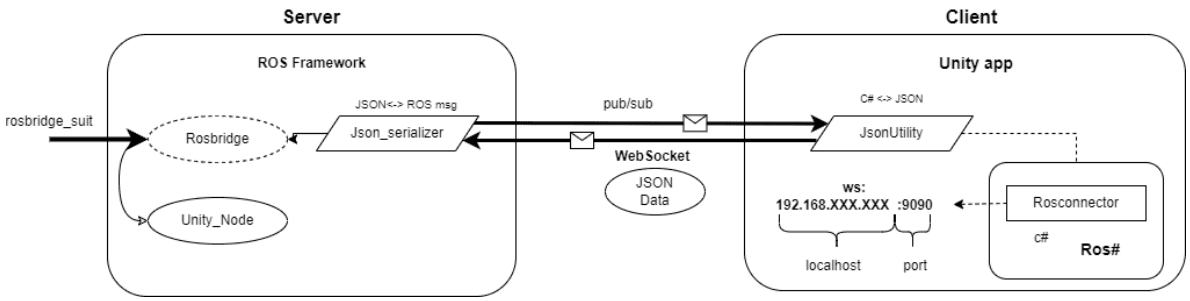


Figura 3.24.: Esquema de comunicación entre Rosbridge y RosSharp.

Para establecer la comunicación, se integra al proyecto de Unity el archivo *RosConnector* de la biblioteca RosSharp. Este archivo proporciona la configuración de los parámetros necesarios para la conexión, como la dirección IP del host de ROS y el puerto de comunicación (Figura 3.25).

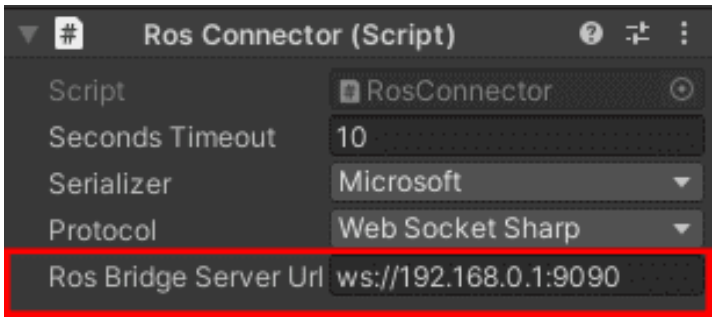


Figura 3.25.: Componente "*RosConnector*" en Unity.

Panel de conexión

Sin embargo, la dirección IP solo es modificable en el modo de edición, lo que impide que el usuario la pueda modificar en una versión ejecutable de la plataforma. Por esta razón, se ha creado un panel llamado "Conexión"que incluye un campo de entrada donde el usuario puede ingresar la dirección IP correspondiente al *host* en el que se ejecuta ROS. Además, se ha agregado un cuadro de texto que proporciona retroalimentación al usuario, mostrando el estado de la conexión (Figura 3.26). Los posibles estados son los siguientes:

- **Desconectado:** cuando no se ha establecido o se ha finalizado la conexión.
- **Conectando...:** mientras se lleva a cabo el proceso de conexión internamente.
- **Falla de conexión:** cuando no se ha podido establecer la conexión debido a la inactividad del nodo de Rosbridge o a una dirección IP incorrecta.
- **Conectado:** cuando se ha logrado establecer la conexión entre Unity y Rosbridge.

Además, es necesario realizar una ligera modificación en el archivo *RosConnector* para recibir y enviar estos parámetros al panel recién creado. Esta modificación implica la creación de una variable de tipo *string* que se definirá mediante el campo de texto de entrada del panel, y una variable que actualizará el cuadro de estado.

Figura 3.26.: Pánel de conexión.

Suscriptores y publicadores en Unity

Con el objetivo de simplificar la comunicación en el proyecto, se ha creado una entidad vacía en la ventana de jerarquía de Unity, denominada "comunicaciónROS". Esta entidad incluirá el archivo de *RosConnector*. Además, contendrá los archivos suscriptores y publicadores que se describen a continuación.

Se incorpora el archivo preescrito *JoinStateSubscriber* de la librería de RosSharp. Este archivo actúa como suscriptor de tipo *sensor_msgs/JointState*, permitiendo la adquisición de los valores de las articulaciones del robot. El archivo es configurado con el nombre del tópico */joint_states*, en el cual el robot publica sus posiciones articulares. Asimismo, se especifica la lista de nombres de las articulaciones del manipulador y la lista de objetos correspondientes a las articulaciones del gemelo digital (Figura 3.27). Para lograr esta sincronización en tiempo real, se agregó previamente a las articulaciones del modelo digital el componente *JointStateWriter*. De esta forma, se garantiza que el gemelo digital copie con precisión las posiciones y movimientos del robot real en todo momento.

Figura 3.27.: Suscriptor articular.

Seguidamente se procede a la creación de otro archivo suscriptor denominado "*CoordenadaSubscriber*", el cual se suscribe al tópico */tf* donde el robot está publicando sus coordenadas cartesianas cada 0.01 segundos, siendo este de tipo *tf2_msgs/TFMessage*. Este suscriptor permite obtener la posición y orientación del efector final del manipulador en tiempo real, lo que posibilita visualizar la coordenada actual en el pánel de control cartesiano (Figura 3.28). Se debe tener en cuenta que la orientación que publica el robot se expresa en cuaterniones, por lo que se ha desarrollado una función en este archivo que convierte la rotación a grados RPY, y esto se logra con ayuda de la librería *Unity.Mathematics*.

Figura 3.28.: Suscriptor cartesiano.

Posteriormente, se desarrollan los archivos publicadores encargados de transmitir los datos de las trayectorias a ROS (Figura 3.29). El primero de ellos es para el modo de control articular y se denomina "*ArtPublisher*". Este publicador de tipo *trajectory_msgs/JointTrajectory* se encarga de publicar en el tópico */Articulaciones* los valores de los deslizadores al ser desplazados mediante el uso del ratón o los botones, así como también la lista de posiciones que se generan en este modo. Es importante destacar que "*ArtPublisher*" incorpora una función que permite la conversión de los datos de rotación expresados en grados RPY a radianes, unidad en la que debe ser publicado el mensaje.

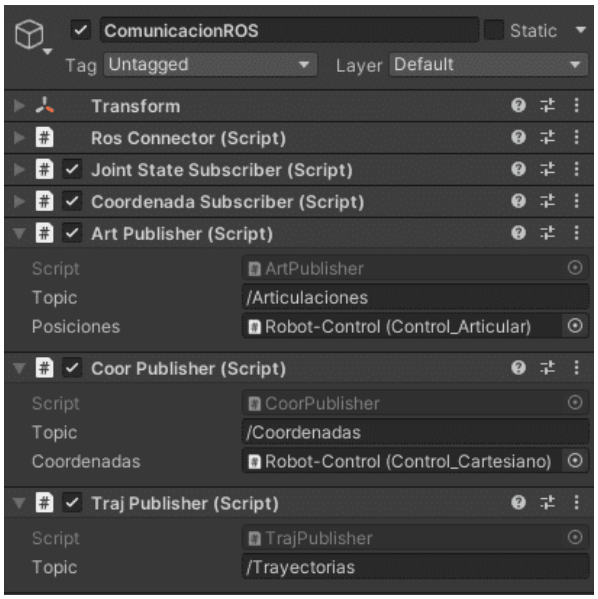


Figura 3.29.: Archivos publicadores.

El segundo archivo publicador, "*CoorPublisher*", pertenece al tipo *CartesianControl_msgs/CartesianTrajectory* y está diseñado para trabajar en conjunto con el modo de control cartesiano. Su función es publicar los datos del control cartesiano en el tópico */Coordenadas*. Este archivo tiene la capacidad de enviar una o varias coordenadas cartesianas de la trayectoria y su velocidad. Además, realiza la conversión de unidades de centímetros a metros para la posición y de radianes a cuaterniones para la orientación, con el objetivo de publicar el mensaje en el formato adecuado de ROS.

Por último, se desarrolló el archivo publicador "*TrajPublisher*", perteneciente al tipo *CartesianControl_msgs/CartesianTrajectory*, que tiene como función enviar al tópico */Trayectoria* la matriz de posiciones cartesianas que conforman la línea del modo de control de trayectoria libre. Con estos archivos integrados en el proyecto, es posible establecer la comunicación de Unity a ROS, tal y como se muestra en el diagrama que se presenta en la Figura 3.30.

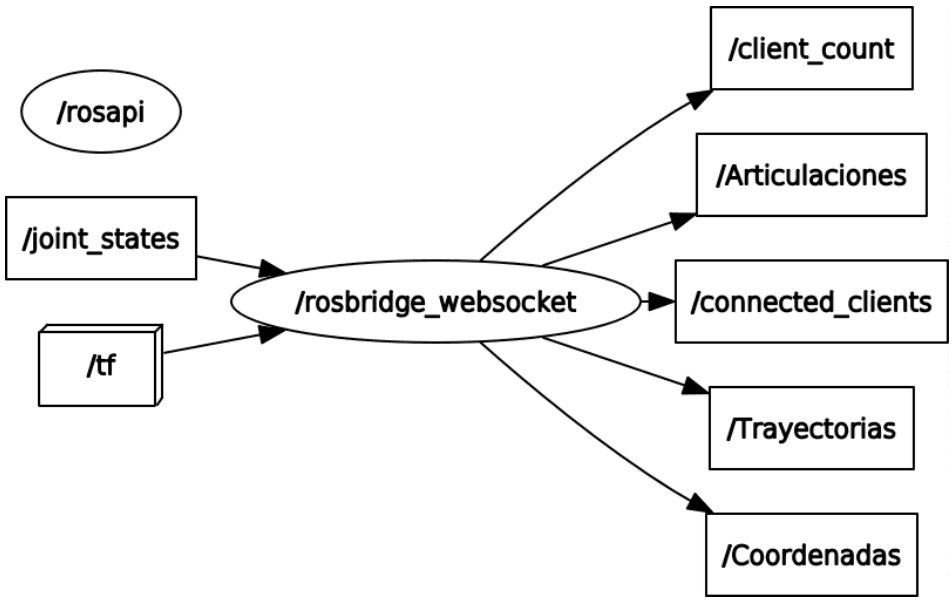


Figura 3.30.: Nodo de Unity con sus tópicos suscriptores y publicadores.

Controlador ROS ur_robot_driver

Por otro lado, para manipular el robot *UR3e*, se utiliza el *ur_robot_driver* como controlador. Este permite la comunicación entre ROS y los cobots de Universal Robots, proporcionando una interfaz ROS que permite a los usuarios enviar y recibir información a través de tópicos y servicios ROS como la posición y la velocidad del robot. Gracias a

esto, es posible personalizar el comportamiento del robot y conocer su estado en tiempo real, utilizando aplicaciones genéricas adaptables a las necesidades específicas de cada usuario [41].

En el diagrama de la Figura 3.31, se puede observar que este controlador crea un nodo llamado */ur_hardware_interface* para el robot, que se conecta a todos los tópicos y servicios que poseen los controladores activos del robot. Entre los controladores disponibles, se destaca el */scaled_pos_joint_traj_controller* y el */pose_based_cartesian_traj_controller*, que contienen los tópicos a los que se enviarán las trayectorias articulares o cartesianas. Además, también se pueden apreciar los tópicos */joint_state* y */tf*, que se mencionaron anteriormente, a los cuales se suscriben los suscriptores de Unity.

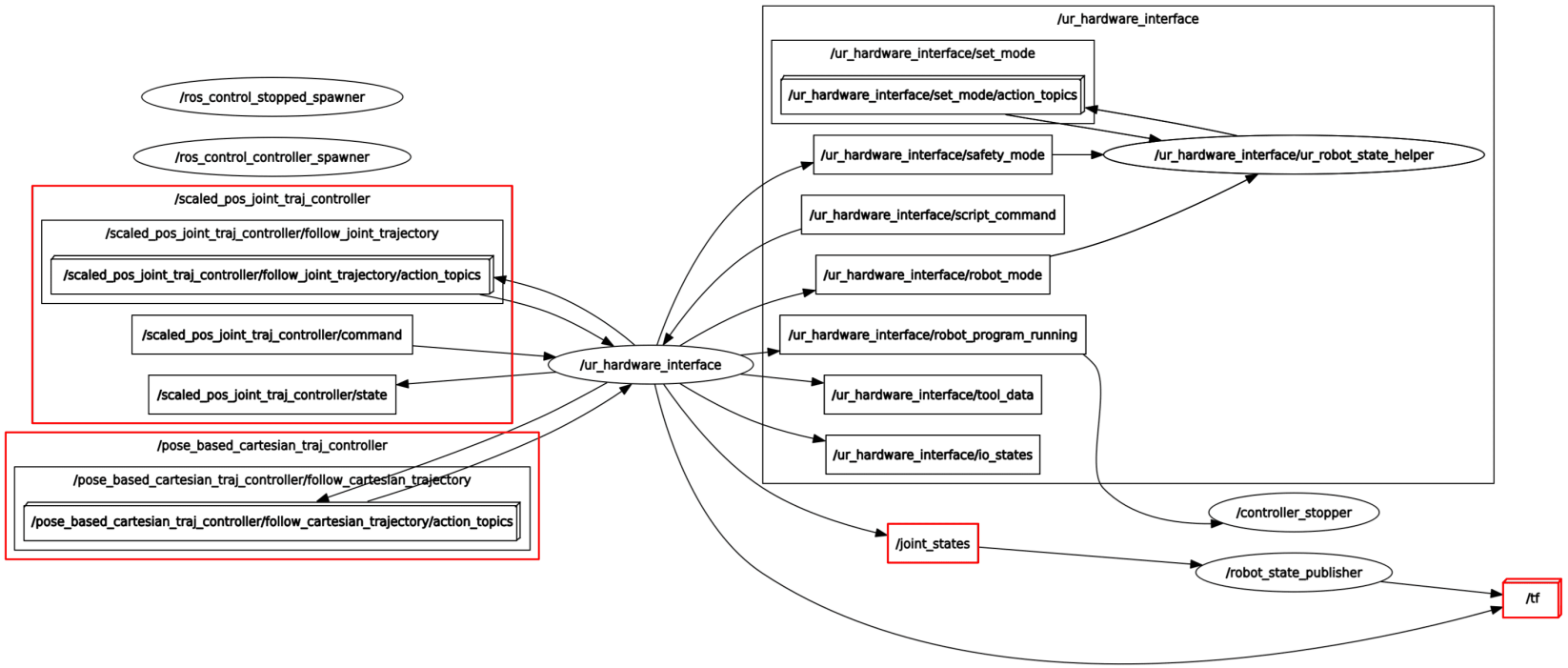


Figura 3.31.: Nodo del controlador del robot con sus tópicos suscriptores y publicadores.

Python

Para transmitir los mensajes ROS desde el nodo de Unity al nodo del robot, se emplean dos archivos de Python encargados de tomar estos datos, organizarlos en los mensajes correspondientes y enviarlos a través de los tópicos de los controladores del robot. El primer archivo, llamado "*ListarDatos.py*", es un suscriptor que crea un nodo ROS de Python y se suscribe simultáneamente a los tres tópicos publicados desde el nodo de Unity (Figura 3.32). Cuando se recibe un mensaje en alguno de estos tópicos, el archivo organiza los datos en forma de lista para enviarlos posteriormente como parámetros cuando es invocada alguna de las funciones del controlador.

Por otro lado, se dispone del archivo "*Controladores.py*", el cual contiene varias funciones, entre ellas, la carga y la inicialización de los controladores */scaled_pos_joint_traj_controller* y */pose_based_cartesian_traj_controller*. Además, cuenta con tres funciones que pueden ser invocadas por el archivo anterior, las cuales reciben los datos y los organizan en los tipos de mensaje */FollowJointTrajectoryGoal* para trayectorias articulares y */FollowCartesianTrajectoryGoal* para trayectorias cartesianas.

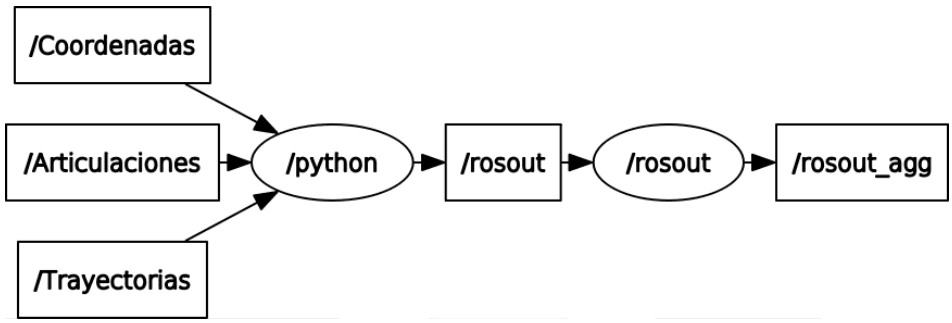


Figura 3.32.: Nodo de Python con sus tópicos suscriptores.

Arquitectura de comunicación

La implementación de la arquitectura de comunicación de la plataforma se puede apreciar en la Figura 3.33. Esta muestra los tres bloques que incluyen un nodo de ROS (Unity, Python y el controlador del robot) y presenta los archivos suscriptores y publicadores que forman parte de la plataforma, así como los tópicos a los que se publican los datos. Además, se han incluido líneas de color rojo y azul que indican los tipos de mensaje que contienen datos para las trayectorias articulares y cartesianas, respectivamente.

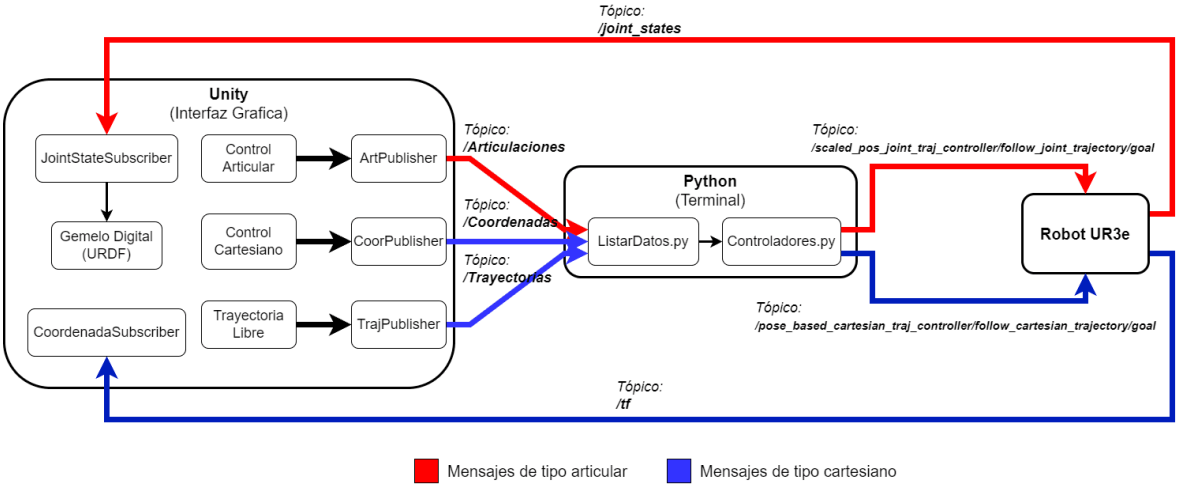


Figura 3.33.: Arquitectura de comunicación.

3.4. Simulación de la plataforma

En un principio, URSim permitió la simulación del manipulador robótico a través de la plataforma construida. Esto facilitó el envío de órdenes de movimiento y evitó preocupaciones sobre posibles errores que podrían causar movimientos no deseados y provocar colisiones del robot. Además, se logró identificar fallas en la interpolación de los *waypoints* que se definen desde el pánel cartesiano para desplazar el robot a una ubicación específica.

En esta sesión se especifica el uso que se le dio al simulador URSim en el proyecto, así como también el uso del robot real. Dado que las configuraciones son similares, se presta especial atención a los puntos en los que hay divergencia.

3.4.1. Identificación del host

Para establecer una conexión, es necesario utilizar la herramienta *URCap External*, la cual habilita una ventana dentro de *Polyscope*. En dicha ventana, se debe ingresar la dirección IP del *host* que ejecuta ROS y agregar esta instrucción en el apartado de *Program*, activar el robot y pulsar sobre el botón *Play*. Esto mostrará en la terminal de *ur_robot_driver* el anuncio "*Robot connected to reverse interface. Ready to receive control commands*", indicando la conexión con ROS. Es importante destacar que tanto el robot de URSim (Figura 3.34(a)), el robot real (Figura 3.34(b)) o la plataforma, deben estar conectados a la misma red para garantizar el éxito de la conexión (Figura 3.35).

(a) Ingreso de IP desde URSim (b) Ingreso de IP desde el *Tech Pendant*

Figura 3.34.: Ventana del *URCap External*.

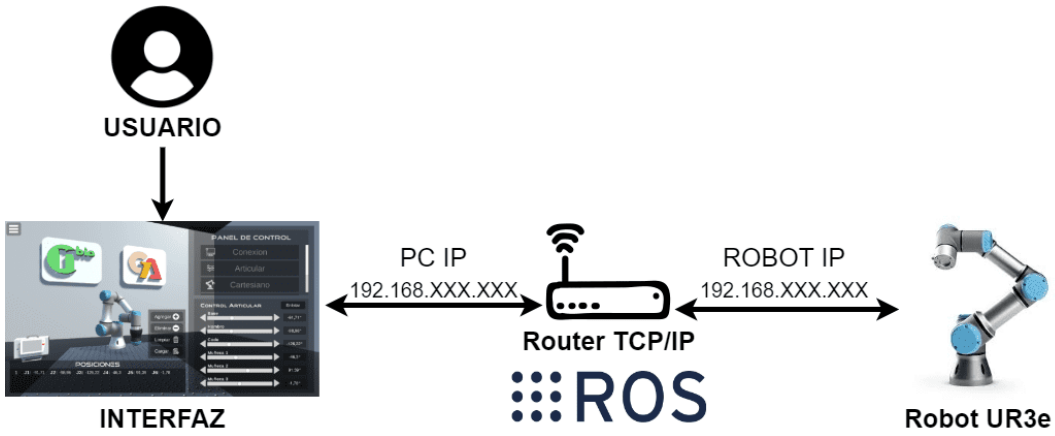


Figura 3.35.: Esquema de conexión de la plataforma software con el robot UR3e (digital o real).

3.4.2. Ejecución del ur_robot_driver

Para ejecutar el driver, se usan los siguientes comandos predeterminados en la terminal (Código 3.4). Estos ubican el paquete del driver en el directorio *catkin_ws*, configuran ROS y lanzan el controlador correspondiente respectivamente.

No obstante, se puede reducir el número de comandos necesarios que ejecutan el controlador, para ello es necesario configurar el archivo */.bashrc* con la siguiente instrucción desde una terminal (Código 3.5).

Código 3.4: ejecución del controlador.

```
$ cd UR3e/catkin_ws
$ source devel/setup.bash
# Esto ejecuta el controlador del robot
$ roslaunch ur_robot_driver ur3e_bringup.launch robot_ip:=<YOUR_IP>
   limited:=true
```

Código 3.5: Configuración bashrc.

```
# Configura bashrc
$ echo source /home/ur/UR3e/catkin_ws/devel/setup.bash >> ~/.bashrc
```

Gracias a lo anterior, se mejora la fluidez en la ejecución de instrucciones ROS, ya que no será necesario configurar el *bash* cada vez que se abre una nueva terminal, por lo que la instrucción se reduce a uno solo comando (Código 3.6). Esto no solo ahorra tiempo, sino que también evita posibles confusiones.

La naturalidad de la instrucción (Código 3.6) se resume así: *roslaunch* es el comando que se utiliza para iniciar un lanzamiento en ROS. El paquete *ur_robot_driver* contiene el controlador del robot UR, y el archivo de configuración *ur3e_bringup.launch* se utiliza para iniciar el controlador del robot *UR3e*. Para establecer la conexión con el robot, se utiliza el argumento *robot_ip := YOUR_IP ;* donde *YOUR_IP* es la dirección IP del robot que se va a manipular. Además, el argumento *limited:=true* se utiliza para activar el modo limitado en el controlador del robot. Este modo restringe el rango de movimiento del robot para garantizar la seguridad al trabajar cerca de él. Si se establece en *false*, el robot podrá moverse libremente, lo que puede ser peligroso si hay personas cerca.

Código 3.6: ejecución del controlador del robot.

```
# Esto ejecuta el controlador del robot
$ roslaunch ur_robot_driver ur3e_bringup.launch robot_ip:=<YOUR_IP>
   limited:=true
```

Es importante comprender que la dirección IP del robot proporcionada por el simulador es la misma a la que está asociado el computador que ejecuta ROS. Sin embargo, en el caso del robot real, este dispone de un *socket* en el *router*, y por lo tanto, tiene una dirección IP distinta que debe estar dentro del dominio de la red.

3.4.3. Ejecución de Rosbridge

Se lanza el nodo Rosbridge desde una nueva terminal por comando como se muestra en (Código 3.7).

Código 3.7: ejecución del Ros Bridge.

```
$ roslaunch Rosbridge_server Rosbridge_websocket.launch
```

Lo anterior se utiliza para iniciar el nodo Rosbridge en ROS en un puerto 9090 predeterminado, lo cual mostrará en la terminal el mensaje de "*Rosbridge WebSocket server started at ws://0.0.0.0:9090*".

3.4.4. Conexión con la plataforma

Para utilizar la plataforma, es necesario localizar el archivo "*ListarDatos.py*" y ejecutarlo en una terminal utilizando el Código 3.8. Este archivo se encarga de recibir los datos enviados desde la interfaz de Unity y de cargar e inicializar los controladores cada vez que se envía una nueva trayectoria. Es importante destacar que este archivo debe permanecer en ejecución mientras se utiliza la plataforma.

Código 3.8: ejecución del archivo ListarDatos.py.

```
$ python3 ListarDatos.py
```

Posteriormente, desde la interfaz de usuario se debe seleccionar el botón de *Conexión*, el cual despliega el panel de conexión. Aquí se ingresa la dirección IP asignada al computador que se está utilizando, tal como se muestra en la Figura 3.36. Luego, se presiona la opción "*Conectar*" y se verifica que el estado del robot cambie a "*Conectado*". Asimismo, en la terminal que está ejecutando Rosbridge, deberá mostrar que un dispositivo ha sido conectado, como se muestra en la Figura 3.37. En ese momento el robot digital de la plataforma tomará la posición del robot real. Una vez conseguida la conexión, se pueden enviar datos de movimiento desde los páneles de control (articular, cartesiano y trayectoria libre), lo que permite que tanto el robot real como el digital se desplacen sobre la trayectoria o puntos especificados.

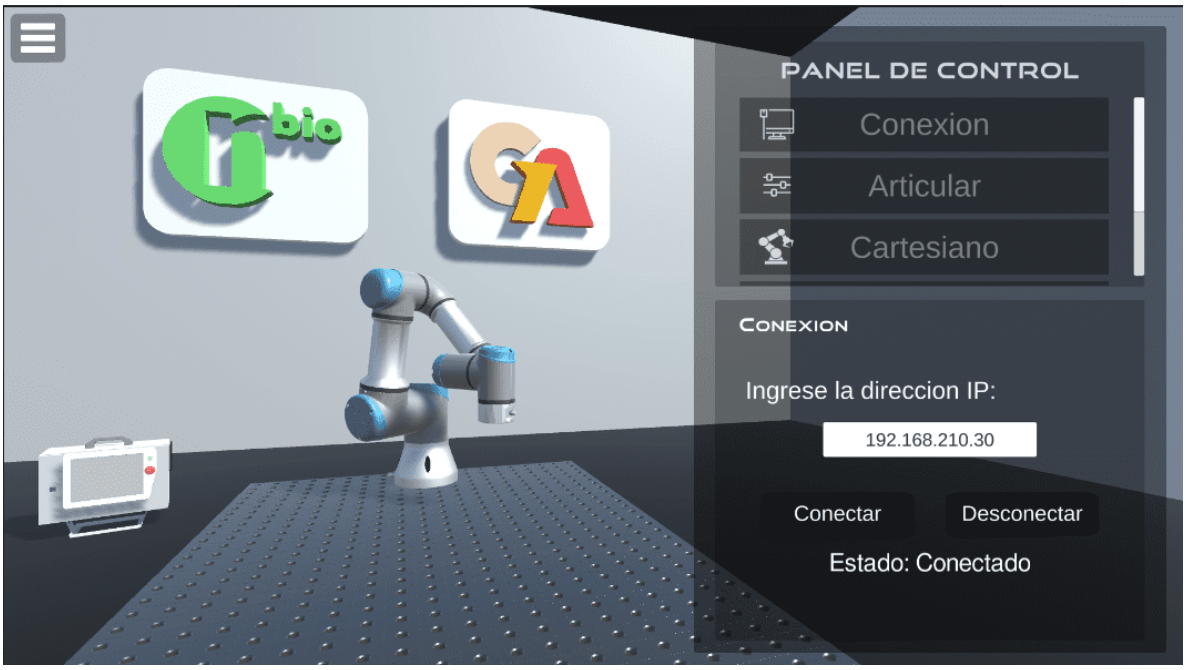


Figura 3.36.: Conexión desde la plataforma software.

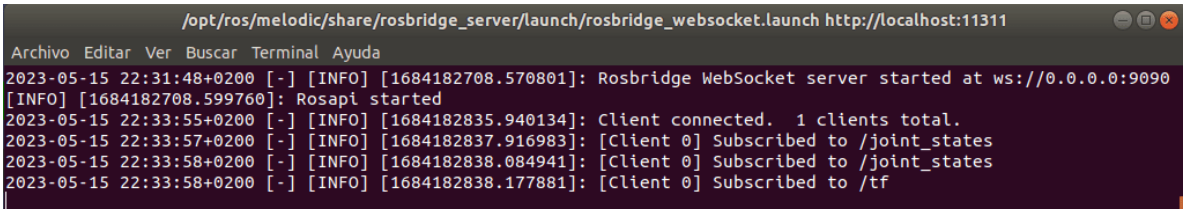


Figura 3.37.: Terminal de Rosbridge.

3.4.5. ROS Controllers cartesian

Durante la ejecución de movimientos cartesianos, se registraron múltiples puntos de referencia que incluían tanto la posición como la orientación del efector final. Estos puntos generaban una trayectoria que pasaba a través de estos, permitiendo identificar desplazamientos irregulares y peligrosos realizados por el robot cuando se le afectaba la rotación, tratándose de problemas en la interpolación de la rotación [47], tal como se muestra en la Figura 3.38. Esta representación visual proporciona una evidencia clara de los movimientos problemáticos que se produjeron durante la tarea.

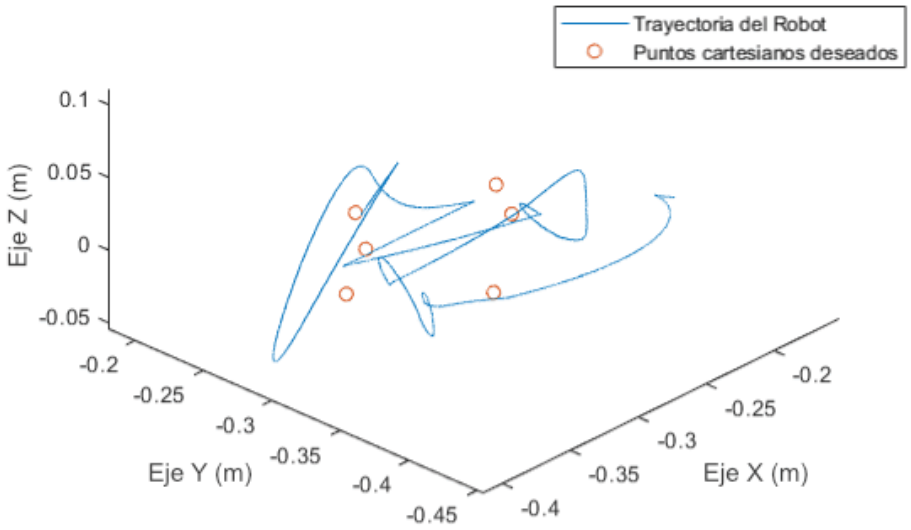


Figura 3.38.: Trayectoria del robot sin *ROS controller cartesian*.

[48] menciona en su repositorio que el control cartesiano no está soportado en ROS, por lo que en este trabajó se recurrió al paquete *ROS Controllers cartesian*.El objetivo de este conjunto es llenar esta brecha y facilitar el inicio del control cartesiano.

Sin embargo las funciones que componen este controlador no aseguran que el robot tome el camino más corto, por lo que si el producto punto entre dos orientaciones es negativo, el signo se invierte en una de ellas para garantizar que tomemos el camino más corto [49].

Para instalar el complemento al proyecto inicialmente se clonó del repositorio de github el controlador *ROS Controllers cartesian* [48], y se sitúo en el espacio de trabajo *catkin_ws*, para posteriormente compilar y acoplar el nuevo paquete.

3.5. Pinza universal

Las pinzas universales son herramientas altamente eficaces para la manipulación de objetos, gracias a su capacidad de adaptarse a diferentes formas y ofrecer una mayor área y puntos de contacto [50]. Se pueden dividir en dos tipos: activas y pasivas. Las pinzas universales activas tratan de imitar la mano humana mediante diseños antropomórficos, lo que implica un control sofisticado y las hace más costosas. Por otro lado, las pinzas universales pasivas suelen ser más sencillas de construir, ya que constan de un cuerpo elástico que se deforma al entrar en contacto con el objeto. Dentro de

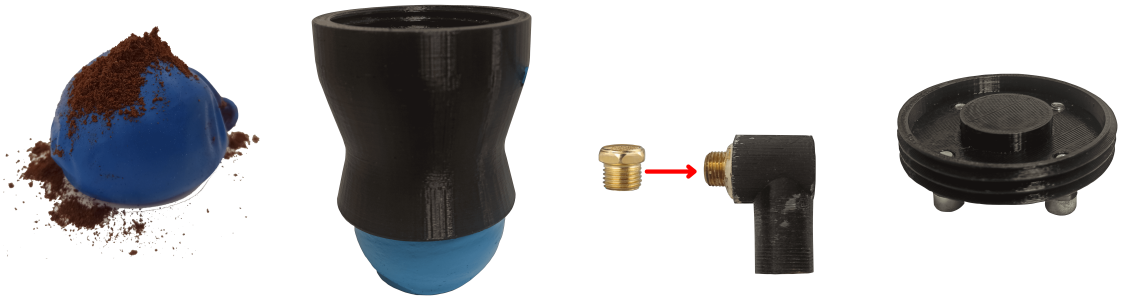
las pinzas universales pasivas, destaca la subcategoría de interferencia granular, la cual se basa en la transición de materiales granulares de baja densidad a un cuerpo rígido de alta densidad provocado por una tensión externa. En palabras más sencillas, es la transición de materia granular de un estado de fluido a sólido, debido a la dilatancia de Reynols [51]. Además, estudios han demostrado que el sistema óptimo de fabricación para las pinzas de interferencia granular emplea una presión negativa y positiva para modular la transición de interferencia, tal como se demostró en [52]. La versatilidad de estas herramientas se refleja en su amplia variedad de aplicaciones. En el caso de este trabajo, se llevó a cabo la implementación de una pinza para realizar operaciones de "*pick and place*"programadas a través de la plataforma diseñada.

3.5.1. Elaboración y funcionamiento de la pinza

Para construir la pinza de interferencia granular se comienza seleccionando un material deformable. En este caso, se eligió el café molido debido a su amplia utilización en estudios previos como material granular [53], ya que permite aplicar el principio de Reynolds. El café molido se vierte dentro de un globo de látex (Figura 3.39(a)), creando una bola que se deforma para adoptar la forma del objeto.

Posteriormente, se diseñó en el software *Fusion 360* una base para sujetar el globo lleno de café (Figura 3.39(b)). Esta pieza se creó mediante la técnica de impresión 3D y posee un cuello interno que permite que el globo pase a través de ella para ubicar un filtro en medio. Para el filtro neumático se requiere una unión (Figura 3.39(c)) que se diseñó y fabricó de la misma manera. Esta unión se ajusta a presión en el centro del cuello del cuerpo para crear un sello hermético y permitir así extraer e introducir aire dentro del globo, evitando que el material granular sea succionado.

Por último, se diseñó un conector (Figura 3.39(d)) que se sujeta al efector final del robot mediante tornillos y se acopla fácilmente a la pinza mediante un sistema de rosca. Una vez que todas las piezas están unidas, se obtiene la pinza final tal como se muestra en la Figura 3.40.



(a) Globo de látex con café (b) Base de la pinza (c) Filtro neumático (d) Conector del robot

Figura 3.39.: Componentes para la fabricación de la pinza de interferencia granular.

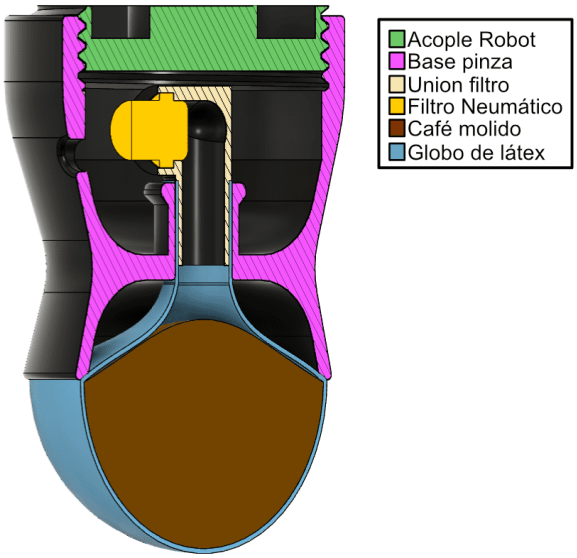


Figura 3.40.: Diseño final de la pinza de interferencia granular.

Como se describe en la Figura 3.41, el funcionamiento de la pinza consiste en ubicarla sobre el objeto y presionarla contra el mismo aplicando la denominada fuerza de activación, de modo que la pinza se envuelva alrededor del objeto y adquiera su forma. Luego se utiliza una presión negativa para succionar aire y compactar el material granular interno, formando un cuerpo rígido que permite desplazar el objeto. Para soltar el objeto, se aplica una presión positiva a la pinza, lo que hace que la bola de látex se expanda y suelte el objeto, devolviendo el material granular a su estado inicial de baja densidad. La pinza construida en este caso cuenta con un sistema de compresión que aplica presión tanto positiva como negativa con una fuerza de 90 KPa. Sin embargo,

el control de activación de estas presiones se lleva a cabo manualmente mediante la apertura y cierre de una válvula de paso.

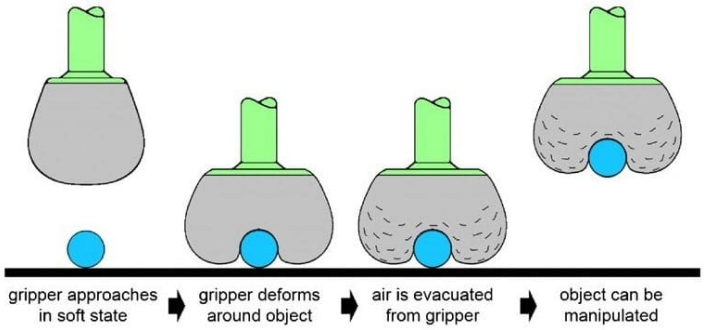


Figura 3.41.: Principio de funcionamiento de la pinza de interferencia granular [54].

3.5.2. Modelo digital de la pinza

Para incorporar el modelo 3D de la pinza a la plataforma, se utiliza el software *Fusion 360* y se agrega el *plugin* "*Fusion2URDF*"(Anexo A.5), el cual permite exportar las piezas creadas en formato URDF. Luego, se importan las piezas al proyecto y se las organiza en la jerarquía de Unity de forma similar a como se importó el gemelo digital del robot. Sin embargo, se agrega la pinza a la última articulación del robot digital para que se mueva simultáneamente, como se ilustra en la Figura 3.42.

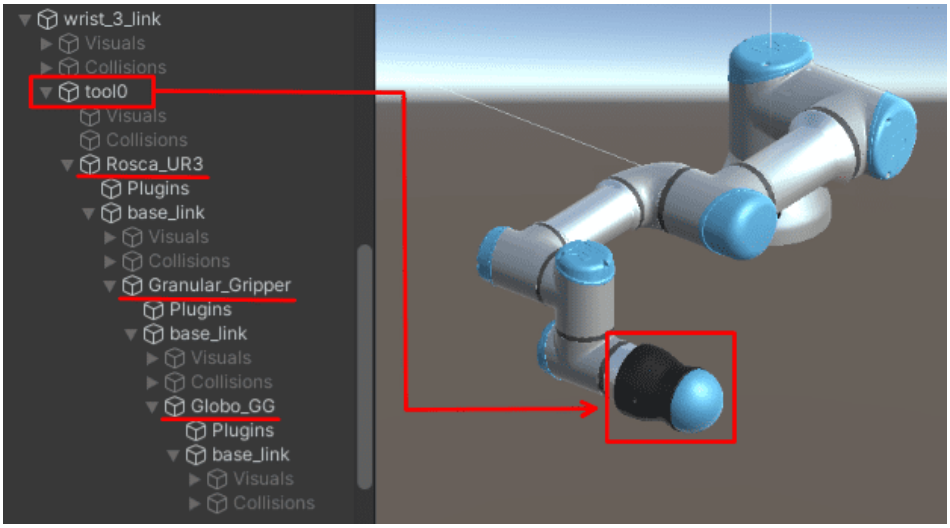


Figura 3.42.: Pinza digital dentro de la plataforma.

4. Resultados

Como resultado de este proyecto, se ha desarrollado una plataforma software que ofrece diversos modos de control para la manipulación de objetos tridimensionales con distintas geometrías por parte del robot *UR3e*. Esta plataforma ha sido implementada utilizando el software Unity y se comunica a través del *middleware* de ROS, lo cual permite el intercambio de información mediante la publicación y suscripción de tópicos con mensajes estructurados.

Dicha plataforma brinda la capacidad de crear diferentes tipos de trayectorias. En primer lugar, se pueden generar trayectorias de tipo articular, las cuales permiten variar los grados de libertad de cada articulación. Asimismo, es posible crear trayectorias de tipo cartesiano, donde se indican las coordenadas deseadas del efector final del robot, tanto en posición como en rotación. Por último, se pueden diseñar trayectorias libres, las cuales ofrecen al usuario la libertad de modelar visualmente la ruta de la trayectoria que desee seguir.

Además, se ha logrado construir e implementar una pinza de interferencia granular, que ofrece como resultado la capacidad de manipular diversos objetos gracias a su diseño que genera fuerzas de agarre mediante la utilización de partículas granulares.

- Enlace video demostrativo: https://youtu.be/fcsOL6F6S14

4.1. Interfaz gráfica de usuario

La interfaz gráfica es la representación visual con la que el usuario podrá interactuar y observar. Para esta interfaz, se ha desarrollado un escenario virtual que presenta el gemelo digital del robot *UR3e*, el cual replica los movimientos del robot real. Este escenario se presenta en un entorno 3D, lo que permite al usuario desplazarse por toda la habitación y visualizar el robot desde diferentes ángulos utilizando el teclado (Anexo A.1.2).

La interfaz también incluye un panel principal ubicado en el lado derecho, el cual contiene varias opciones disponibles para el usuario. Estas opciones incluyen la conexión

al robot, el control articular, el control cartesiano y el control por trayectoria libre. En la Figura 4.1 se muestra el modo de control articular, el cual presenta 6 deslizadores que el usuario puede ajustar para modificar la rotación de cada una de las 6 articulaciones del robot. Junto a cada deslizador se muestra en tiempo real el valor en grados de cada articulación. En el panel inferior se muestra una lista de valores articulares que el usuario puede agregar para crear una trayectoria con diferentes posiciones.

Figura 4.1.: Panel del control articular.

La Figura 4.2 presenta el panel de control cartesiano, donde el usuario puede ingresar los valores de posición y rotación deseados para la ubicación del efector final del robot. Este panel también proporciona retroalimentación sobre los valores de posición y rotación actuales del robot, y cuenta con un deslizador que permite al usuario ajustar la velocidad de los movimientos. Este modo también incluye un panel para que el usuario pueda listar una serie de coordenadas y así crear puntos de paso en la trayectoria.

Figura 4.2.: Panel del control cartesiano.

Por último, el modo de control por trayectoria libre muestra una línea de color rosa en el centro del escenario, representando la ruta que seguirá el robot. El usuario tiene la posibilidad de modificar esta línea utilizando el ratón de la computadora para crear la trayectoria deseada (Figura 4.3). Asimismo, se dispone de un deslizador que permite controlar la velocidad a la cual el robot ejecutará la trayectoria. Además, el usuario cuenta con la capacidad de guardar las rutas creadas en el almacenamiento local del computador, así como importar rutas previamente generadas.

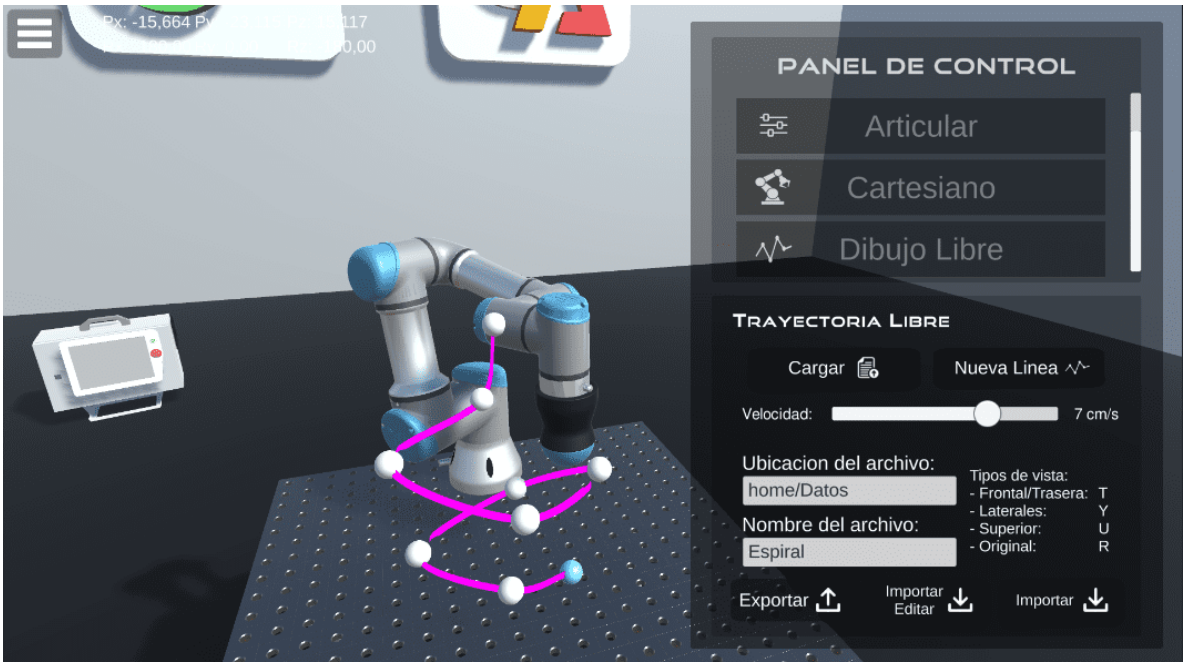


Figura 4.3.: Panel del control por trayectorias libres.

En resumen, la interfaz gráfica proporciona al usuario la posibilidad de observar el robot e interactuar con él. Con opciones de control articular, control cartesiano y control por trayectoria libre, el usuario puede ajustar los movimientos y configuraciones del robot según sus necesidades. La interfaz también facilita la creación y gestión de trayectorias, así como el almacenamiento de rutas personalizadas.

4.2. Pinza de interferencia granular

Durante esta prueba, se llevó a cabo la sujeción de objetos no convencionales utilizando la pinza granular, con el objetivo de evaluar su capacidad de agarre. Mediante esta experimentación, se buscó obtener un mayor conocimiento acerca de las capacidades y limitaciones de dicha pinza para sujetar objetos de diferentes formas. Los elementos probados fueron un juego de llaves, una jeringa, un alicate, un instrumento quirúrgico genérico, un bolígrafo y una llave mecánica, tal como se observa en las fotos de la Figura 4.4.

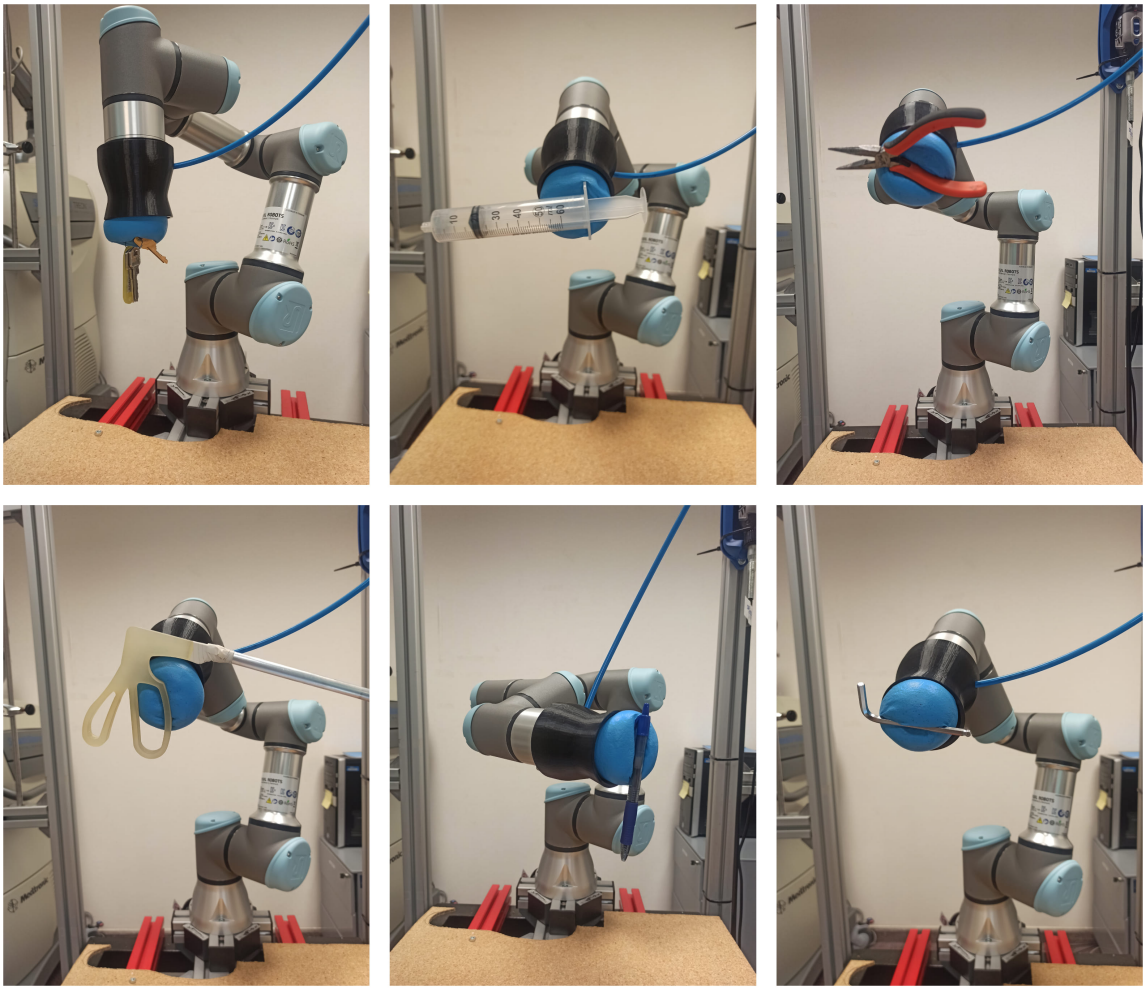


Figura 4.4.: Sujeción de objetos no convencionales con pinza granular.

Con esta prueba se pudo verificar que el globo con café molido se adapta al objeto y proporciona la rigidez suficiente para sujetar y sostener una diversidad de objetos, sin embargo la capacidad para agarrar objetos planos se dificulta (Figura 4.4), pues para la pinza es casi imposible adaptarse a la forma del objeto de manera que proporcione suficiente agarre, como se menciona en [55], que demostró un manejo exitoso de objetos con formas muy diversas.

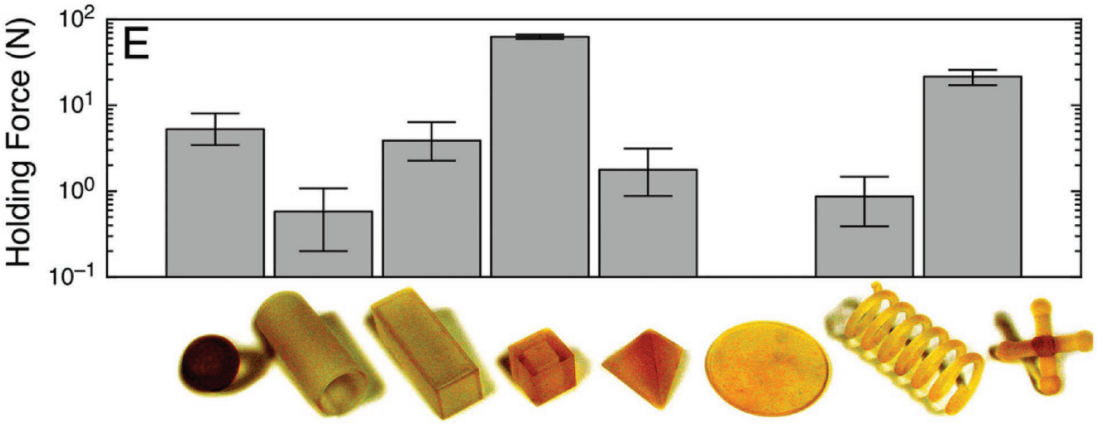


Figura 4.5.: Estudio de sujeción de objetos por la pinza granular [55].

4.3. Manipulación de objetos con diferentes geometrías

A continuación se presentan varios ejemplos de pruebas que demuestran cómo se controla el robot a través de la plataforma para manipular diferentes objetos y llevar a cabo diversas tareas. La evaluación de su desempeño en estas tareas resulta de vital importancia para determinar su aplicabilidad y eficiencia en diversos contextos.

4.3.1. Manipulación de objetos con el control articular

Con el propósito de evaluar el modo de control articular de la plataforma, se llevó a cabo el presente experimento. Se dispusieron tres objetos de plástico sobre una mesa adyacente al robot, con la finalidad de programar desde la plataforma las posiciones articulares que el manipulador debía adoptar para elevar las piezas a una altura superior (Figura 4.6). Previo a ello, se procedió a posicionar manualmente el robot en las ubicaciones requeridas para el desplazamiento de los tres objetos. Cada posición se replicó mediante el ajuste de los deslizadores del panel articular hasta alcanzar los grados deseados. Una vez obtenida una posición, era registrada en la lista, generando un total de 25 posiciones distintas. Posteriormente, se ejecutó el programa y se envió el conjunto de posiciones al robot, logrando cumplir exitosamente el objetivo propuesto en la prueba. A partir de este experimento, se pudo observar que el control articular no brinda una precisión elevada en términos de grados, aunque continúa siendo una forma eficiente de desplazar rápidamente el robot de un punto a otro.

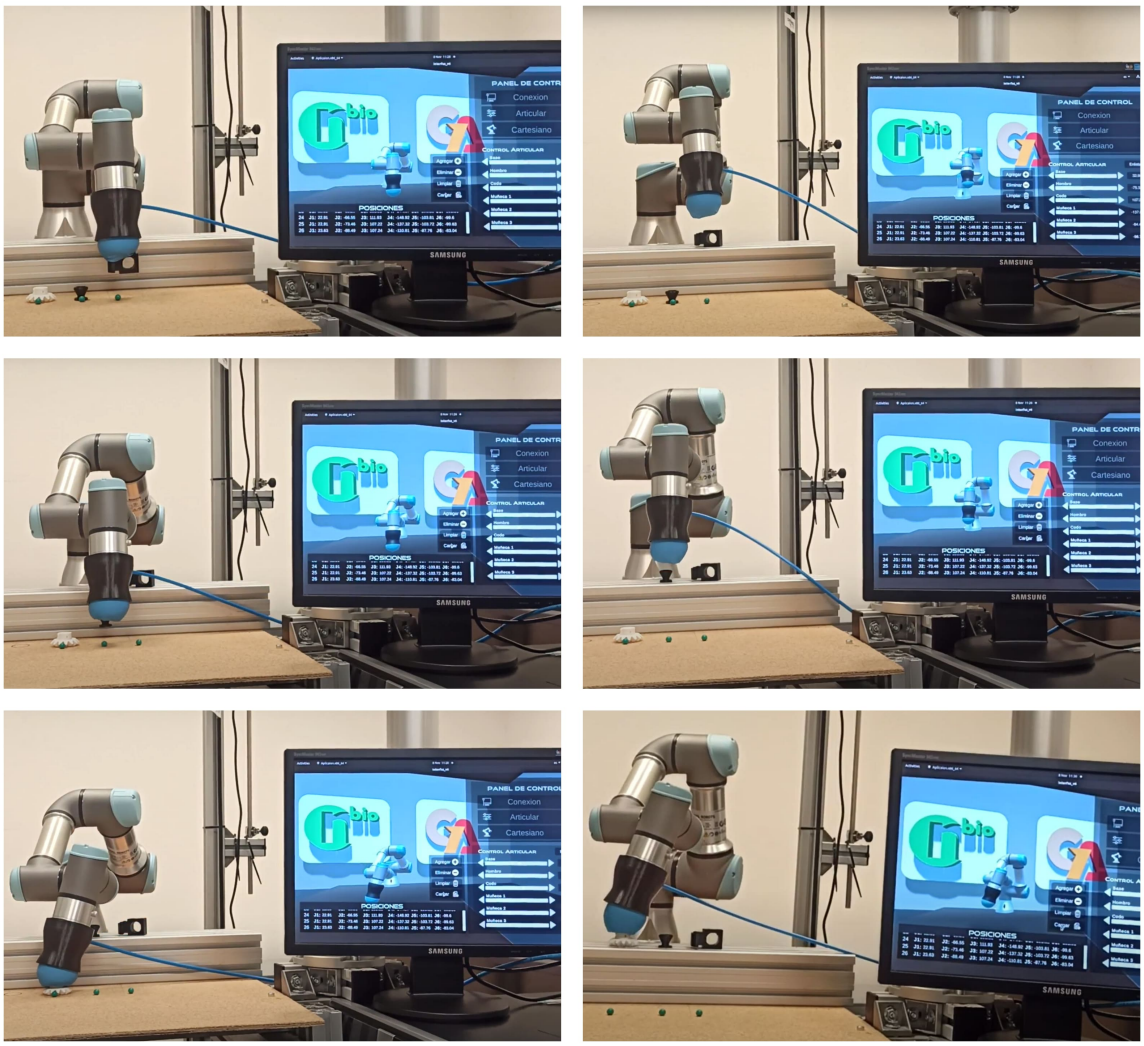


Figura 4.6.: Manipulación de objetos desde el control articular.

4.3.2. Manipulación de objetos con el control cartesiano

En el marco de esta prueba, se llevó a cabo la colocación estratégica de cuatro objetos con geometrías distintas en posiciones específicas determinadas por los marcadores verdes. Los objetos se encuentran inicialmente en una posición horizontal y deben ser trasladados hacia una posición vertical final, indicada por los marcadores rojos. Las ubicaciones de cada marcador son conocidas, y a través del panel cartesiano se le proporcionan al robot las coordenadas necesarias para que se desplace hacia dichas ubicaciones. Utilizando su pinza, el manipulador del robot toma estos objetos y los transporta hasta la posición deseada, tal como se muestra en la Figura 4.7.

Figura 4.7.: Manipulación de objetos desde el control cartesiano.

4.3.3. Manipulación de objeto para dibujo

En esta prueba, se establecieron una serie de coordenadas en el panel cartesiano con el objetivo de permitir el desplazamiento del robot hacia una posición específica donde se encontraba un objeto cilíndrico, concretamente un marcador (Figura 4.8). Una vez que el robot tomó el marcador, procedió a su desplazamiento y rotación con un ángulo de inclinación determinado, para trazar un triángulo sobre una hoja de papel con dimensiones de 21 *cm* de base y 29,7 *cm* de altura (Figura 4.9). Esta prueba tuvo como finalidad evaluar la capacidad del robot para llevar a cabo tareas de manipulación dadas desde la plataforma.

Figura 4.8.: Prueba de sujeción de objeto cilíndrico irregular.

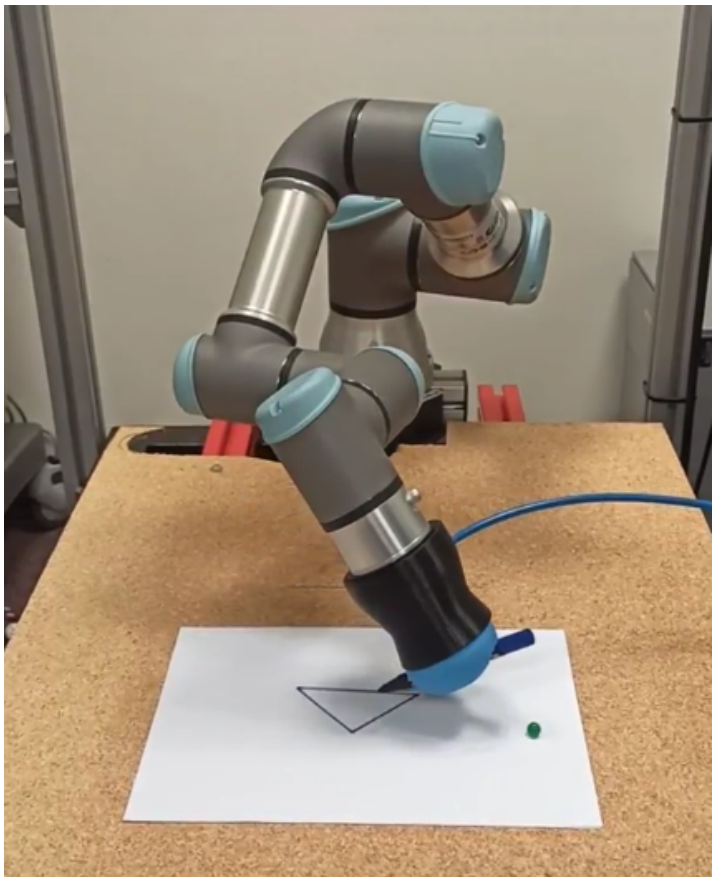


Figura 4.9.: Dibujo de triangulo sobre una hoja de papel en una base plana.

4.3.4. Construcción de torre con piezas cúbicas

En esta prueba, se dispusieron tres cubos regulares de dimensiones de 3 cm, los cuales se ubicaron estratégicamente sobre tres puntos de referencia en una mesa plana. El desafío consistió en que el robot formara una torre con estos objetos, demostrando su eficiencia en la realización de tareas de *pick and place*.

La Figura 4.10 muestra claramente cómo el manipulador se desplaza para sujetar y trasladar los cubos a las posiciones establecidas en el panel cartesiano. Esta representación visual resulta fundamental para visualizar y comprender la planificación y ejecución de las acciones del robot en relación a la tarea de manipulación de los cubos. La imagen proporciona una visión detallada del proceso y permite evaluar la eficacia y precisión del robot en la realización de dichas tareas.

Figura 4.10.: Construcción de torre con piezas cúbicas.

4.3.5. Pick and place con objetos esféricos

La prueba se llevó a cabo mediante la colocación de cuatro esferas de 2 *cm* de diámetro en diferentes posiciones sobre una mesa. El objetivo consistía en recoger dichas esferas y depositarlas en un vaso de cristal ubicado en el centro de la mesa (Figura 4.11). Para lograrlo, se registraron en el panel cartesiano las posiciones y orientaciones en las que el robot debía agarrar las esferas para posteriormente soltarlas en el vaso. Estos datos fueron enviados desde la plataforma desarrollada y capturados por el robot, que se desplazó con ángulos variables para transportar las esferas hasta el recipiente, manteniendo una velocidad constante de 5 cm/s.

El objetivo principal de esta evaluación fue analizar la capacidad de la pinza para manipular objetos tridimensionales con forma esférica. Este escenario de prueba representa

una situación realista en la que el robot podría enfrentarse en diferentes entornos.

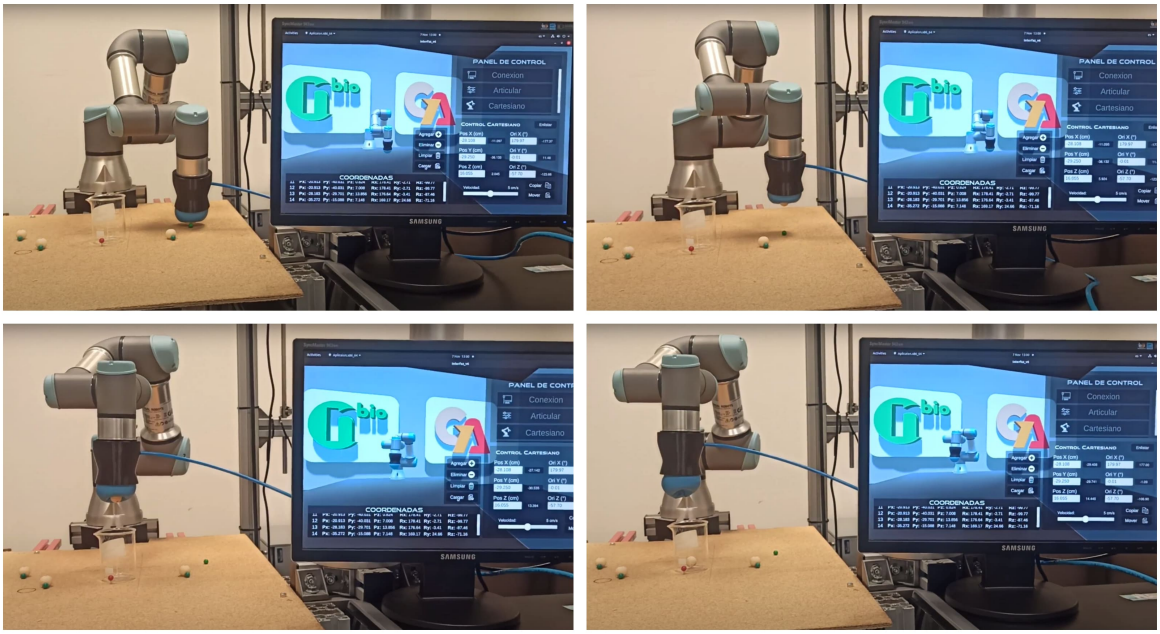


Figura 4.11.: Manipulación de objetos esféricos.

4.4. Precisión de la plataforma

La prueba tuvo como objetivo evaluar la precisión entre una serie de coordenadas enviadas desde la plataforma y las coordenadas alcanzadas por el robot. Para ello, se procedió a colocar manualmente el efector final del robot sobre seis puntos de referencia, previamente definidos en el espacio tridimensional mediante marcadores dispuestos sobre una mesa plana. Estos marcadores se organizaron en dos columnas, separadas por 7.5 *cm*, y tres filas, separadas por 6 *cm* (Figura 4.12). Con el fin de obtener los datos del manipulador en cada punto, se utilizó el tópico */tf*, el cual proporciona las posiciones y orientaciones del efector final. Posteriormente, se ingresaron las seis coordenadas registradas en el panel de control cartesiano para que el robot alcanzara las mismas posiciones siguiendo la trayectoria descrita en la Figura 4.13. Dicha trayectoria es calculada por el controlador del robot a partir de los puntos enviados. Con el propósito de validar lo anterior, se capturaron los valores de salida de la plataforma a través del tópico */Coordenadas* y se monitoreó nuevamente el tópico */tf* para obtener las nuevas posiciones.

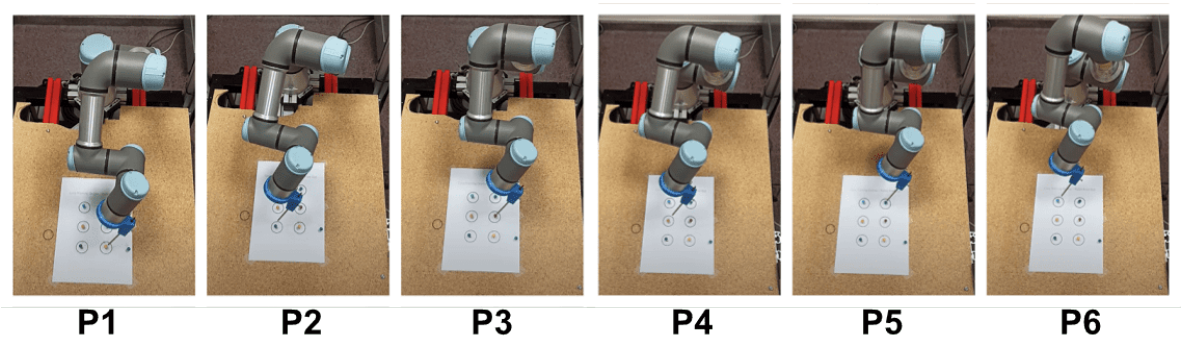


Figura 4.12.: Toma de los puntos de referencia.

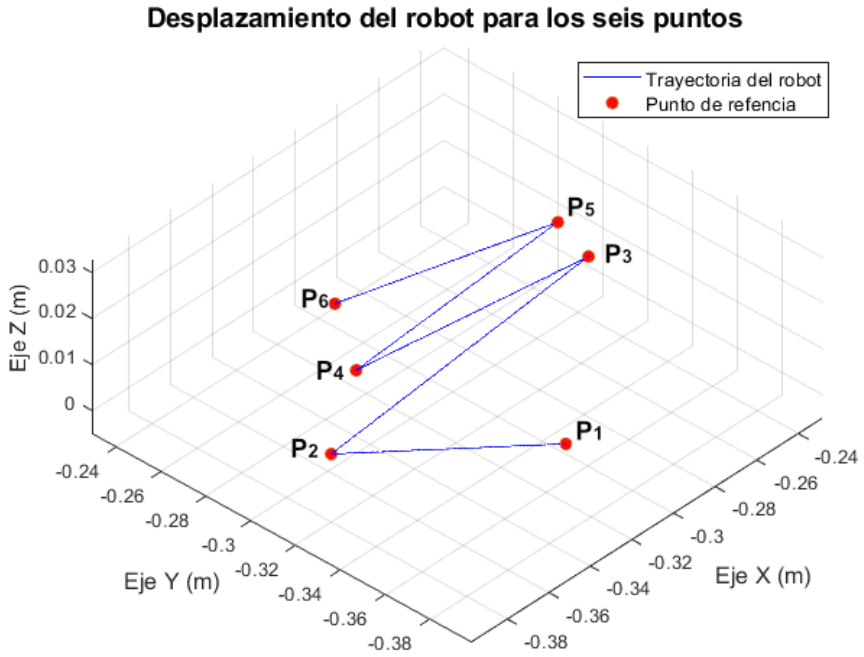


Figura 4.13.: Trayectoria del robot para los seis puntos ingresados.

Los datos obtenidos fueron organizados en una tabla de Excel y se compararon utilizando un algoritmo en Matlab para determinar el error máximo mediante una medición euclidiana. Los resultados revelaron un error máximo de $4{,}43 \times 10^{-6}$ m en la posición y de $5{,}05 \times 10^{-4}$ rad en la rotación (Figura 4.14). Sin embargo para hallar el error en la rotación fue necesario realizar la conversión de la orientación recibida, de cuaterniones a grados RPY previamente, por lo que esta conversión puede afectar

ligeramente los resultados. Por consiguiente, se supone que el error en la rotación del punto número 1 fue mayor que el de los demás.

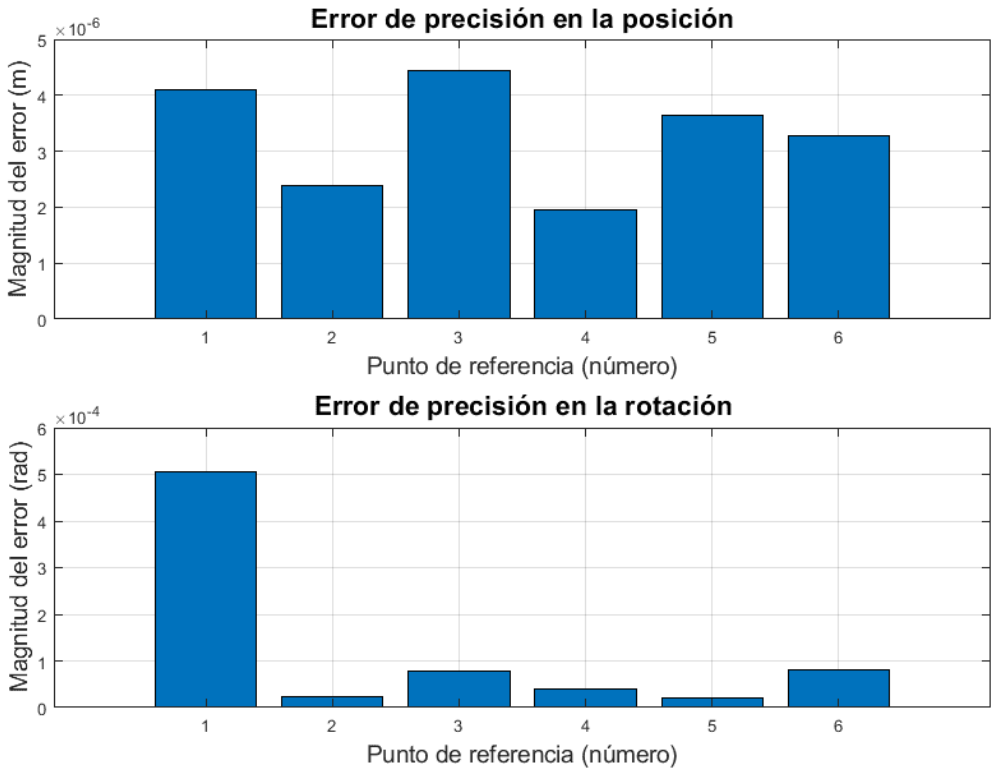


Figura 4.14.: Error de precisión en la posición y rotación.

4.5. Evaluación de trayectoria libre

Para evaluar el control de trayectoria libre, se realizó una prueba que constó de tres casos distintos. El objetivo de esta prueba fue trasladar un objeto irregular de un lugar a otro sin que el robot colisionara con obstáculos en el entorno conocido. Para lograr este propósito, se empleó una ruta inicial en forma de línea recta, la cual se adaptó según cada caso específico. Cada ruta trazada en los tres casos consistía en un conjunto de puntos cartesianos que variaban en función del número de segmentos.

Con el fin de determinar la precisión de la ruta planificada en la plataforma en comparación con la trayectoria real del robot, se midió la distancia euclidiana entre el conjunto de puntos enviados desde la plataforma y los datos obtenidos del efector final del robot a través del tópico */tf*. Se utilizó un algoritmo en Matlab para calcular dicha

distancia, generando gráficas para cada caso y analizando el error máximo obtenido.

- **Caso I:** En esta situación particular, se plantea el escenario de tomar una pinza quirúrgica desde una posición inicial definida y trasladarla del punto A al punto B, evitando la zona de colisión que está delimitada por un prisma cuadrado con dimensiones de $19 \times 14 \times 9$ cm (obstáculo 1). Para lograr este objetivo, se realizó una modificación en la ruta planificada, de manera que se adaptara a una posición ubicada fuera del área de colisión. Esto permitió que el instrumento quirúrgico pudiera ser llevado hacia su destino de manera segura y sin riesgo de colisión (Figura 4.15).

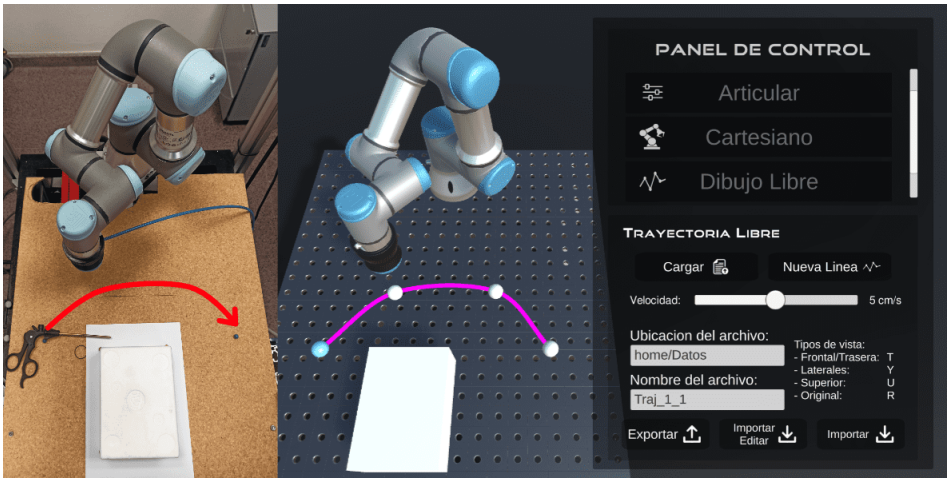


Figura 4.15.: Evasión del obstáculo caso I.

La ruta seguida estaba compuesta por un total de 150 puntos, y su recorrido se completó en un tiempo total de $10 segundos$, manteniendo una velocidad constante de 5 cm/s. Con el propósito de obtener posiciones precisas del robot a lo largo de su trayectoria real, se realizó un muestreo a una frecuencia de 0,0666 s utilizando el tópico */tf* para obtener las posiciones en cada punto. Posteriormente, se compararon estas posiciones obtenidas con las posiciones enviadas desde la plataforma, utilizando la métrica de distancia euclidiana para determinar el error máximo de seguimiento. Este error fue registrado como $2{,}75 \times 10^{-4}$ m (Figura 4.16).

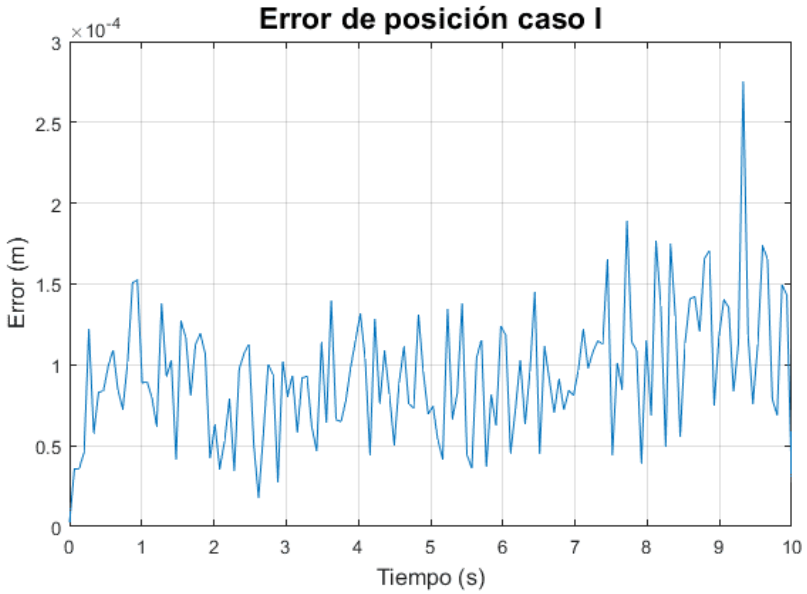


Figura 4.16.: Error de trayectoria caso I.

- **Caso II:** Siguiendo una estructura similar al caso anterior, se llevó a cabo el movimiento del objeto de un lado a otro. Sin embargo, en esta ocasión, la pinza quirúrgica experimentó una rotación de 90° alrededor de su eje, lo que nuevamente generó una zona de colisión con el obstáculo. Para superar esta situación, se procedió a modificar la trayectoria con el fin de evitar cualquier colisión entre el objeto y el obstáculo (Figura 4.17).

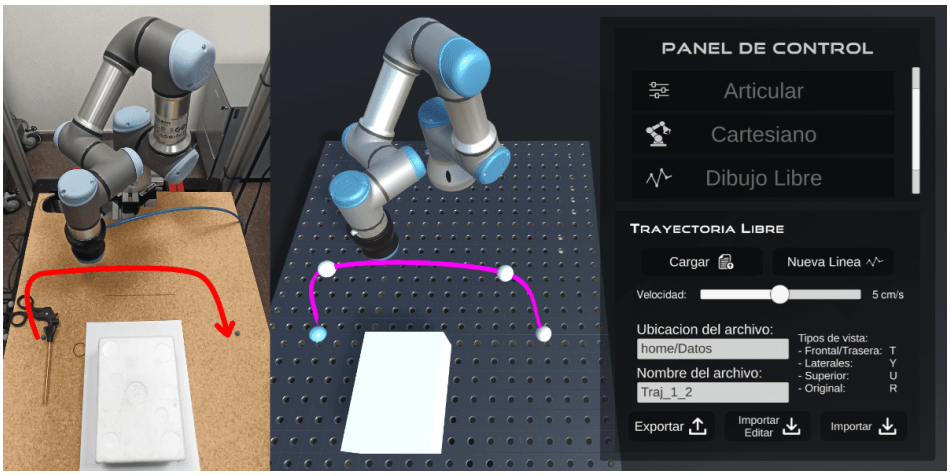


Figura 4.17.: Evasión del obstáculo caso II.

En este segundo caso, la trayectoria utilizada no sufrió modificaciones en cuanto al

número de puntos empleados. No obstante, se registró un incremento en el tiempo total necesario para completar el recorrido con la misma velocidad anterior, pasando a ser de 12 s . Como consecuencia de este cambio, se ajustó el tiempo de muestreo utilizado para obtener las posiciones a través del tópico */tf*, estableciéndolo en 0,08 s. Los resultados obtenidos revelaron que el error de precisión más elevado en esta ruta modificada fue de $2{,}08 \times 10^{-4}$ m, tal como se ilustra en la Figura 4.18.

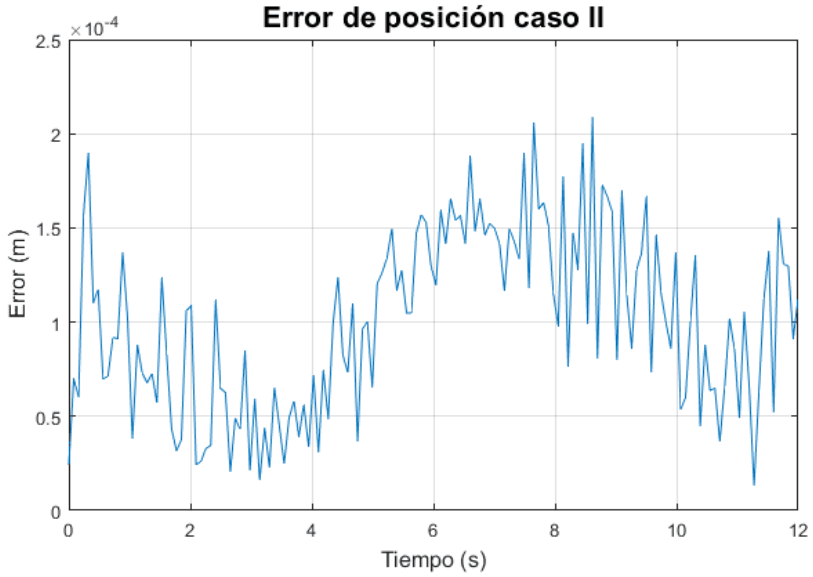


Figura 4.18.: Error de trayectoria caso II.

- **Caso III:** En este caso, se siguió la misma línea del caso II, pero se añadió un obstáculo adicional al espacio de trabajo, con dimensiones de $19 \times 6 \times 14$ cm. Se identificó que este nuevo obstáculo generaba una zona de colisión en un tramo de la trayectoria anterior. Como resultado, se realizó una nueva modificación en la trayectoria, generando una ruta de mayor complejidad que evitaba la colisión con ambos obstáculos (Figura 4.19).

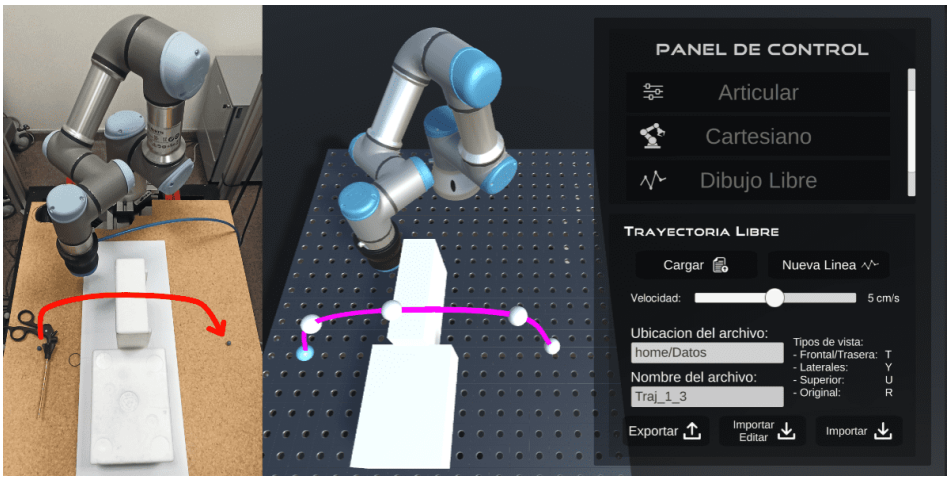


Figura 4.19.: Evasión del obstáculo caso III.

En este caso final, se diseñó una ruta planificada que constaba de 200 posiciones, y el tiempo total requerido para completar el recorrido fue de 13 s a una velocidad constante de 5 cm/s. Esto implicó que el tiempo de muestreo utilizado para obtener los valores de la trayectoria real fue de 0,065 s. Se compararon los puntos enviados con la trayectoria realizada por el manipulador utilizando la medición de la distancia euclidiana. Como resultado de estas comparaciones, se observó un error máximo de precisión de $1{,}37 \times 10^{-4}$ m, como se muestra en la Figura 4.20.

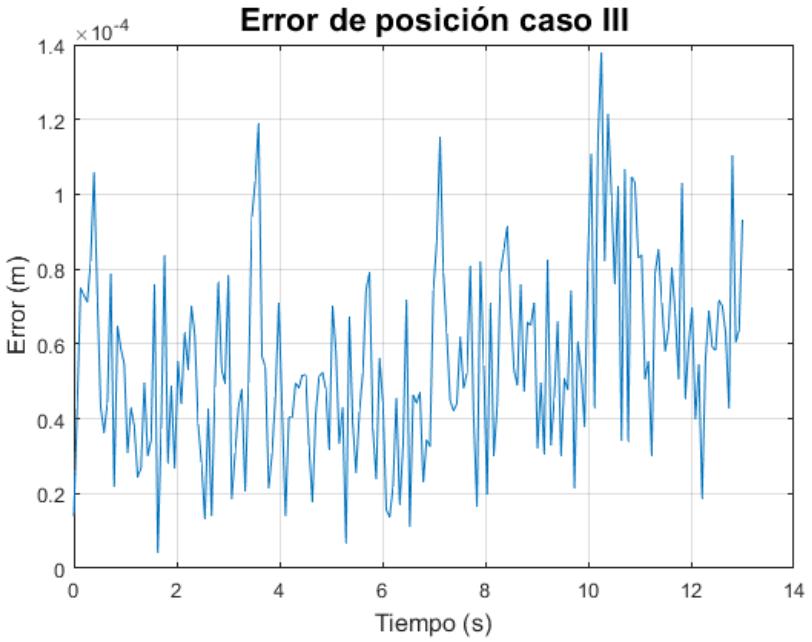


Figura 4.20.: Error de trayectoria caso III.

En la Figura 4.21 se puede observar la variación de la trayectoria en los tres casos, evidenciando cómo se adaptó para evitar los obstáculos presentes.

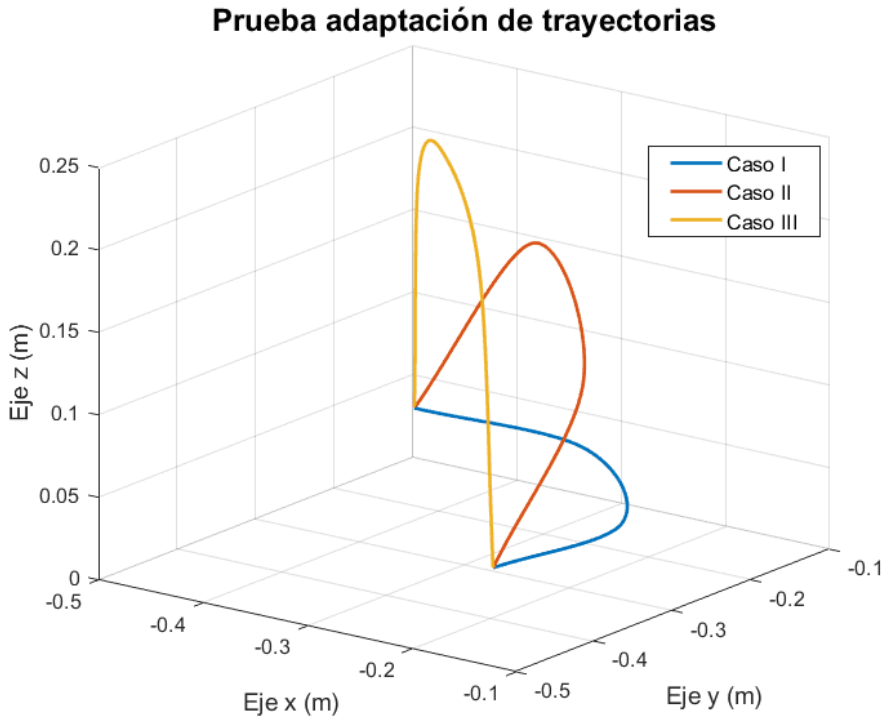


Figura 4.21.: Modificación de la trayectoria en los tres casos.

4.6. Comparativa con otros métodos de control

En este estudio, se realizó una comparativa de precisión entre diferentes métodos de control utilizados para mover el robot. Para realizar la prueba, se definieron 6 coordenadas de posición (Tabla 4.1) que se envían al robot para que ejecute una trayectoria pasando por estos puntos objetivos, con una velocidad constante de 5 cm/s.

Tabla 4.1.: Coordenadas de puntos de referencia.

N°	Posición x	Posición y	Posición z
1	-32,9193	-37,363	1,3444
2	-37,6758	-32,2216	0,4744
3	-28,4874	-33,2678	-0,2102
4	-32,9298	-29,1713	0,14
5	-24,3402	-29,5401	-0,3904
6	-29,3788	-24,6995	-0,2193

La finalidad fue evaluar la exactitud con la que el robot alcanzaba estos puntos utilizando cuatro métodos de control distintos.

- **Método 1:** Consistió en el envío de datos a través de la plataforma desarrollada en el proyecto. Utilizando el *middleware* ROS (*Robot Operating System*), se enviaron las coordenadas de la trayectoria al tópico del control cartesiano, donde fueron procesadas por el controlador *ur_robot_driver* para planificar la trayectoria a ejecutar.

- **Método 2:** Implicó el envío de datos a través de un archivo de Python utilizando URScript, el lenguaje de programación de los cobots de Universal Robots. Las coordenadas numéricas se convirtieron en cadenas de texto y se generó un mensaje en URScrip. Este mensaje fue enviado al tópico */ur_hardware_interface/script_command* de ROS, que a su vez lo transmitió al controlador *ur_robot_driver* para la planificación de la trayectoria.

- **Método 3:** Consistió en el envío de datos mediante URScrip, pero en lugar de ser publicados en ROS, se utilizaron *sockets* para establecer una comunicación directa entre el cliente Python y el robot. Se especificó la dirección IP y el puerto de comunicación del robot, que estaban determinados por el enrutador al que estaba conectado.

- **Método 4:** Se basó en el control directo desde la interfaz de *Polyscope* con la *Tech Pendant*. A través de esta interfaz se programaron los seis puntos objetivo y el controlador interno del robot calculó la trayectoria requerida para alcanzarlos. Cabe destacar que este método prescindió del uso de ROS.

Para evaluar la precisión de cada método, se monitoreó el tópico */tf*, que proporcionaba las coordenadas de la trayectoria ejecutada por el robot. Estas coordenadas se compararon

con los puntos objetivo para determinar la discrepancia en la trayectoria obtenida mediante la medición de distancia euclidiana. Los resultados de la comparativa se representaron gráficamente, permitiendo una visualización clara de las diferencias en la precisión entre los métodos evaluados (Figura 4.22). A partir de estos resultados, se pudo observar que el método de control de la plataforma ofrecía un error de precisión máximo de $1{,}85 \times 10^{-4}\ m$.

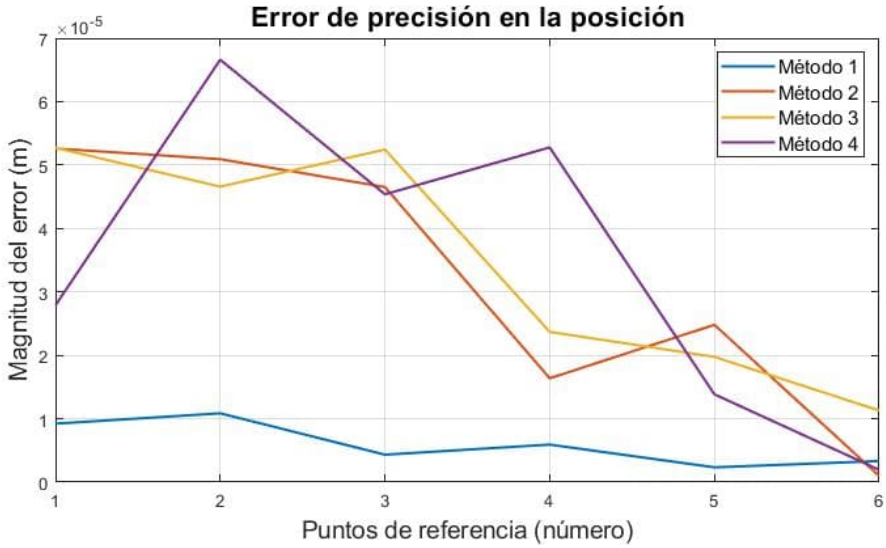


Figura 4.22.: Error de precisión de los cuatro métodos de control.

4.7. Repositorio del proyecto

En esta sección se presenta el repositorio de GitHub que alberga todo el proyecto realizado en este trabajo de investigación. En dicho repositorio, se encuentra una guía de instalación del simulador URSim, un manual de uso de la plataforma, videos de soporte y los archivos de los diseños 3D que componen la pinza de inferencia granular. Además, también se encuentran disponibles tanto el archivo ejecutable como el archivo editable que conforman esta contribución.

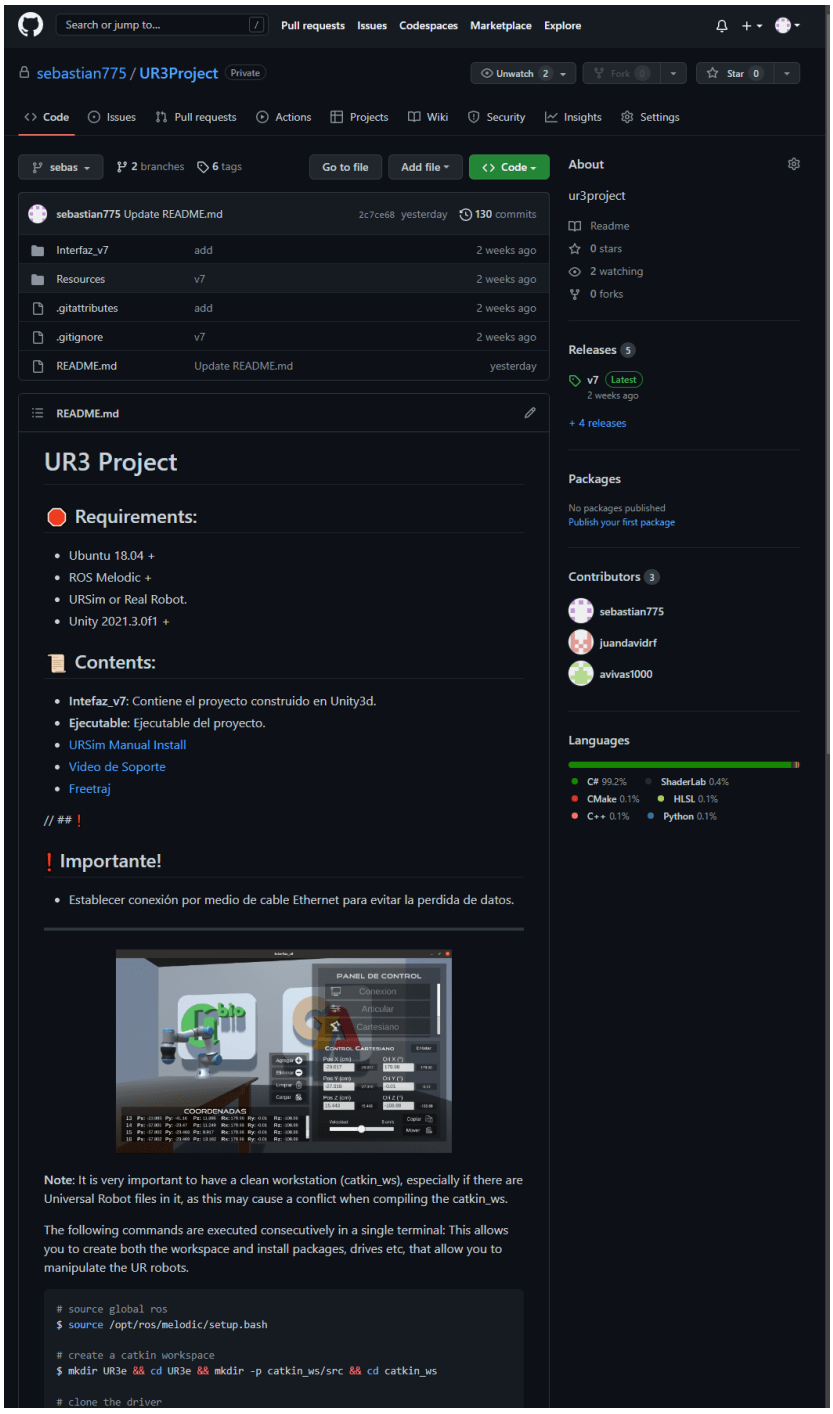


Figura 4.23.: Repositorio en Github del proyecto.

4.8. Publicaciones

Publicaciones en el marco de esta tesis:

- J.D. Ruiz, J.S. Montenegro, O.A. Vivas
  Plataforma para la manipulación de un gemelo digital de un robot UR3
  XI Congreso Internacional de Ingeniería Mecánica, Macarrónica y Automatización 2023, Cartagena, Colombia, 2023.

- J.S. Montenegro, J.D. Ruiz, J.D. Romero, J. Manrique, O.A. Vivas, J.M. Sabater
  Generador 3D de trayectorias libres de colisiones para un manipulador UR3e con pinza blanda
  Revista Iberoamericana de Automática e Informática Industrial. (artículo en tercera revisión de parte de la revista).

5. Conclusiones y trabajos futuros

5.1. Conclusiones

En el marco del convenio interinstitucional establecido entre la Universidad del Cauca (Popayán, Colombia), y la Universidad Miguel Hernández (Elche, España), se llevó a cabo una estancia de investigación de tres meses en los laboratorios del grupo nBio, que se detalla en este trabajo de grado. Durante este período, se desarrolló una plataforma software utilizando el motor gráfico de Unity para el control de un robot *UR3e*. En dicha plataforma, se incluyó un gemelo digital del manipulador, y se implementaron tres modos de control distintos: control articular, control cartesiano y control por trayectoria libre. Para establecer una comunicación efectiva entre la plataforma y el robot, se utilizó ROS, empleando una conexión por WebSocket a través de Rosbridge. Además, se diseñó una pinza blanda de interferencia granular, utilizando café molido como material granular. Esta pinza es capaz de tomar la forma y agarre del objeto deseado mediante una fuerza de succión que compacta los granos en su interior. Para evaluar los resultados obtenidos, se realizaron rigurosas pruebas de manipulación de objetos, donde se empleó la plataforma para controlar el robot y diseñar las trayectorias requeridas en operaciones de *pick and place*.

Se diseñó una interfaz de usuario en Unity que permite establecer una conexión local con el robot *UR3e*. Esta interfaz cuenta con un escenario virtual en el cual el usuario puede desplazarse y observar el gemelo digital del robot desde diferentes ángulos. Dentro del modo de control articular, se cuenta con seis deslizadores que representan las seis articulaciones del robot y permiten un movimiento rápido. Para el control cartesiano, el usuario puede ingresar las coordenadas en las que desea que se encuentre el efector final del robot, así como crear una lista de coordenadas. Uno de los aspectos más destacados de esta interfaz es el modo de control por trayectoria libre, el cual permite al usuario crear visualmente una ruta deformando una línea, lo que facilita la definición de la trayectoria que seguirá el robot. Además, estas rutas pueden guardarse localmente para su uso o modificación en sesiones posteriores.

Dentro del proyecto, se integraron dos elementos esenciales: RosSharp y Rosbridge.

Estos componentes jugaron un papel fundamental al posibilitar una interconexión eficiente entre Unity y ROS. Tanto RosSharp como Rosbridge resultaron herramientas imprescindibles para facilitar una comunicación fluida entre ROS y otros programas a través del envío de mensajes estructurados. Es importante enfatizar que Rosbridge se destacó por establecer una conexión sólida mediante WebSocket, asegurando una comunicación confiable y bidireccional entre la plataforma y el robot. Por otro lado, RosSharp no solo proporcionó soporte de comunicación, sino que también facilitó la importación de modelos URDF a Unity, lo que permitió una representación digital precisa del manipulador *UR3e* y la pinza granular, así como su simulación en tiempo real dentro del entorno de desarrollo de Unity.

A partir de las pruebas realizadas, se logró llevar a cabo de manera eficiente la manipulación de objetos con diversas geometrías, trasladándolos desde un punto inicial hasta un punto final. Este éxito fue alcanzado mediante la definición de las poses o posiciones que el robot debía adoptar para ejecutar la trayectoria, aprovechando los diferentes modos que la plataforma ofrecía. Además, se logró sortear obstáculos preestablecidos de forma rápida y sencilla mediante la modificación de la trayectoria visual del robot, sin necesidad de conocer ubicaciones o valores numéricos específicos, sino simplemente ajustando la línea en el simulador.

Este trabajo se enmarca dentro del ámbito de la robótica y busca brindar un importante aporte a esta disciplina. El propósito principal ha sido la construcción y desarrollo de un prototipo innovador, cuyo objetivo fundamental es servir como punto de partida y referencia para futuros investigadores. La idea central es que este prototipo sea utilizado como una herramienta sólida y confiable, sobre la cual se puedan desarrollar y explorar nuevas aplicaciones en el campo de la robótica. Al proporcionar este prototipo como base, se pretende estimular y fomentar la investigación y el avance en la robótica, permitiendo a los investigadores y desarrolladores centrarse en la exploración de nuevas funcionalidades y aplicaciones, en lugar de invertir tiempo y recursos en la construcción inicial de un prototipo desde cero.

5.2. Desarrollos futuros

Considerando el propósito fundamental de este proyecto, como el desarrollo de una plataforma software para el control de un manipulador de manera sencilla sin requerir grandes conocimientos en robótica, se han evidenciado múltiples aplicaciones para esta plataforma, así como posibles mejoras que se podrían incorporar en el futuro. Como resultado, se plantean las siguientes tareas o proyectos futuros:

- Evaluar la precisión de las trayectorias generadas por medio de sistemas sensoriales y cámaras en entornos reales.
- Incorporar la ubicación precisa de objetos virtuales con objetos que se encuentran en el entrono real.
- Realizar trayectorias quirúrgicas para espacios de trabajo reducido, por ejemplo para operaciones a través de la cavidad endonasal.
- Integrar el control de la pinza de interferencia granular aplicando un lazo de control cerrado.
- Implementar la comunicación de varios manipuladores para aplicaciones que requieran el control de más de un robot.
- Implementar una comunicación remota entre la plataforma y el robot mediante ROS para un telecontrol con fines educativos.

A. Anexos

A.1. Guía de la plataforma

A.1.1. Requisitos previos e instalación

Esta sección muestra el contenido del proyecto construido en Unity. [1]

Anexo A.1.: Plataforma construída en Unity3D.

Para trabajar sobre el proyecto es importante tener en cuenta los siguientes requisitos:

- Ubuntu 18.04 +
- ROS Melodic +

[1] https://github.com/sebastian775/UR3Project

- URSim o Robot real
- Unity 2021.3.0f1 +

Importante: Establecer conexión por medio de un cable Ethernet para evitar la perdida de datos. Así mismo es fundamental contar con un espacio de trabajo limpio y organizado (*catkin_ws*), de lo contrario, pueden generarse conflictos al momento de compilar este *catkin_ws*.

Los siguientes comandos se ejecutan consecutivamente en un único terminal, lo que le permitirá crear el espacio de trabajo y también instalar paquetes y unidades necesarias para manipular los robots UR.

```
# source global ros
$ source /opt/ros/melodic/setup.bash

# create a catkin workspace
$ mkdir UR3e && cd UR3e && mkdir -p catkin_ws/src && cd catkin_ws

# clone the driver
$ git clone https://github.com/UniversalRobots/
   Universal_Robots_ROS_Driver.git src/Universal_Robots_ROS_Driver

# clone the description. Currently, it is necessary to use the melodic
   -devel-staging branch.
$ git clone -b melodic-devel-staging https://github.com/ros-industrial
   /universal_robot.git src/universal_robot

# clone  the ur control cartesian
$ git clone https://github.com/UniversalRobots/
   Universal_Robots_ROS_controllers_cartesian.git src/
   Universal_Robots_ROS_controllers_cartesian

# install dependencies
$ sudo apt update -qq
$ rosdep update
$ rosdep install --from-paths src --ignore-src -y

# build the workspace
$ catkin_make

# activate the workspace (ie: source it)
```

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

Instalación de RosBridge

Rosbridge proporciona una API JSON para comunicar programas con ROS.

```
$ sudo apt-get install ros-melodic-rosbridge-server
```

Descarga del ejecutable

El siguiente ejecutable es construido en Unity 3D, y se puede descargar de:
https://github.com/sebastian775/UR3Project/releases/tag/v7

A.1.2. Ejecución del proyecto

Es importante tener en cuenta que el simulador debe estar inicializado con su configuración correspondiente, tal como se menciona en (Anexo A.2).

Desde una terminal ejecutar lo siguiente:

```
# Se inicializa el controlador del robot
$ cd cd UR3e/catkin_ws
$ source devel/setup.bash
# Esto ejecuta el controlador del robot
$ roslaunch ur_robot_driver ur3e_bringup.launch robot_ip:=<YOUR_IP>
   limited:=true
```

En una nueva terminal ejecutar:

```
$ roslaunch rosbridge_server rosbridge_websocket.launch
```

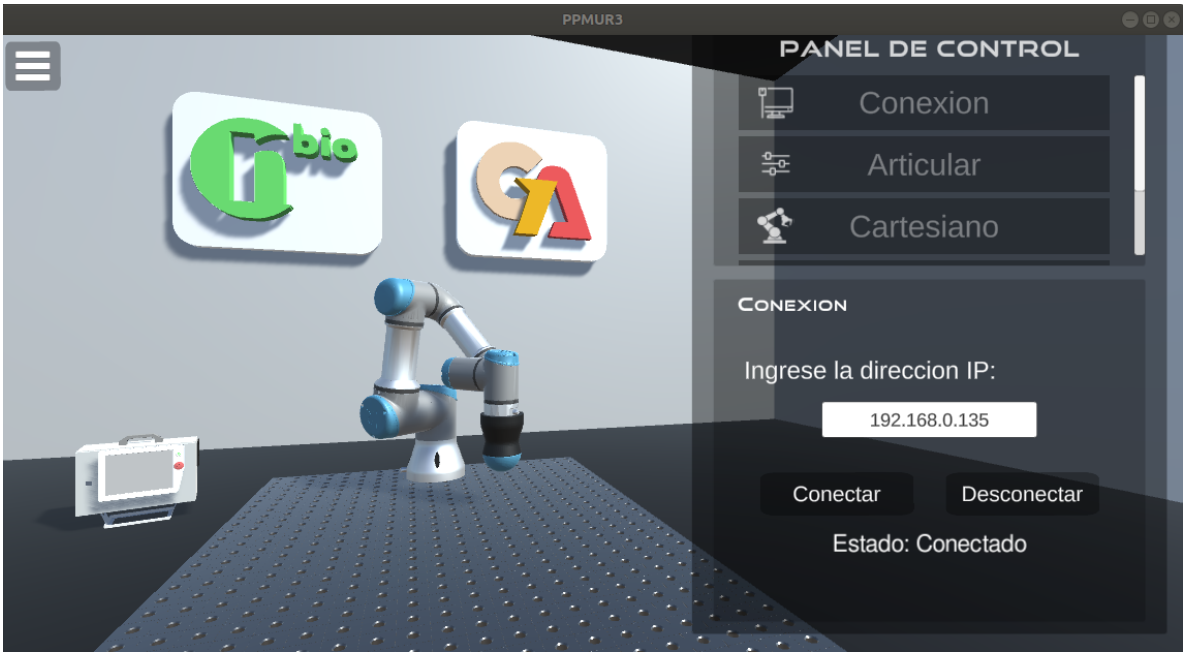
En la carpeta Interfaz_v7 que se descarga se encuentra un archivo de python llamado "*ListarDatos.py*", se debe abrir una terminal en esta dirección y ejecutar este *script* con el siguiente comando:

NOTA: Se requiere contar con la versión de python3.

```
$ python3 ListarDatos.py
```

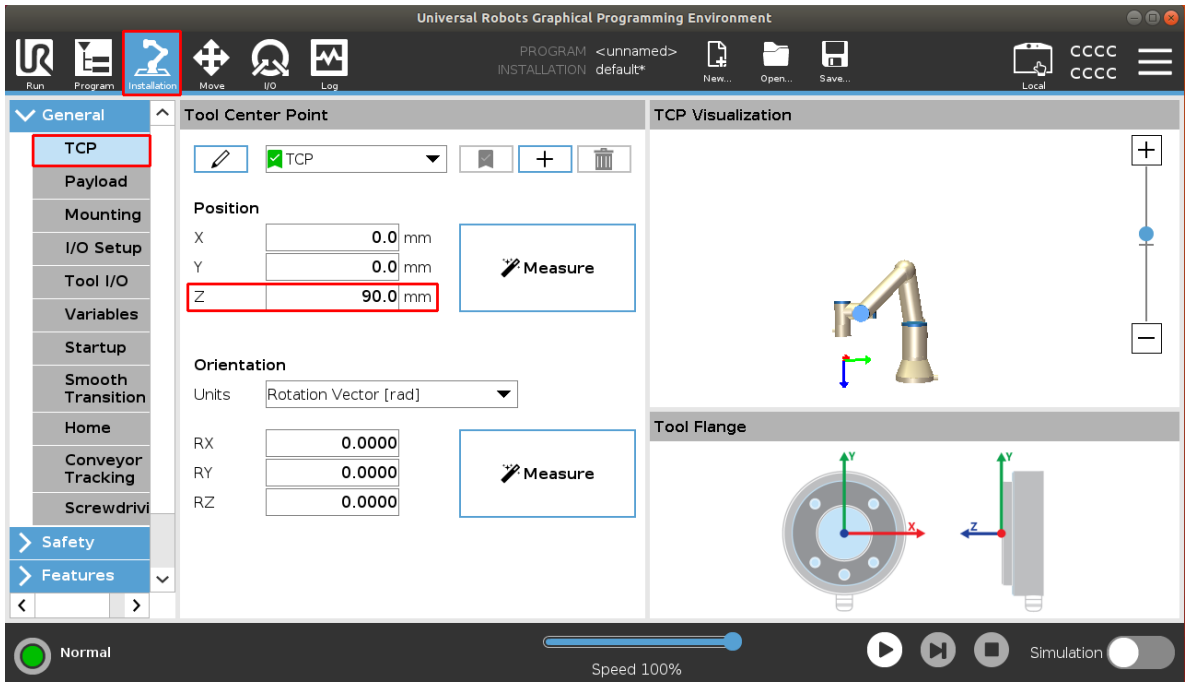
Ahora ejecutar el archivo llamado "*PPMUR3.x86_64*" contenido en esta carpeta (dar doble clic sobre el ícono). Una vez abierto, aparecerá el escenario con el robot digital y la interfaz principal donde se debe seleccionar la opción de conexión para que despliegue el panel de conexión. En el cuadro de texto ingresar la dirección IP del *host* que esté ejecutando ROS, y presionar en el botón de conectar. Una vez hecho esto el estado cambiara a "*Conectado*" y el gemelo digital tomará la posición del robot (Figura A.2). Luego de esto se puede mover el robot seleccionando alguno de los modos de control de la plataforma.

NOTA: Para que el robot del simulador o el robot real se mueva, se debe habilitar el movimiento desde el botón de *play* de la interfaz de *Polyscope*.



Anexo A.2.: Conexión de la plataforma desde el ejecutable.

Adicionalmente se debe configurar el punto central de la pinza (TCP) desde la ventana de *instalation* de *Polyscope*. Para el caso de la pinza de interferencia granular se tiene una desviación en la posición de 90 *mm* en el eje z (Figura A.3).



Anexo A.3.: TCP para la pinza de interferencia granular.

Atajos para modificar la vista de la escena

Cuando se está dentro de la plataforma, se pueden utilizar los atajos de teclado de la Figura A.4 para moverse a través del laboratorio virtual o cambiar la vista de la cámara en las vistas ya predeterminadas.

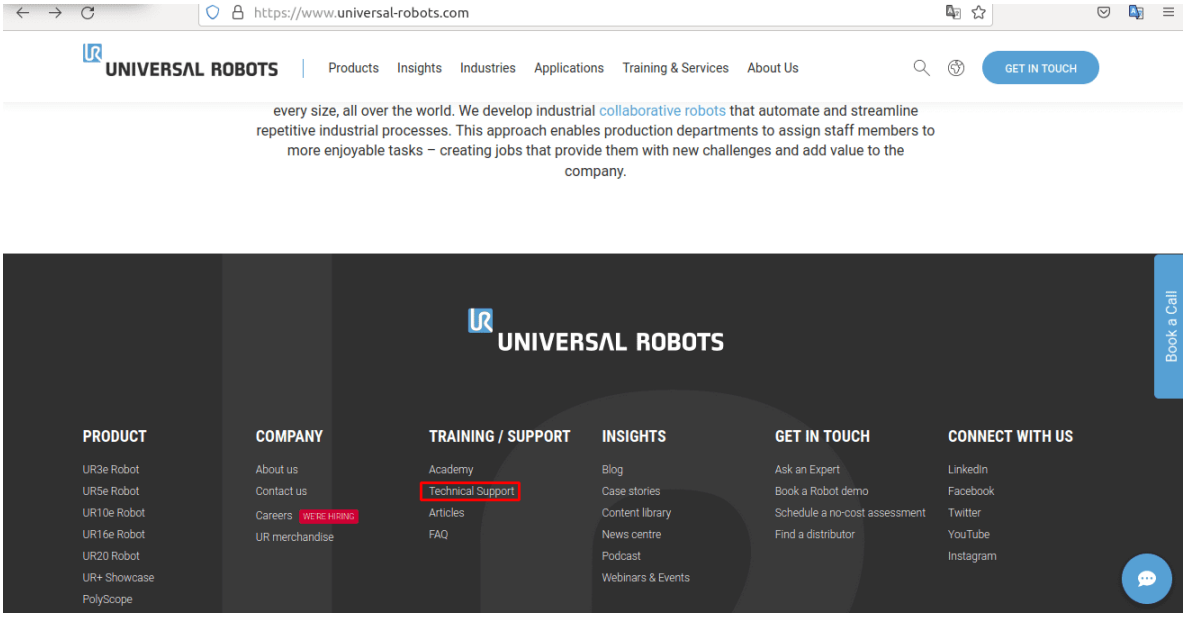
NOTA: Los atajos de teclado se habilitan mientras el cursor del ratón no está sobre el panel principal.

Anexo A.4.: Atajos de teclado.

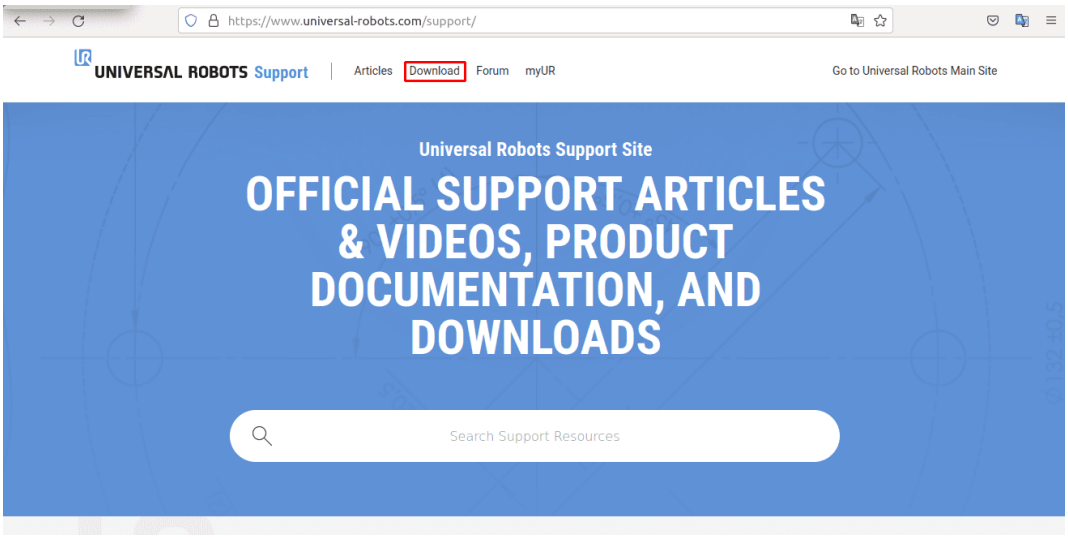
A.2. Guia de instalación de URSim

Es importante tener en cuenta que la instalación del simulador URSim puede provocar la desinstalación de ROS. Por esta razón, se recomienda seguir los pasos de esta guía antes de proceder a instalar ROS.

Para comenzar, es necesario descargar el simulador desde la página oficial de Universal Robots. Para ello, debe desplazarse hasta el final de la página y seleccionar la sección ***Training / Support*** (Figura A.5), seguido de ***Technical Support***. Esto le redirigirá a una nueva página, donde encontrará el botón ***Download*** ubicado en el *banner* superior (Figura A.6).

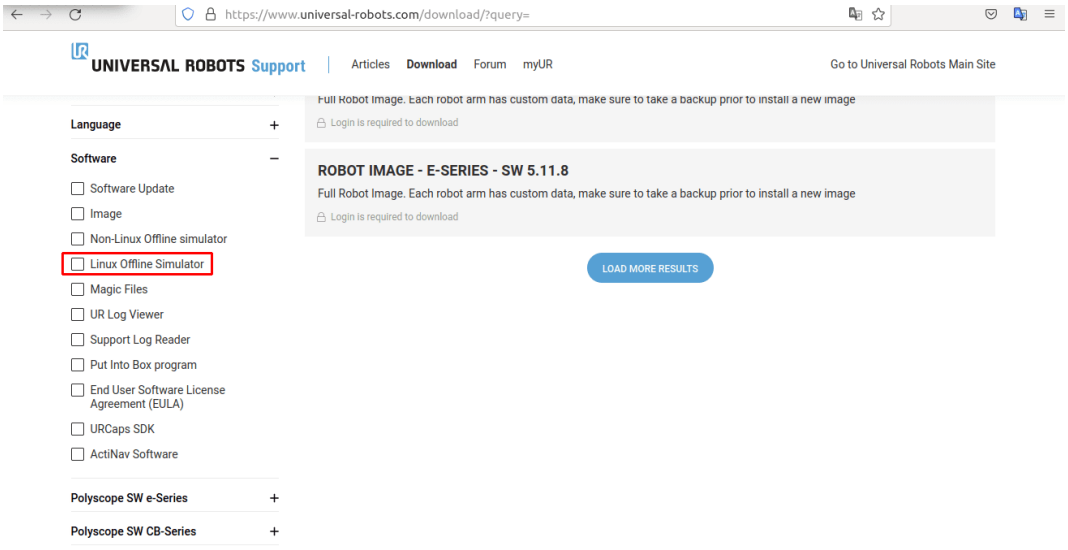


Anexo A.5.: *Training / Support.*

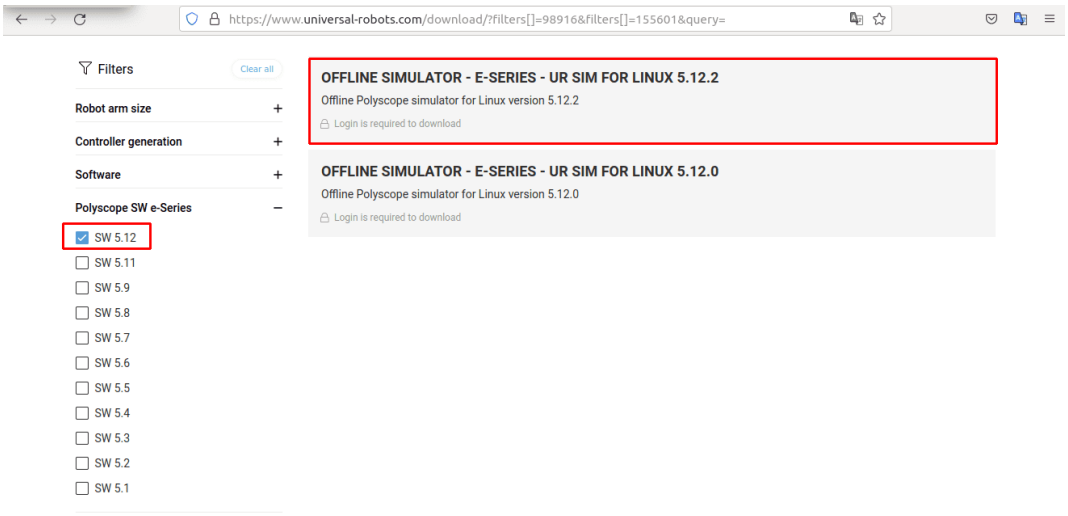


Anexo A.6.: *Download*.

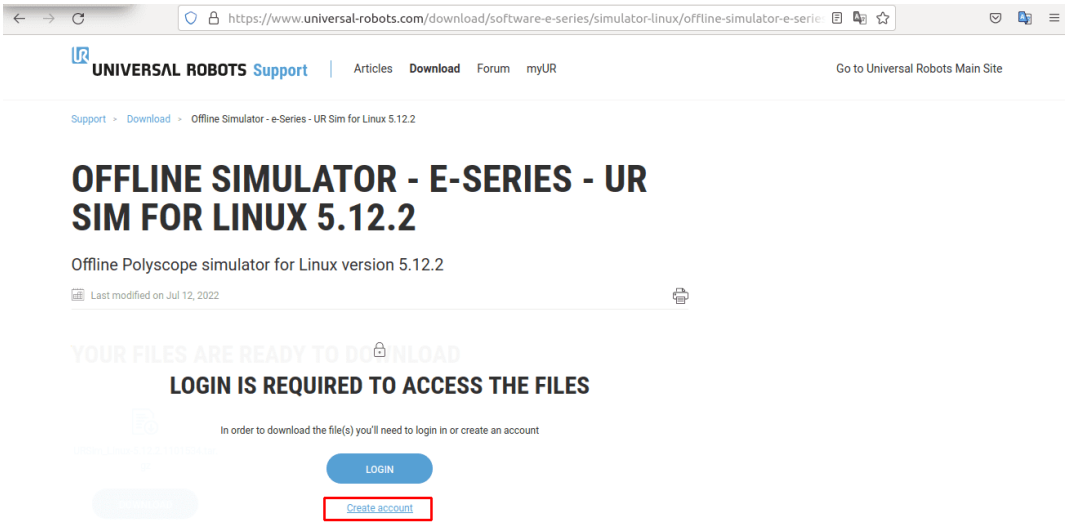
Al hacer clic en el botón **Descargar**, se abre una nueva página en la que aparecen los filtros de búsqueda en el lateral izquierdo. Aquí, se debe seleccionar la opción ***Software*** y marcar **Linux Offline Simulator**. Además, es necesario elegir el tipo de robot con el que se va a trabajar, como ***e-series*** o **CB-series**. En este ejemplo, se instalará la versión para **e-series**. Cabe destacar que los pasos son similares e intuitivos, independientemente de la serie que se utilice. Finalmente, es necesario crear una cuenta de usuario para poder descargar el simulador (Figura A.7) (Figura A.8) (Figura A.9).



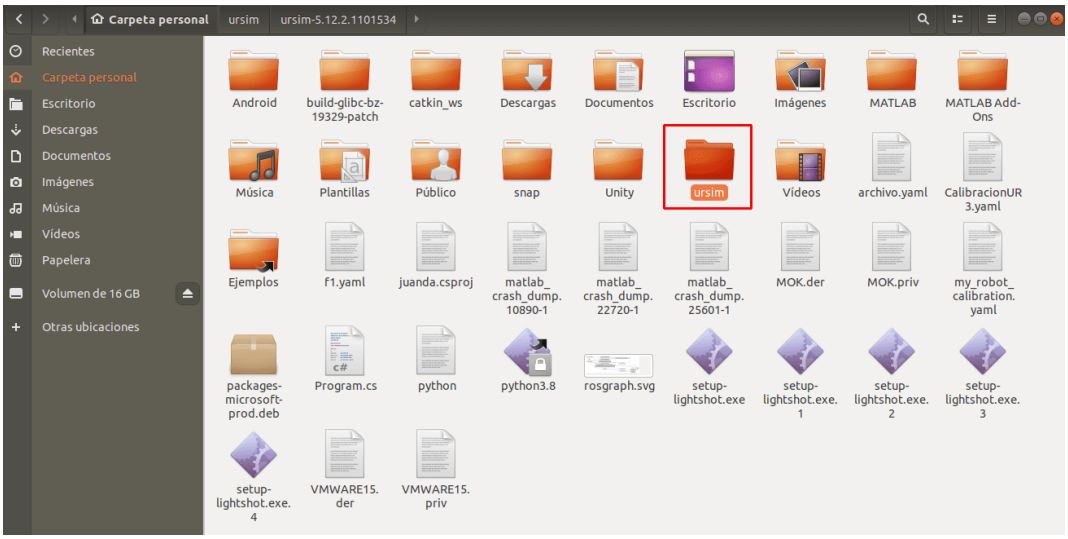
Anexo A.7.: *Linux Offline Simulator*.



Anexo A.8.: SW 5:12.

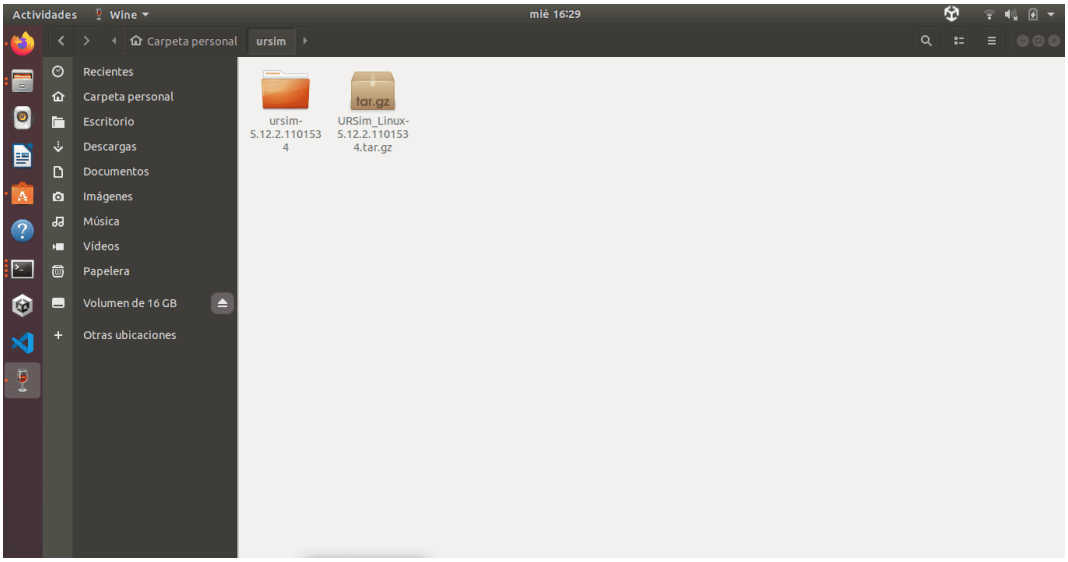


Anexo A.9.: *Create account.*



Anexo A.10.: Directorio URSim.

Después de haber descargado el archivo, se sugiere crear una carpeta llamada *ursim* en la ubicación **/carpeta personal** y copiar allí el archivo descargado.



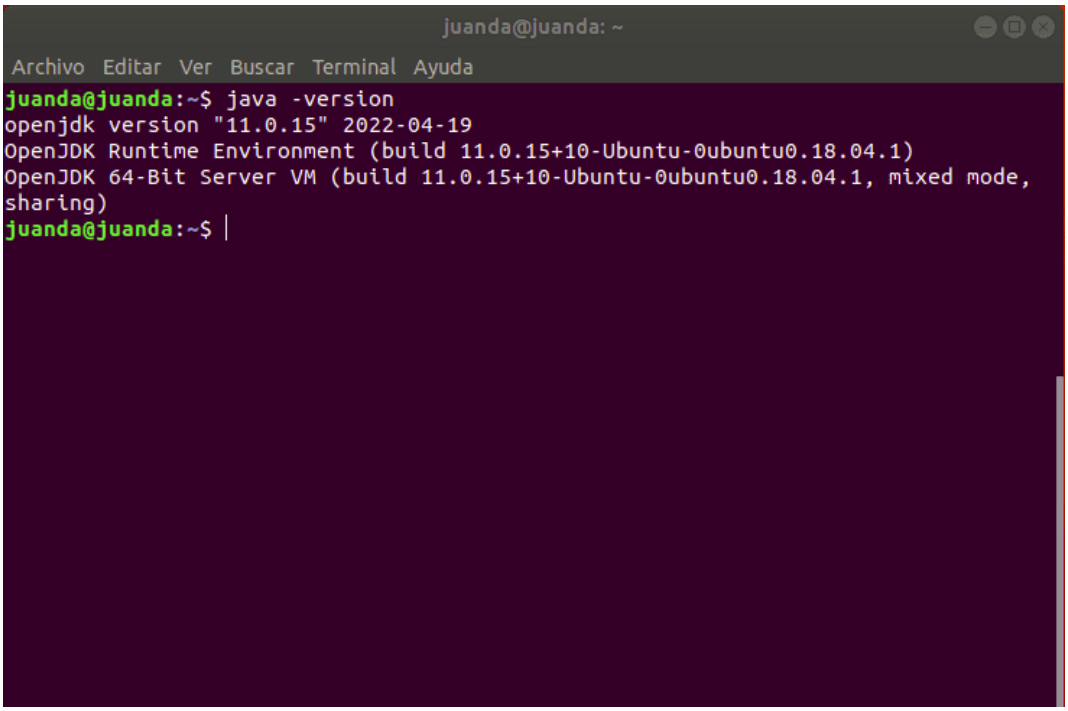
Anexo A.11.: Ruta URSim.

NOTA: Antes de instalar el software se deberá verificar la **versión del SDK** con el siguiente comando en una nueva terminal:

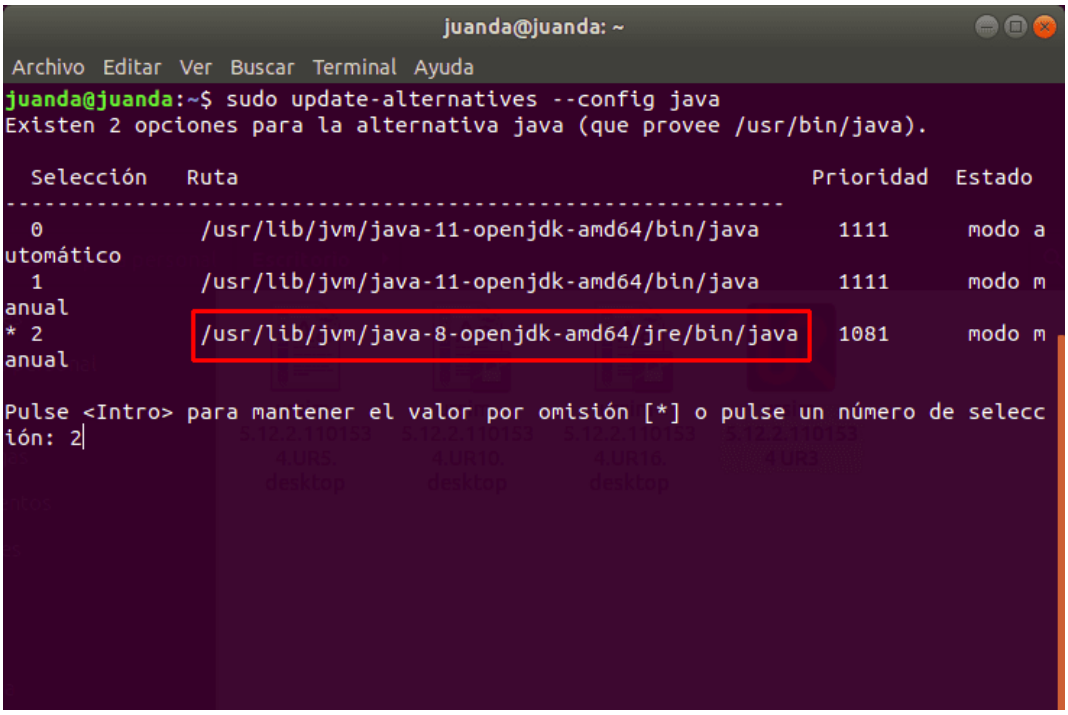
```
$ java -version
```

Para garantizar el correcto funcionamiento del programa, es necesario reemplazar su versión actual por **openjdk version 1.8.0**. Para hacerlo, se pueden utilizar los siguientes comandos:

```
# Comando para instalar jdk v8
$ sudo apt install openjdk-8-jdk
# Comando para cambiar version jdk
$ sudo update-alternatives --config java
```



Anexo A.12.: *Java version.*

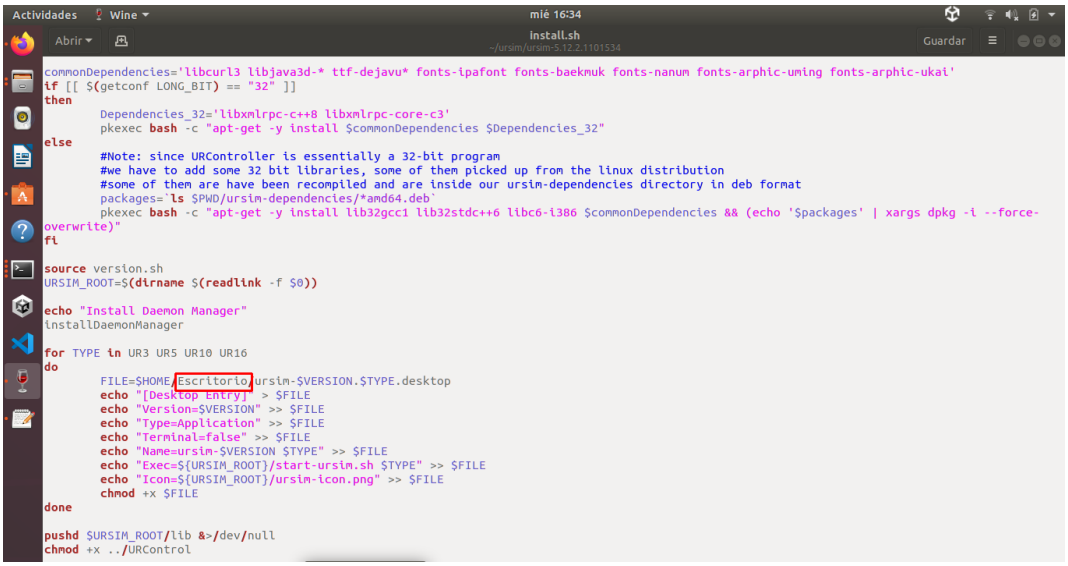


Anexo A.13.: *Alternatives version.*

Después de verificar la versión del SDK, diríjase a la carpeta del simulador, tal como se recomendó previamente **/carpeta personal**, abra una terminal y descomprima el archivo utilizando los siguientes comandos.

```
# tar xvzf [File_name.tar.gz]
$ tar xvzf URSim_Linux-5.12.2.1101534.tar.gz
$ cd ursim-5.12.2.1101534
```

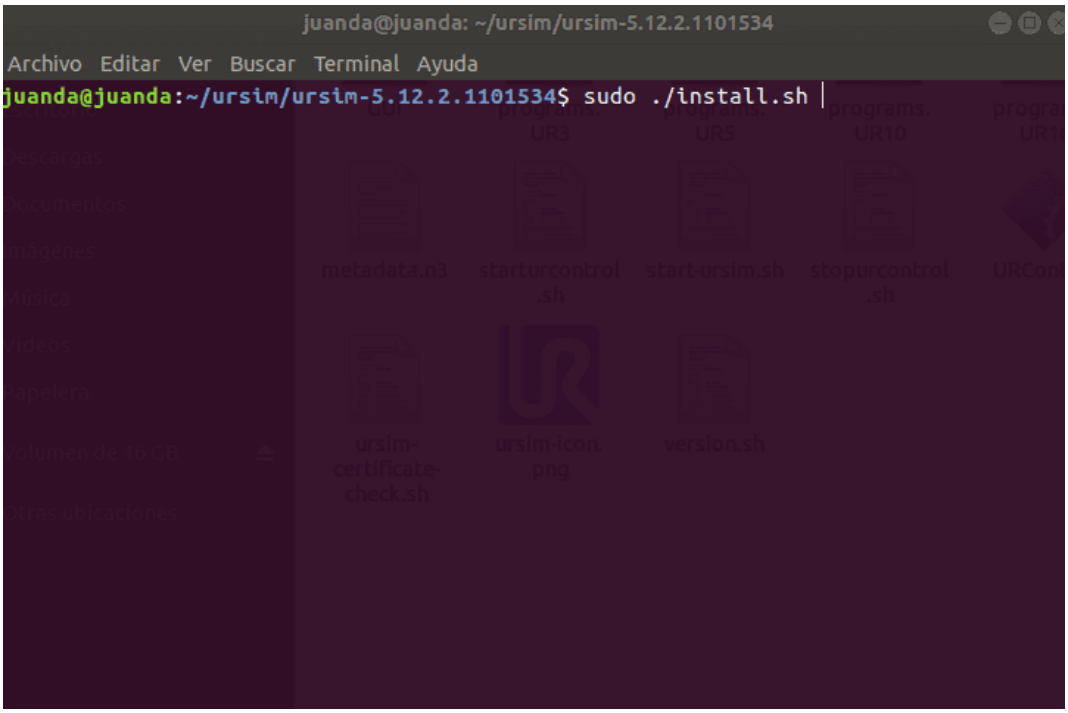
Nota: Si la versión de Ubuntu esta en inglés omitir la siguiente recomendación, de lo contrario deberá dirigirse a la carpeta que descomprimió y en un editor de texto abrir el archivo llamado ***install.sh*** y modificar la palabra ***Desktop*** por **Escritorio**, como muestra la siguiente imagen.



Anexo A.14.: Modificación en el archivo *install.sh*.

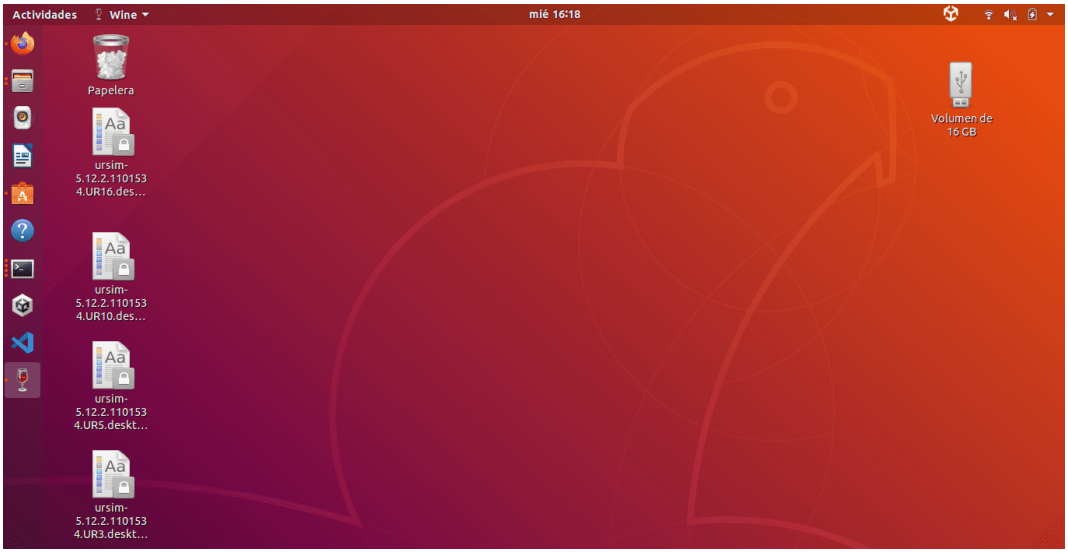
Posteriormente, se ejecuta el siguiente comando en la terminal:

```
$ sudo ./install.sh
```



Anexo A.15.: Instalación del simulador.

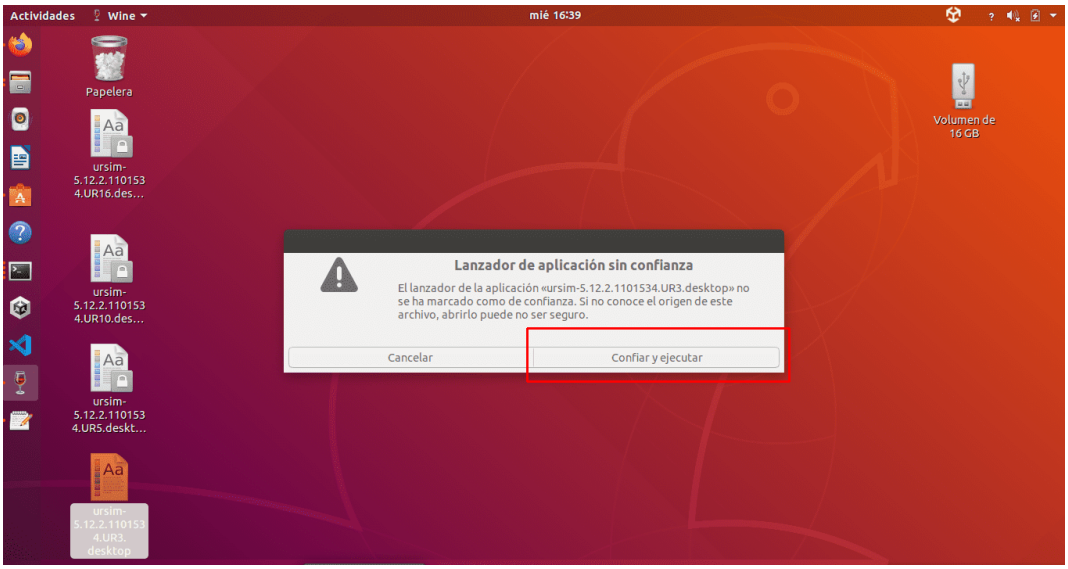
La acción anterior permitirá visualizar en el escritorio los accesos directos correspondientes a cada versión del simulador. Si los íconos mostrados son similares a los que se presentan en las siguientes imagenes, será necesario ejecutar el siguiente comando:



Anexo A.16.: Vista de íconos de URSim en el escritorio.

```
$ sudo chown <user>/home/user/Escritorio/<nombre archivo>.desktop
```

Después de eso, se selecciona el acceso directo correspondiente a la versión del robot que se desea ejecutar, otorgándole permisos de confianza. De esta manera, el ícono cambiará para indicar que cuenta con los permisos necesarios y se podrá entonces ejecutar el simulador.



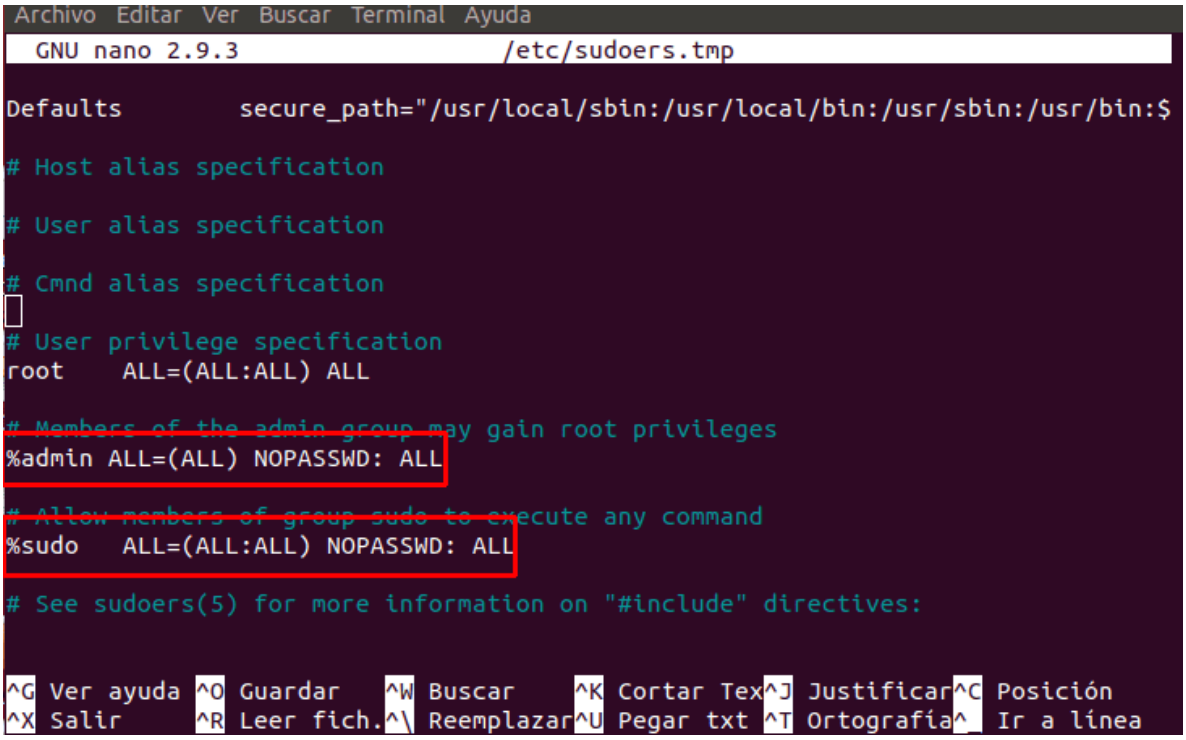
Anexo A.17.: Lanzador de aplicaciones desconocidas.

Anexo A.18.: Aplicación con permiso de ejecución.

También es necesario agregar permisos de ejecución para poder iniciar el simulador desde los íconos del escritorio. Para ello, se debe ejecutar el siguiente comando en una terminal:

```
$ sudo visudo
```

Es necesario agregar la palabra ***"NOPASSWD"*** a las líneas de comando, tal como se muestra en la siguiente imagen:

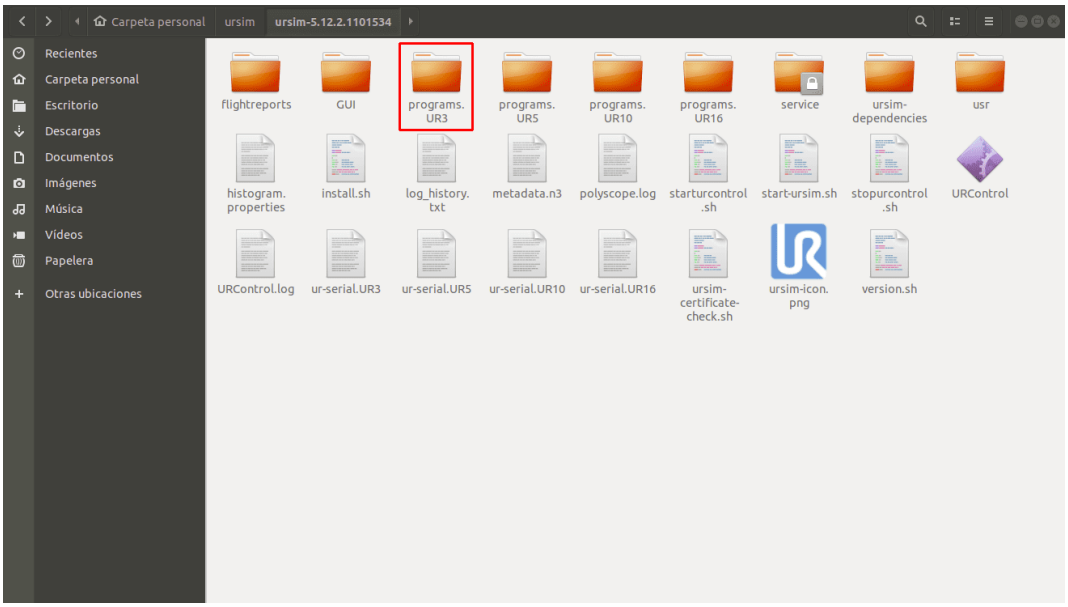


Anexo A.19.: Configuración de permisos de usuario.

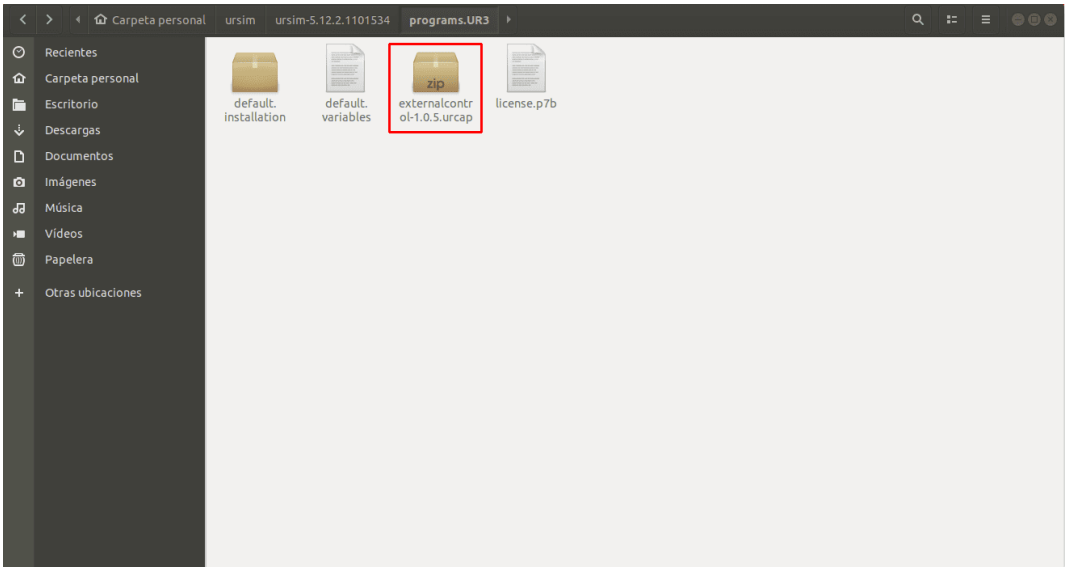
Nota: Si se tienen problemas para ejecutar el programa desde los iconos o simplemente se opta por no hacerlo de esa manera, se puede dirigir a la carpeta donde se encuentra el ejecutable y escribir el siguiente comando:

```
$ sudo ./start-ursim.sh <Robot_Version>
# Ejemplo: sudo ./start-ursim.sh UR3
```

Antes de iniciar la aplicación, se requiere descargar el *plugin* **externalcontrol-1.0.5.urcap** y pegarlo en la carpeta correspondiente a la versión del robot en uso, por ejemplo, **program.UR3** dentro de la carpeta **program.versionRobot**.

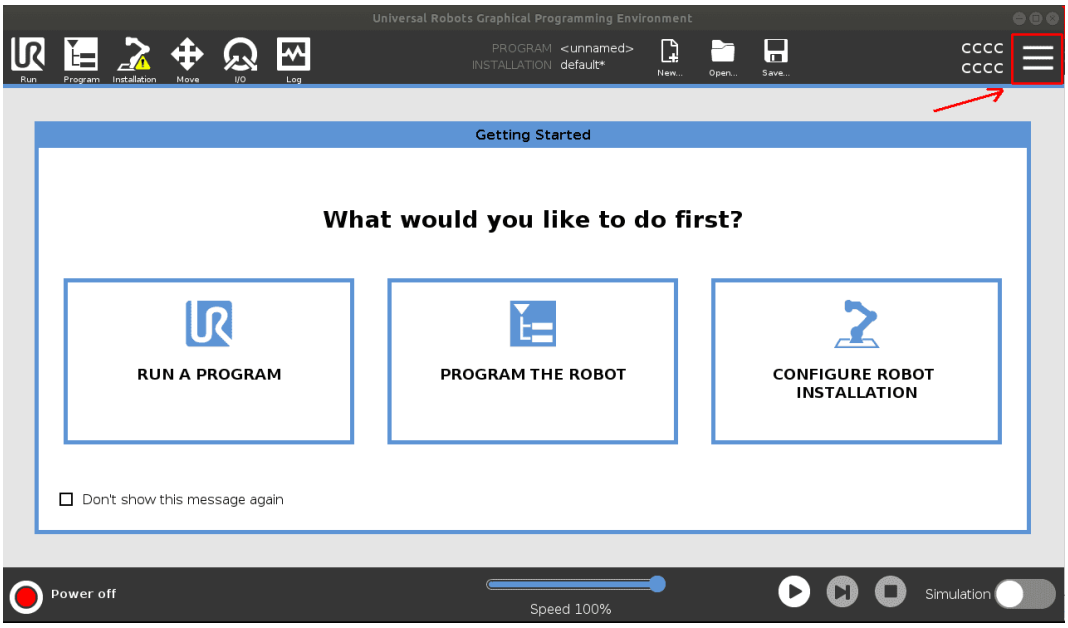


Anexo A.20.: Directorio de UR3.

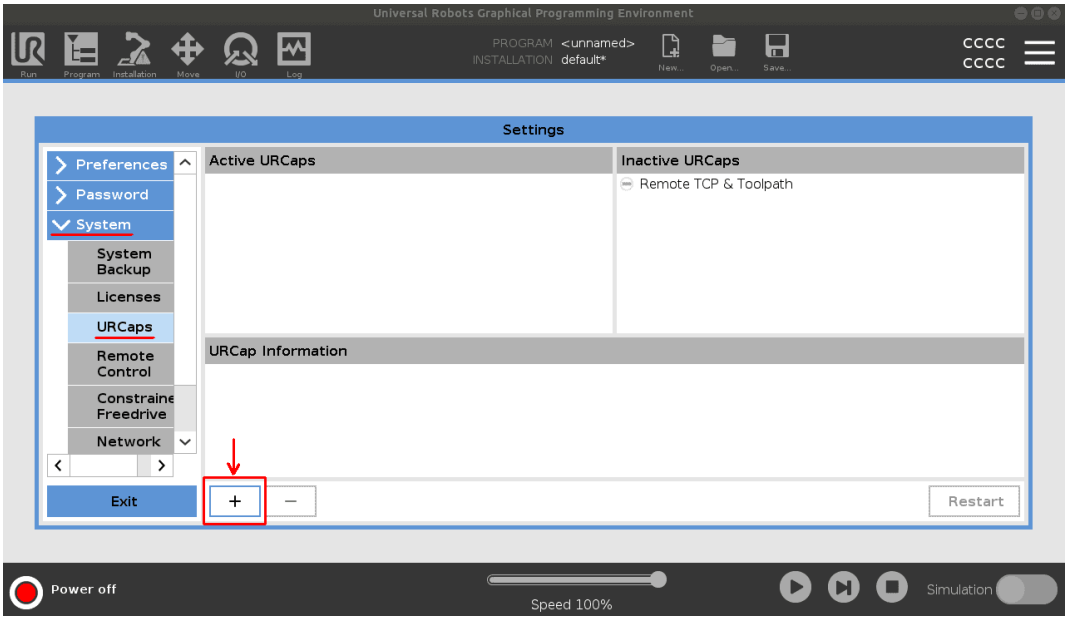


Anexo A.21.: Archivo external control.

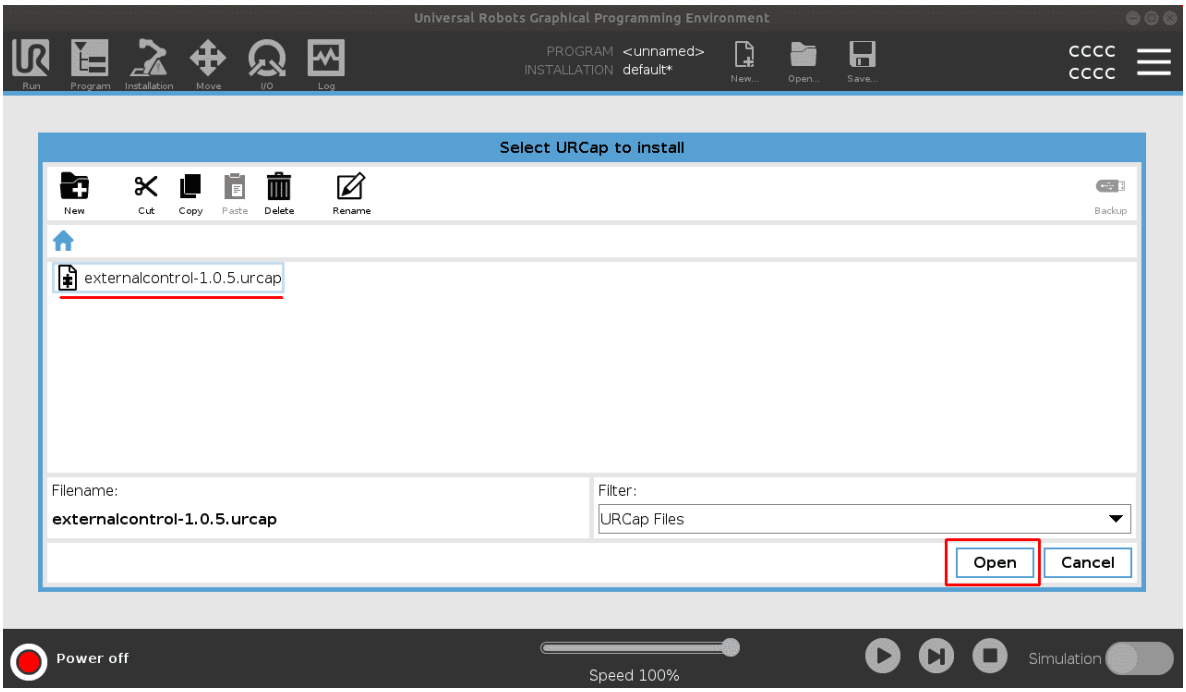
Una vez descargado el *plugin external control*, se puede ejecutar el simulador y proceder a su configuración. Para ello, deberá ingresarse la configuración del simulador y seguirse los pasos que se muestran en las siguientes imágenes con el fin de agregar el mencionado *plugin*.



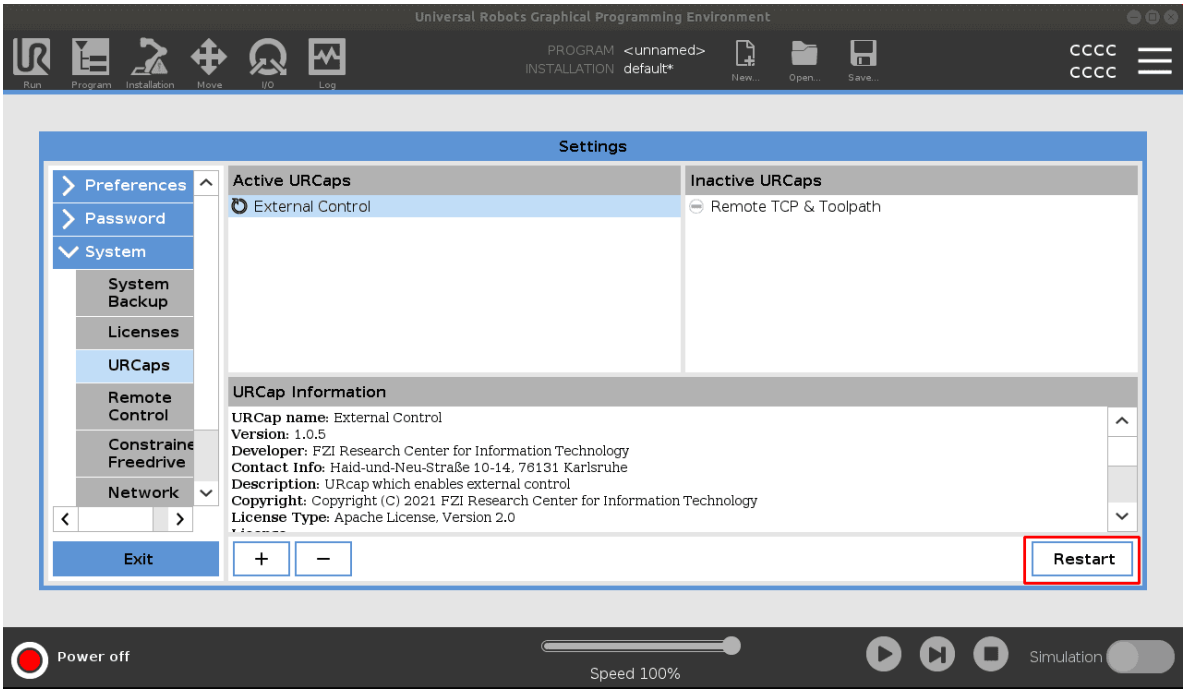
Anexo A.22.: Vista de inicio de URSim.



Anexo A.23.: Ubicación URCap *external control*.



Anexo A.24.: Importar URCap *external control*.

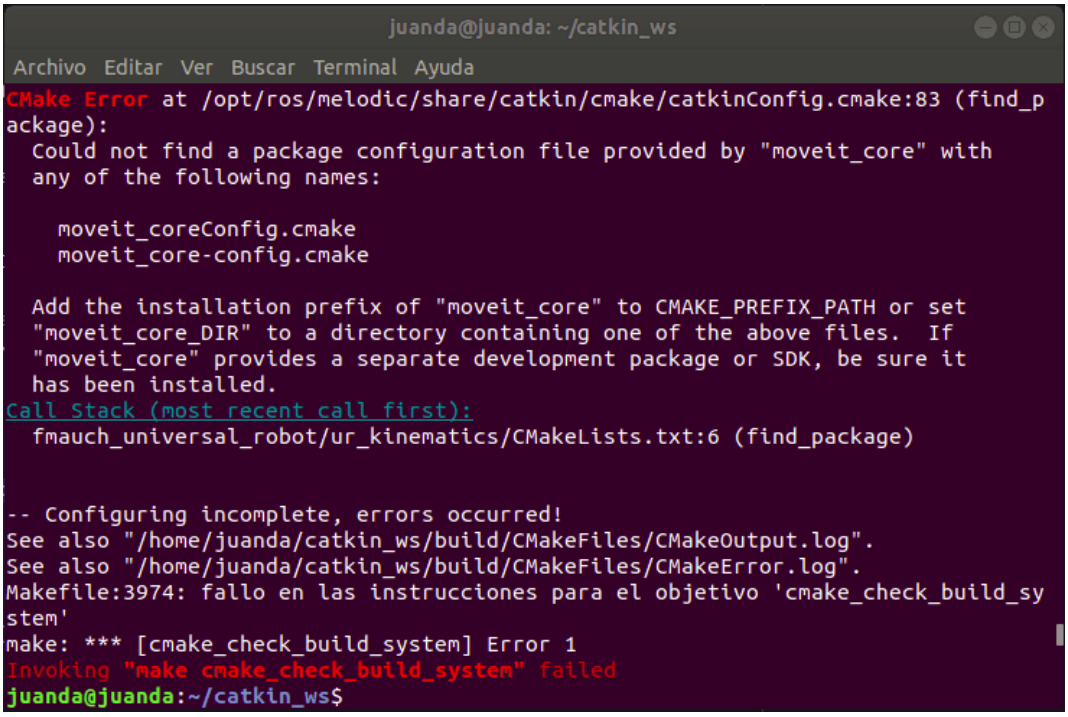


Anexo A.25.: *External control* instalado.

Una vez llegado a este punto, se puede proceder a la instalación de ROS, seleccionando

la versión compatible con su sistema operativo y siguiendo los pasos indicados en la página oficial: http://wiki.ros.org/es/ROS/Installation

Es importante tener en cuenta que si después de realizar la instalación de ROS y ejecutar el comando **$ catkin_make** en el espacio de trabajo "*catkin_ws*", se presentan problemas similares a los mostrados en la imagen, es posible que falte alguna librería. En este caso, puede utilizar los siguientes comandos para solucionar el problema:



Anexo A.26.: Posibles fallos en el catkin.

```
$ sudo apt update -qq
$ rosdep update
$ rosdep install --from-paths src --ignore-src -y
```

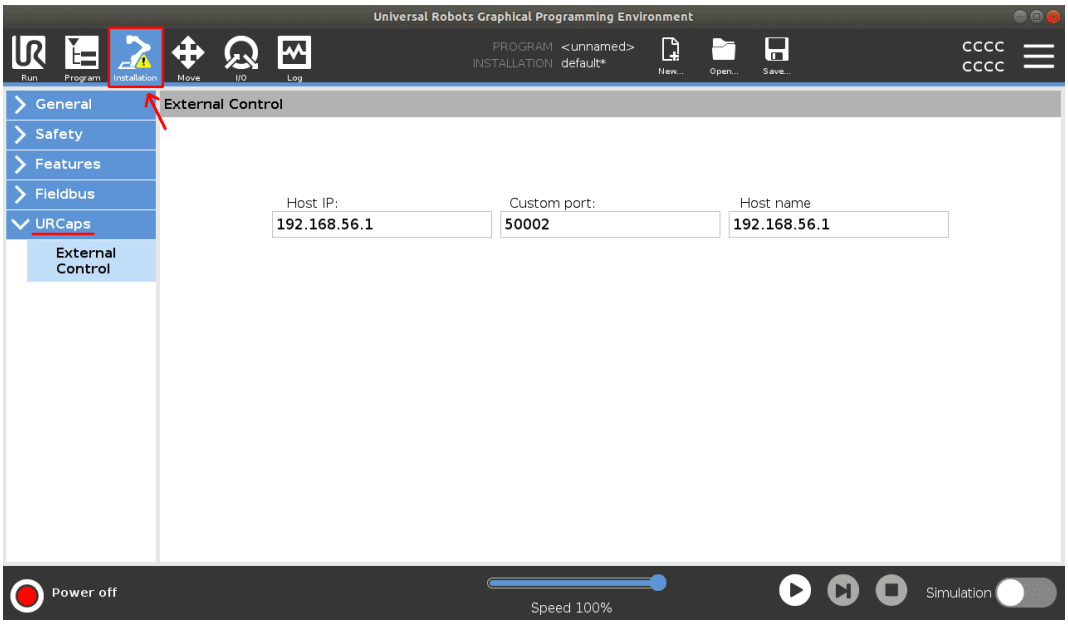
Si el error persiste utilice el siguiente comando:

```
$ sudo apt install ros-melodic-PACKAGE
```

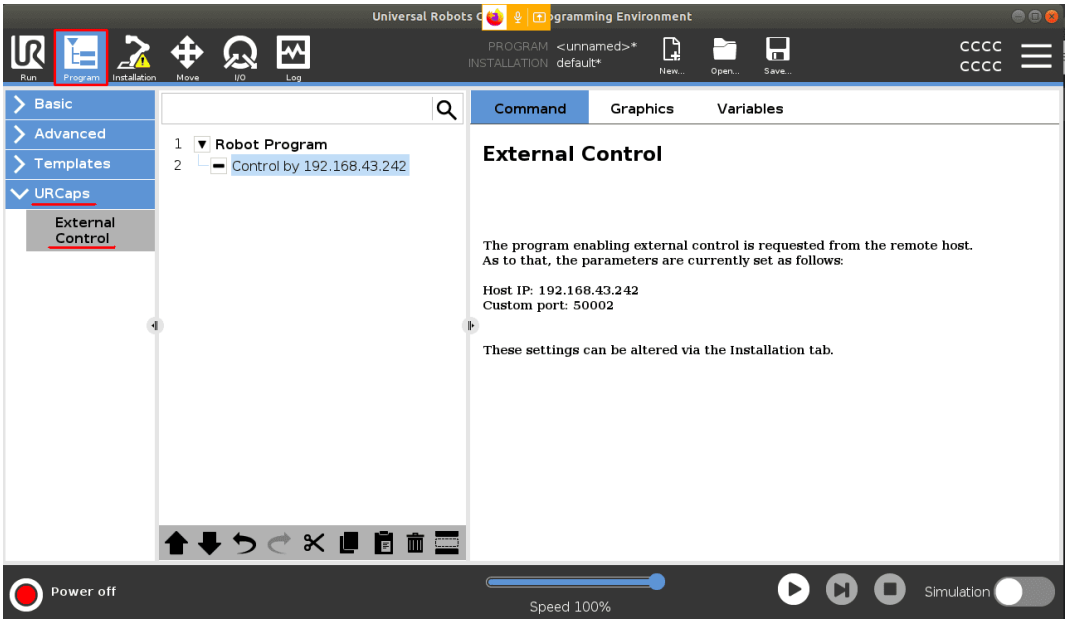
por ejemplo:

```
$ sudo apt install ros-melodic-moveit-core
```

Una vez instalado ROS, es posible utilizar el simulador. Para ello, es necesario realizar la configuración de la dirección IP en la sección *Installation/URCaps/External Control*, y luego activarla en *Program/URCaps/External Control* **(ver las siguientes imágenes)**.

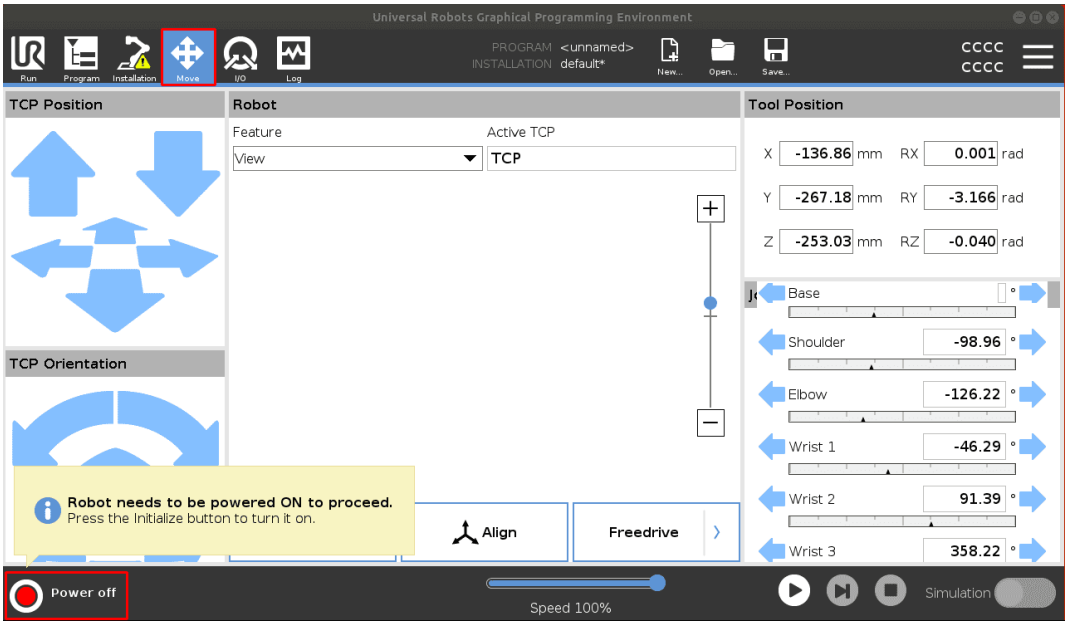


Anexo A.27.: Configuración de la IP.

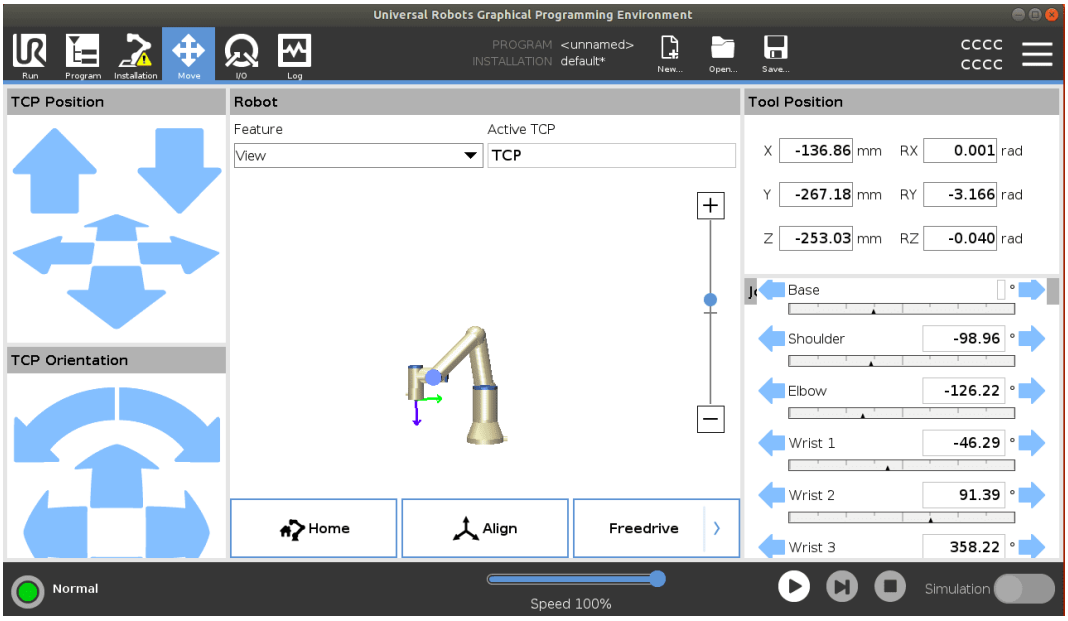


Anexo A.28.: Programar instrucción de conexión por red.

El programa ya está listo para ser utilizado. Simplemente presione el botón de encendido para activarlo y ejecutar el controlador del robot desde la terminal de comandos.



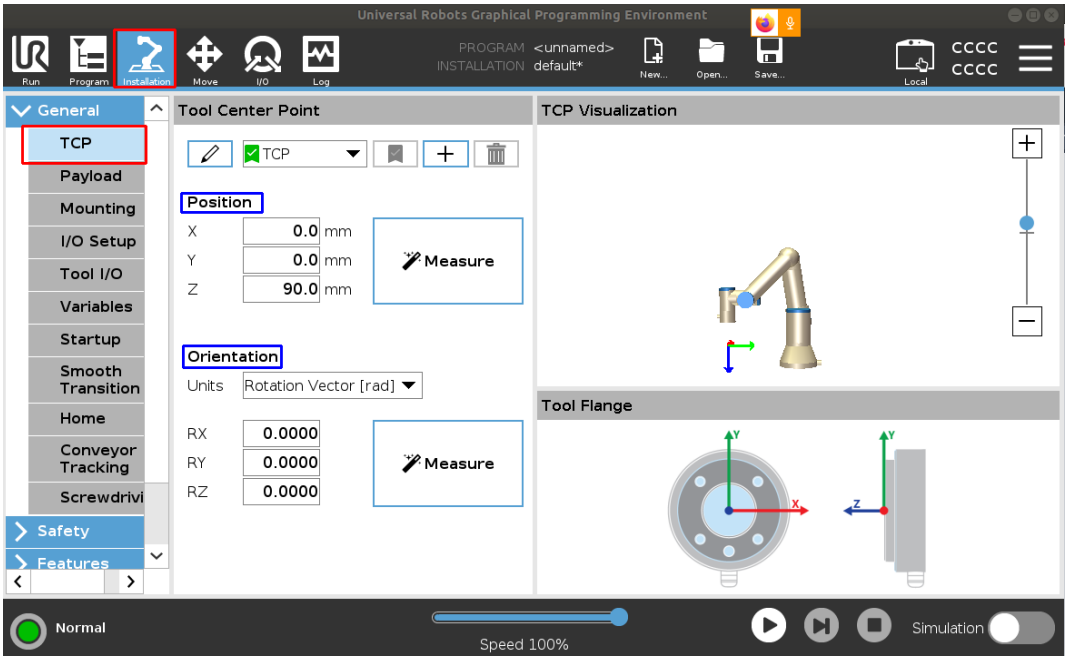
Anexo A.29.: Encender el robot.



Anexo A.30.: Configuración final del *Polyscope*.

Es importante tener en cuenta que si se desea enviar señales desde ROS el simulador debe estar en modo *PLAY* (▶).

NOTA: Si se agrega una pinza al robot, es necesario ajustar los valores de referencia del TCP (*Tool Center Point*) en el apartado de *instalation*.



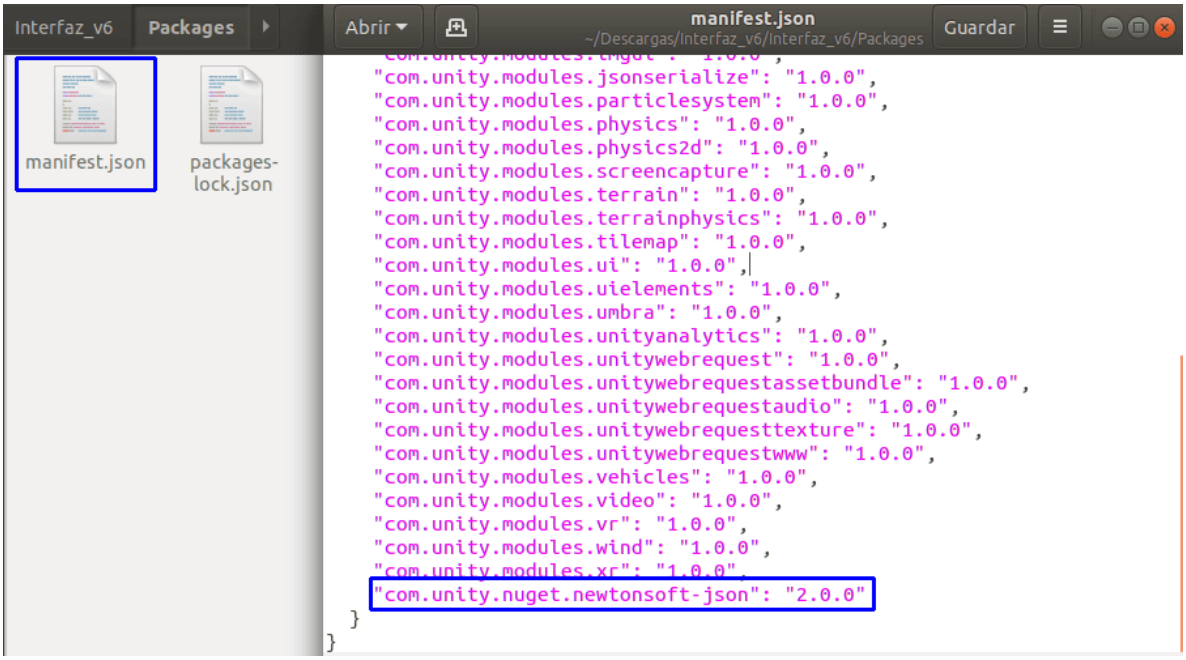
Anexo A.31.: Configuración del *Tool Center Point*.

A.3. Creación de ejecutable en Unity desde Ubuntu

Un posible problema que puede acontecer es que al crear proyectos en Unity desde Ubuntu, se desee crear un ejecutable de este y que resulten errores como:

```
ArgumentException: The Assembly Newtonsoft.Json is referenced by
    Newtonsoft.Json.Bson ('Assets/ROS/RosSharp/Plugins/External/
    Newtonsoft.Json.Bson.dll'). But the dll is not allowed to be
    included or could not be found
```

Para solucionar este problema, es necesario agregar manualmente al archivo "*manifest.json*"la siguiente línea: *"com.unity.nuget.newtonsoft-json": "2.0.0"* [56]. Unity no realiza esta operación de forma automática (posiblemente debido a que está alojado en Linux). Este archivo oculto se encuentra dentro del proyecto de Unity en la ruta "Interfaz/Packages/manifest.json".

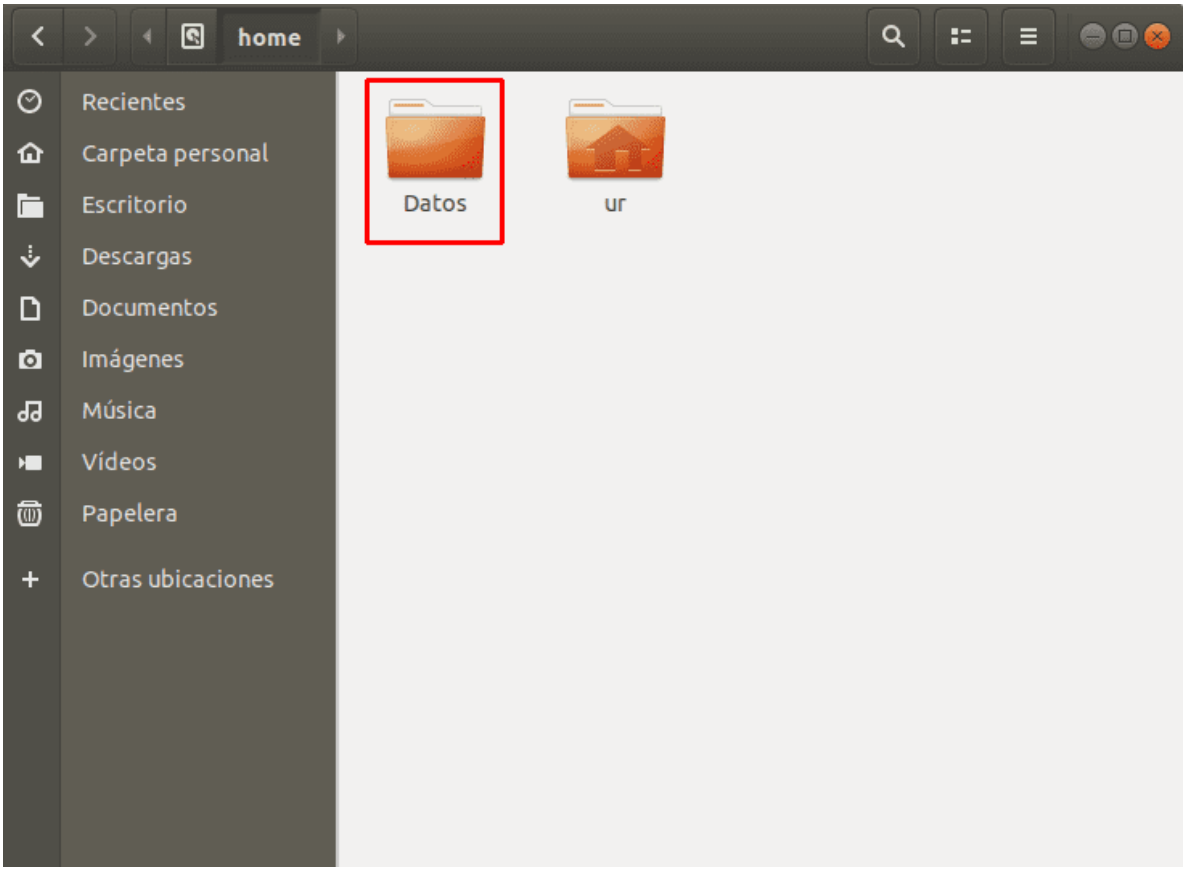


Anexo A.32.: Módulo para generar ejecutables en Unity desde Ubuntu.

A.4. Directorio con permiso de superusuario

Para crear un directorio con permisos de escritura y lectura para cualquier usuario en el sistema de Ubuntu se puede ejecutar el siguiente comando desde una nueva terminal:

```
$ cd /home && sudo mkdir -m 777 Datos
```

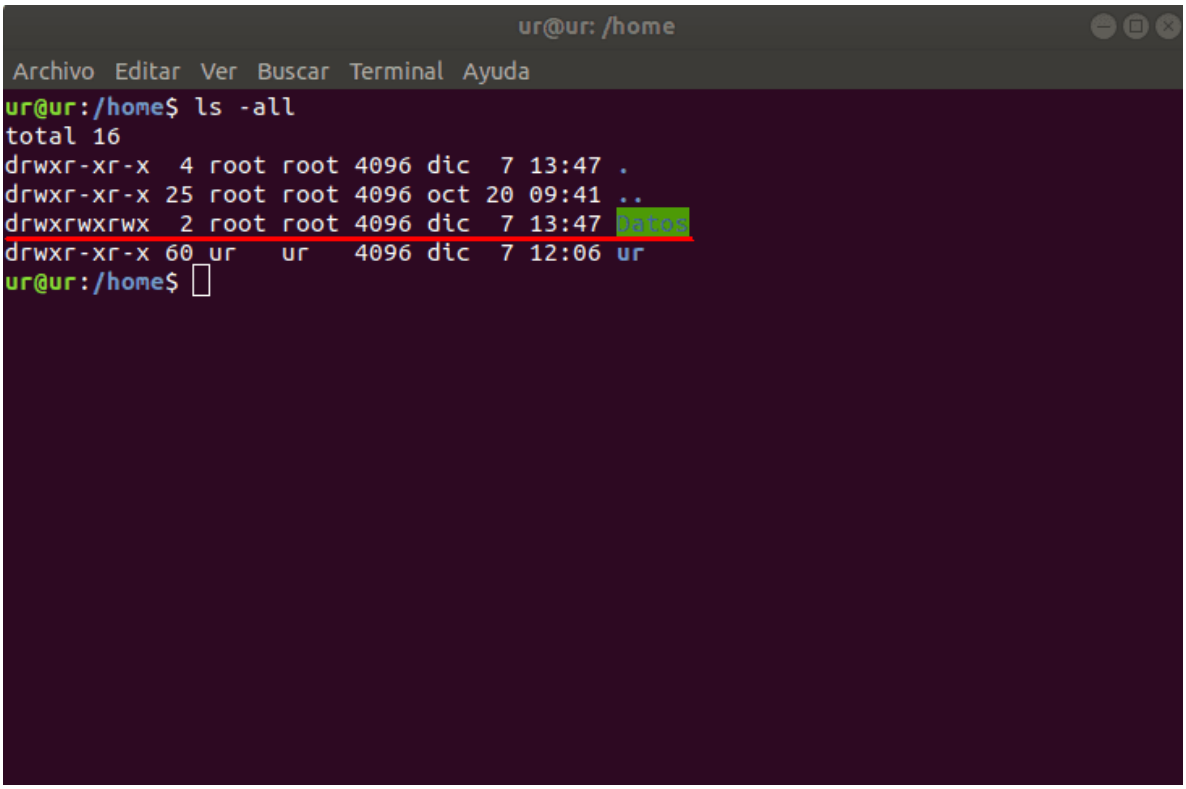


Anexo A.33.: Directorio con todos los permisos.

Para comprobar que el directorio posee todos los permisos, se ejecuta:

```
$ ls -all
```

El directorio debe presentar los permisos **"rwxrwxrwx"** que representan permisos de lectura (r, *read*), escritura (w, *write*) y ejecución (x, *excute*) para todos los usuarios.



Anexo A.34.: Permisos de directorio.

A.5. Creación e importación de modelos 3D a URDF

Esta sección se enfoca en el proceso de modelado e importación de una pinza en formato URDF utilizando *Autodesk Fusion 360* en el sistema operativo Windows 10. El objetivo es incorporar este modelo al robot en la escena de Unity. El primer paso consiste en instalar el complemento requerido para exportar modelos URDF desde *Autodesk Fusion 360*. Para lograrlo, se siguen las siguientes instrucciones desde el cmd:

```
$ cd <path to fusion2urdf>
Copy-Item ".\URDF_Exporter\" -Destination "${env:APPDATA}\Autodesk\
    Autodesk Fusion 360\API\Scripts\" -Recurse
```

Para habilitar la funcionalidad de exportación de modelos en formato URDF, se agrega un complemento en la barra de menús de *Autodesk Fusion 360*. Este complemento se denomina "*URDF Exporter*" [57] y consiste en un archivo Python que se ejecuta para generar el archivo URDF correspondiente.

A continuación, se detallan los pasos necesarios para utilizar este complemento:

- Instala *Autodesk Fusion 360* en tu sistema operativo Windows 10.
- En la barra de menús superior, busca la opción "Complementos" y haz clic en ella.
- En el menú desplegable, busca y selecciona la opción "*URDF Exporter*".
- Esto mostrará una ventana adicional donde se encuentra el archivo Python del complemento.
- Haz doble clic en el archivo Python "*URDF Exporter*" para ejecutarlo.

Al ejecutar el archivo Python, se generará automáticamente el archivo URDF correspondiente al modelo que se desea exportar. Asegurarse de seguir las instrucciones adicionales que puedan aparecer en la ventana para configurar las opciones de exportación, según las necesidades específicas que se tengan.

A.6. Datos para las pruebas

A continuación se muestran las listas de datos que se utilizaron para enviar al robot desde la plataforma y ejecutar las pruebas de manipulación de objetos.

Tabla A.1.: Valores articulares de la prueba de manipulación de objetos con control articular.

	Articulación (Grados RPY)					
N°	**J1**	**J2**	**J3**	**J4**	**J5**	**J6**
1	27.41	-61.91	79.79	-118-91	-89.69	-106.02
2	35.58	-39.47	66	-128.95	-87.02	-97.75
3	35.58	-36.5	67.36	-133.28	-87.05	-97.74
4	35.57	-34.49	67.85	-135.79	-87.06	-97.73
5	35.58	-31.98	68.09	-138.53	-87.06	-97.73
6	35.58	-39.53	65.96	-128.85	-87.06	-97.75
7	33.86	-67.39	102.54	-145.69	-84.33	-97.75
8	33.16	-70.77	115.25	-150.07	-84.3	-98.47
9	33.16	-77.65	109.99	-137.92	-84.31	-98.51
10	29.67	-44.7	75.13	-137.62	-86.09	-103.62
11	29.68	-38.84	77.51	-145.09	-86.24	-103.61
12	29.68	-36.34	77.72	-147.79	-86.26	-103.61
13	29.68	-33.28	77.58	-150.72	-86.28	-103.61
14	29.69	-41.45	77.02	-141.97	-86.22	-103.61
15	26.97	-67.84	105.49	-141.93	-87.81	-103.57
16	25.56	-69.09	112.77	-144.58	-88.03	-103.22
17	25.56	-75.73	107.65	-132.83	-87.94	-103.26
18	26.66	-43.67	74.36	-134.74	-102.07	-104.02
19	26.66	-37.88	76.18	-142.36	-102.13	-104
20	26.66	-34.78	76.34	-145.61	-102.15	-104
21	26.66	-33.08	76.25	-147.22	-102.15	-103.98
22	26.66	-39.37	75.94	-140.6	-102.08	-103.98
23	22.89	-69.26	108.35	-144.67	-103.82	-101.86
24	22.91	-66.55	111.93	-148.92	-103.81	-99.6
25	22.91	-73.46	107.22	-137.32	-103.72	-99.63
26	23.63	-88.49	107.24	-110.81	-87.76	-83.04

Tabla A.2.: Coordenadas de la prueba de manipulación de objetos con control cartesiano.

	Posición (m)			**Rotación (Grados RPY)**		
N°	**Px**	**Py**	**Pz**	**Rx**	**Ry**	**Rz**
1	-29.112	-27.054	18.649	-180	0	-117.71
2	-27.03	-34.774	6.87	-179.41	-1.98	-166.34
3	-27.03	-34.775	5.387	-179.41	-1.98	-166.33
4	-27.031	-34.775	3.146	-179.41	-1.98	-166.34
5	-27.029	-34.775	2.225	-179.41	-1.97	-166.34
6	-27.032	-34.774	7.512	-179.41	-1.98	-166.33
7	-31.539	-17.169	7.423	-179.41	-1.98	-166.34
8	-31.541	-17.168	2.747	-179.41	-1.98	-166.34
9	-31.539	-17.171	8.264	-179.41	-1,98	-116.34
10	-21.726	-34.546	6.537	177.41	-18.63	-122.69
11	-21.724	-34.547	3.513	177.41	-18.63	-122.69
12	-21.725	-34.548	1.767	177.41	-18.63	-122.69
13	-21.723	-34.547	0.445	177.41	-18.63	-122.69
14	-21.723	-34.546	5.045	177.41	-18.63	-122.69
15	-36.498	-19.12	4.89	177.41	-18.63	-122.69
16	-36.498	-19.117	1.303	177.41	-18.62	-122.7
17	-36.496	-19.121	5.381	177.41	-18.63	-122.69
18	-22.283	-39.905	7.051	155.3	-8.9	-109.76
19	-22.282	-39.905	3.943	155.3	-8.9	-109.76
20	-22.283	-39.906	1.886	155.29	-8.9	-109.76
21	-22.282	-39.907	0.384	155.29	-8.91	-109.76
22	-22.284	-39.905	7.723	155.29	-8.91	-109.77
23	-39.303	-22.994	5.928	154.75	-9.76	-110.99
24	-39.303	-22.993	1.234	154.76	-9.75	-110.99
25	-39.306	-22.994	8.167	154.76	-9.76	-110.98
26	-16.423	-38.845	7.097	-178.21	16.03	-124.01
27	-16.421	-38.845	3.769	-178.21	16.03	-124.02
28	-16.425	-38.844	2.321	-178.22	16.02	-124.02
29	-16.426	-38.846	0.665	-178.21	16.02	-124.02
30	-16.422	-38.846	5.819	-178.21	16.02	-124.01
31	-42.925	-25.003	5.308	175.92	19.44	-39.17
32	-42.924	-25.005	1.5	175.95	19.44	-39.08
33	-42.923	-25.006	5.769	175.95	19.44	-39.08
34	-28.173	-27.346	11.09	174.71	3.7	-50.45

Tabla A.3.: Coordenadas de la prueba de manipulación para dibujo sobre plano XY.

	Posición (m)			**Rotación (Grados RPY)**		
N°	**Px**	**Py**	**Pz**	**Rx**	**Ry**	**Rz**
1	-30.162	-34.145	12.19	-180	0	-165.05
2	-26.426	-40.152	5.625	-176.8	-1.35	156.17
3	-26.427	-40.154	3.672	-176.79	-1.36	156.17
4	-26.427	-40.151	1.54	-176.8	-1.35	156.16
5	-26.429	-40.154	-0.022	-176.79	-1.36	156.17
6	-26.428	-40.152	7.757	-176.79	-1.35	156.16
7	-29.384	-42.61	9.974	-167.74	-18.68	-135.93
8	-36.471	-33.814	3.229	-148.08	-18.14	-141.59
9	-30.019	-40.231	3.068	-150.15	-19.11	-129.92
10	-36.109	-39.855	3.134	-152.35	-23.71	-132.7
11	-36.471	-33.814	3.134	-148.08	-18.14	-141.59
12	-27.328	-39.731	9.025	-177.22	0.65	153.13
13	-26.665	-40.293	1.087	-178.9	0.39	153.03
14	-26.663	-40.295	4.602	-178.9	0.39	153.03
15	-33.499	-33.387	12.271	-176.52	-3.02	-168.3

Tabla A.4.: Coordenadas de la prueba de construcción de torre con piezas cúbicas.

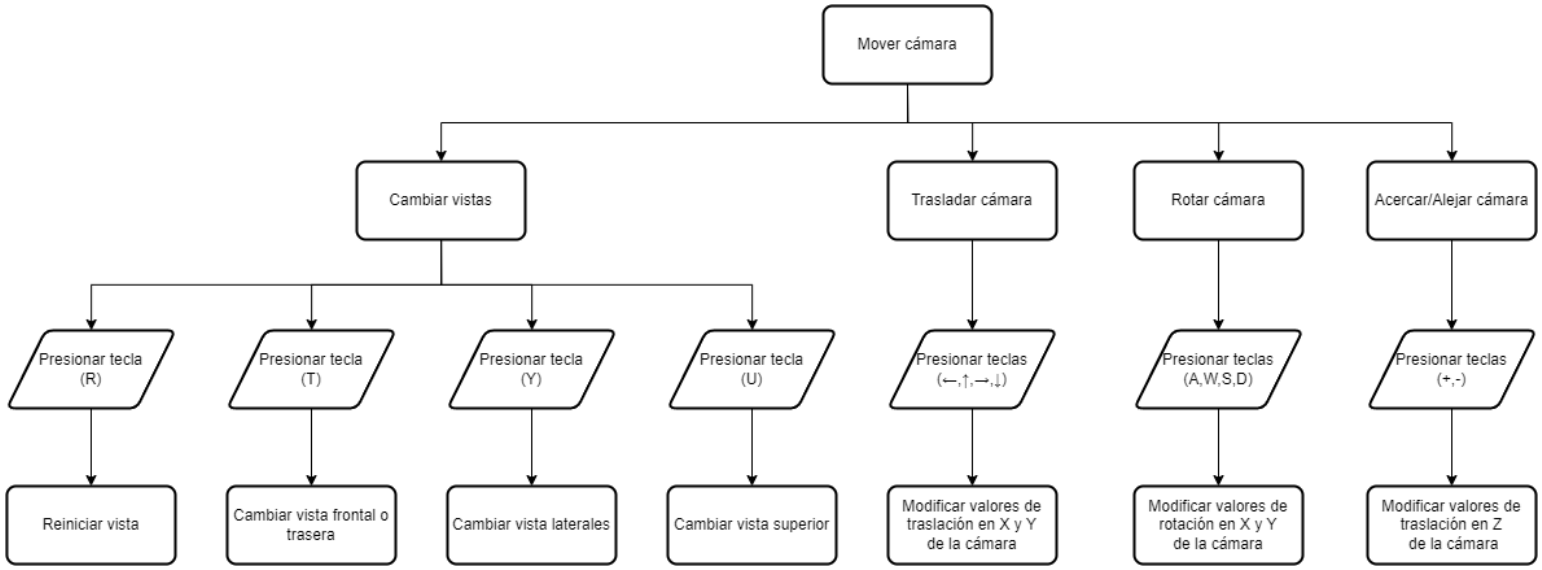
	Posición (m)			**Rotación (Grados RPY)**		
N°	**Px**	**Py**	**Pz**	**Rx**	**Ry**	**Rz**
1	-32.09	-29.982	14.059	-180	0	-108.88
2	-20.501	-29.75	7.998	179.98	-0.01	-108.88
3	-20.503	-29.748	5.066	179.98	-0.01	-108.88
4	-20.501	-29.748	3.031	179.98	-0.01	-108.88
5	-20.504	-29.748	1.991	179.98	-0.01	-108.88
6	-20.506	-29.747	13.372	179.98	-0.01	-108.88
7	-37.476	-23.383	5.987	179.98	-0.01	-108.88
8	-37.072	-23.178	8.534	179.98	-0.01	-108.88
9	-20.883	-41.16	8.691	179.98	-0.01	-108.88
10	-20.881	-41.16	5.193	179.98	-0.01	-108.88
11	-20.883	-41.16	2.952	179.98	-0.01	-108.88
12	-20.881	-41.163	2.065	179.98	-0.01	-108.88
13	-20.883	-41.16	11.896	179.98	-0.01	-108.88
14	-37.801	-23.47	11.249	179.98	-0.01	-108.88
15	-37.802	-23.469	8.917	179.98	-0.01	-108.88
16	-37.802	-23.468	13.162	179.98	-0.01	-108.88
17	-29.017	-27.316	15.443	179.98	-0.01	-108.88

Tabla A.5.: Coordenadas de la prueba de *pick and place* con objetos esféricos.

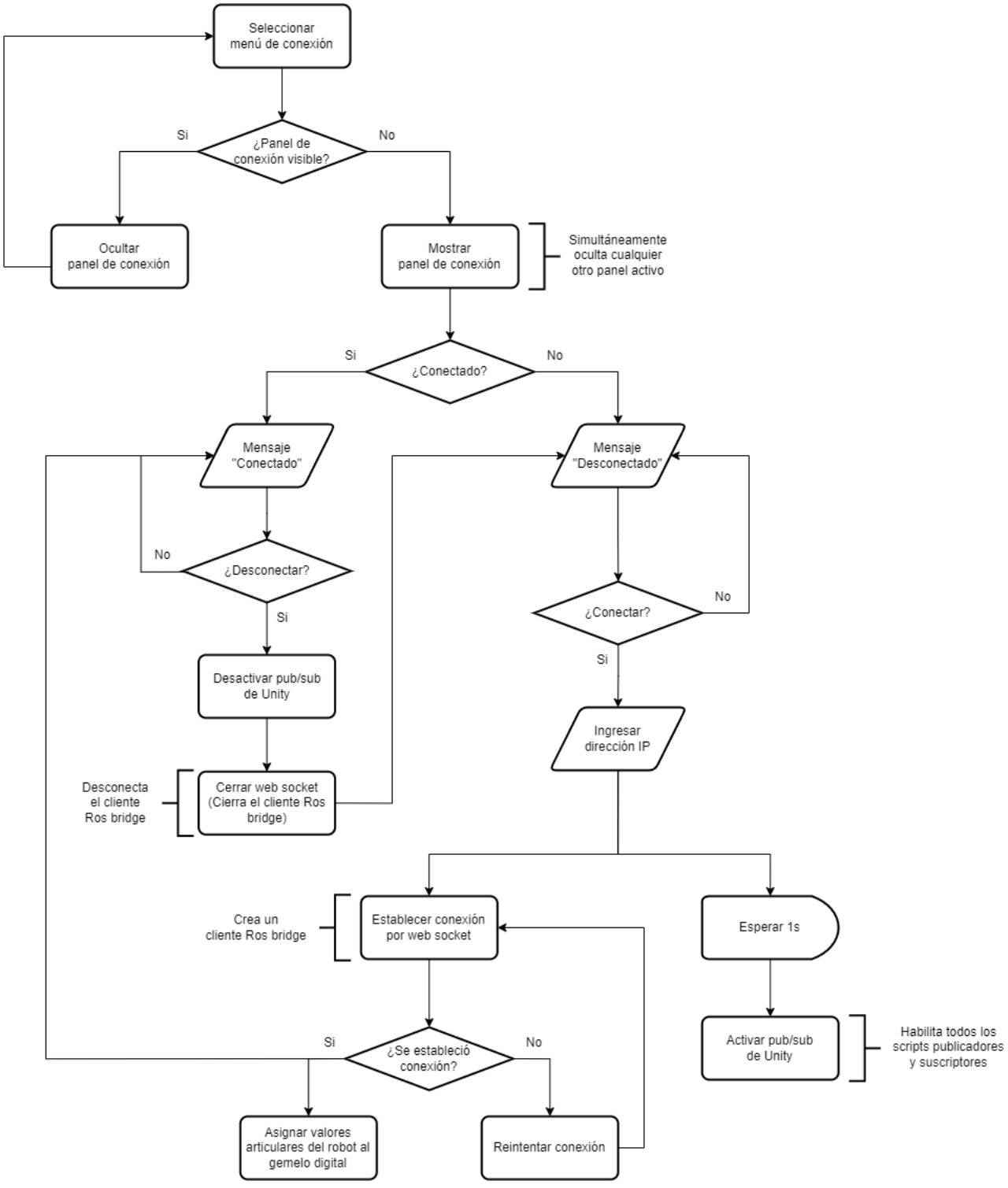
	Posición (m)			**Rotación (Grados RPY)**		
N°	**Px**	**Py**	**Pz**	**Rx**	**Ry**	**Rz**
1	-29.063	-29.045	14.662	180	-0.01	-133.58
2	-11.0927	-36.132	6.267	-177.37	11.48	-123.66
3	-11.099	-36..133	3.845	-177.37	11.48	-123.66
4	-11.97	-36.132	1.342	-177.37	11.48	-123.66
5	-11.097	-36.131	0.519	-177.37	11.49	-123.67
6	-11.097	-36.132	5.927	-177.37	11.48	-123.66
7	-29.409	-29.741	14.445	177	-1.09	-106.9
8	-20.917	-40.025	8.372	178.43	-2.7	-99.78
9	-20.913	-40.031	4.479	178.42	-2.71	-99.77
10	-20.913	-40.032	2.632	178.41	-2.71	-99.77
11	-20.913	-40.031	0.824	178.41	-2.71	-99.77
12	-20.913	-40.031	7.008	178.41	-2.71	-99.77
13	-28.183	-29.701	13.856	176.64	-3.41	-87.46
14	-35.272	-15.088	7.148	169.17	24.66	-71.16
15	-35.272	-15.088	4.098	169.17	24.67	-71.16
16	-35.27	-15.089	1.805	169.17	24.66	-71.16
17	-35.271	-15.089	0.134	169.16	24.67	-71.16
18	-35.271	-15.088	9.904	169.17	24.66	-71.16
19	-28.686	-29.478	13.298	169.17	24.66	-83.11
20	-37.79	-24.407	8.87	165.36	10.67	-29.42
21	-37.793	-24.408	3.901	165.35	10.68	-29.42
22	-37.788	-24.41	2.095	165.35	10.67	-29.42
23	-37.791	-24.407	0.496	165.35	10.67	-29.42
24	-37.789	-24.408	9.312	165.36	10.67	-29.42
25	-28.101	-29.249	16.048	166.98	-0.8	-59.51

A.7. Diagramas de flujo

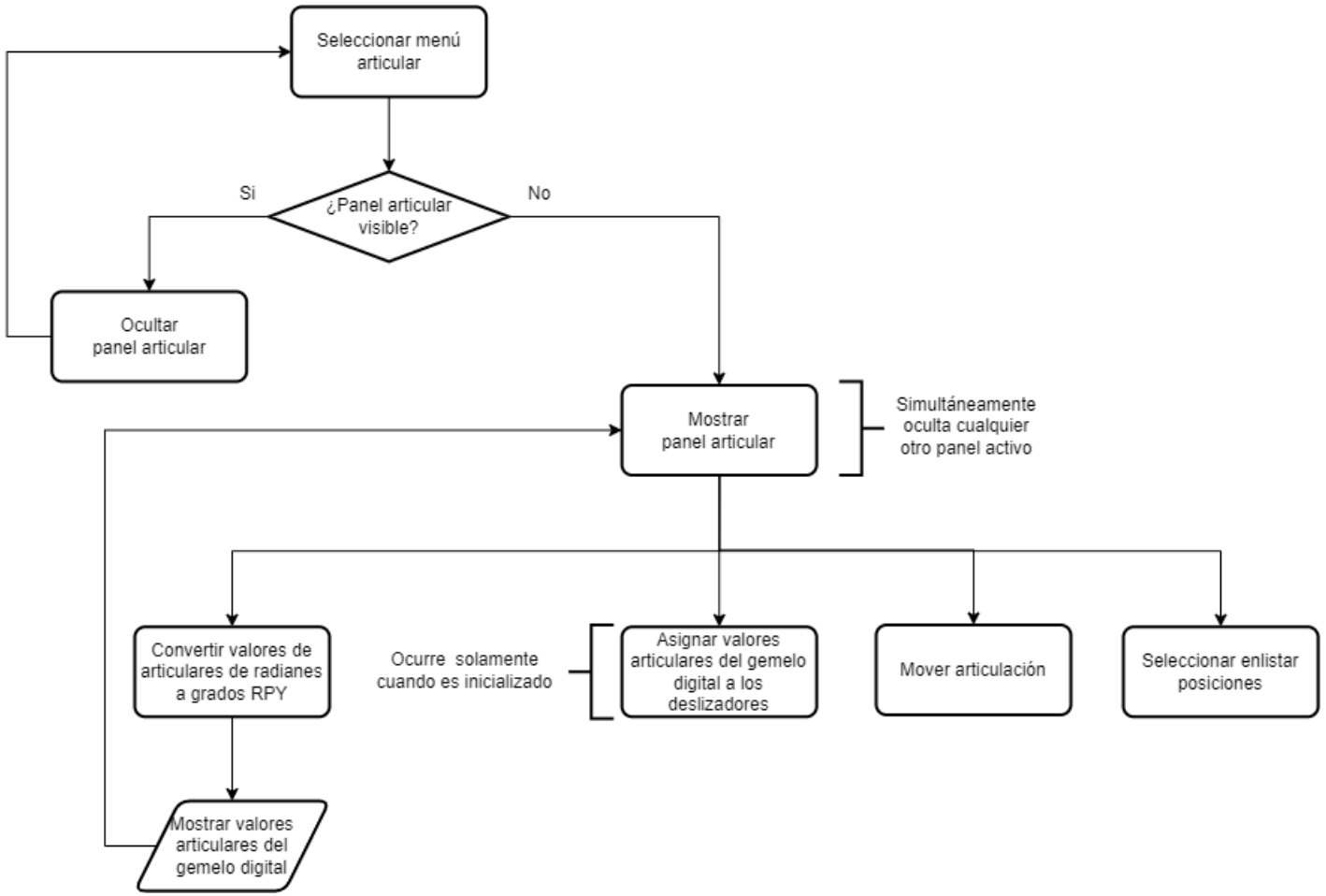
Anexo A.35.: Diagrama de flujo de la pantalla principal en Unity.



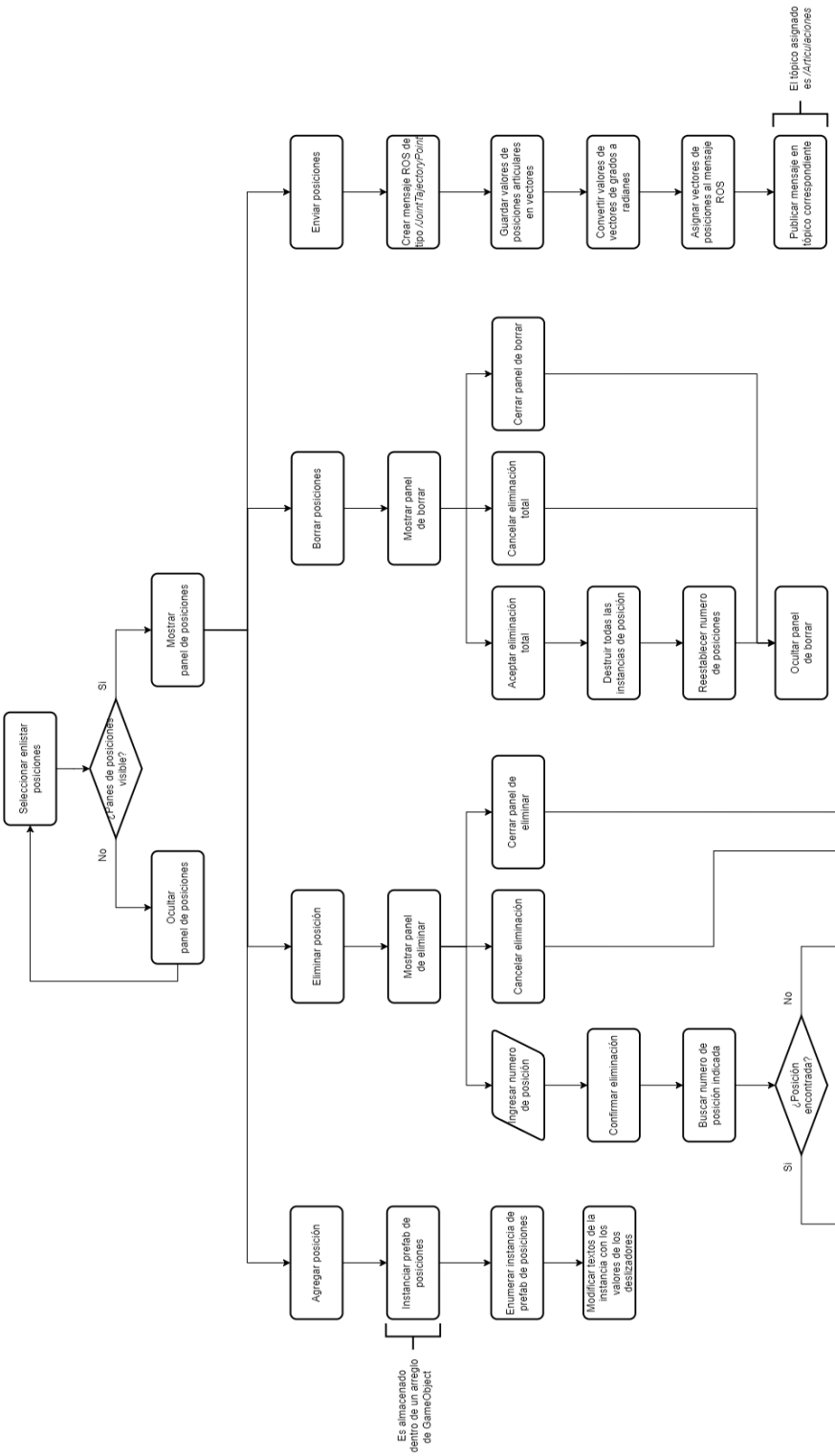
Anexo A.36.: Diagrama de flujo para mover la cámara de Unity.



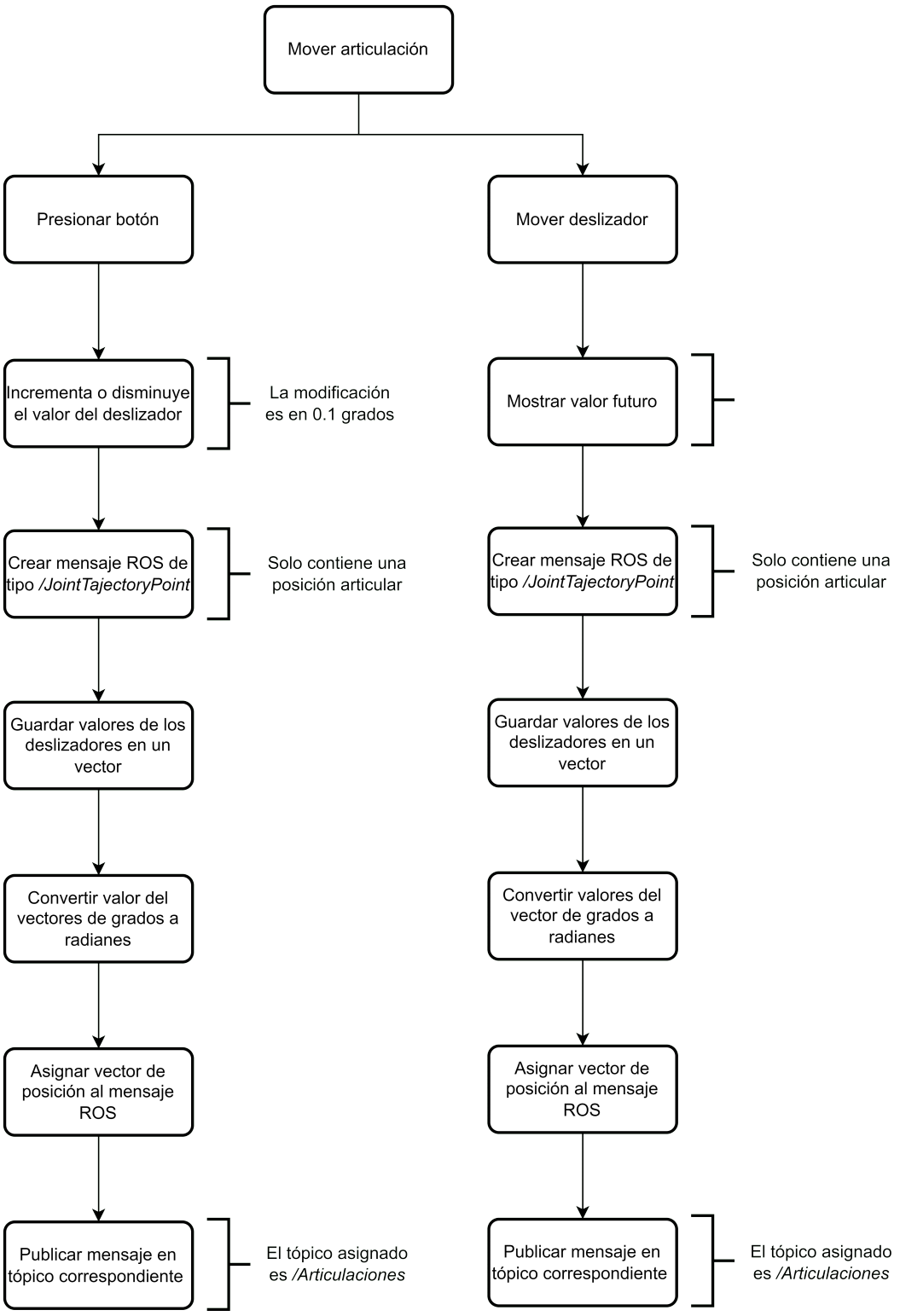
Anexo A.37.: Diagrama de flujo para la conexión de Unity.



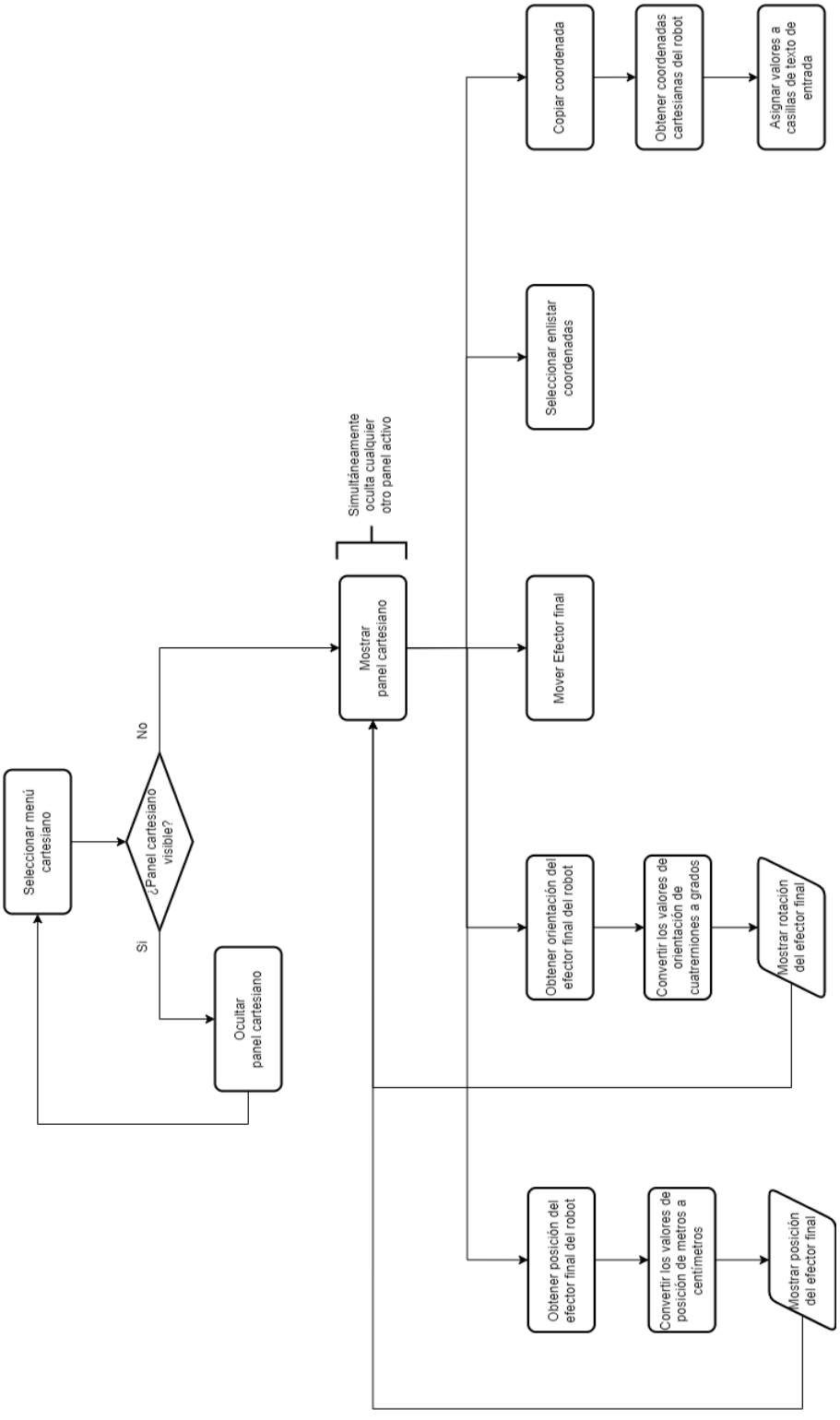
Anexo A.38.: Diagrama de flujo para el panel articular.



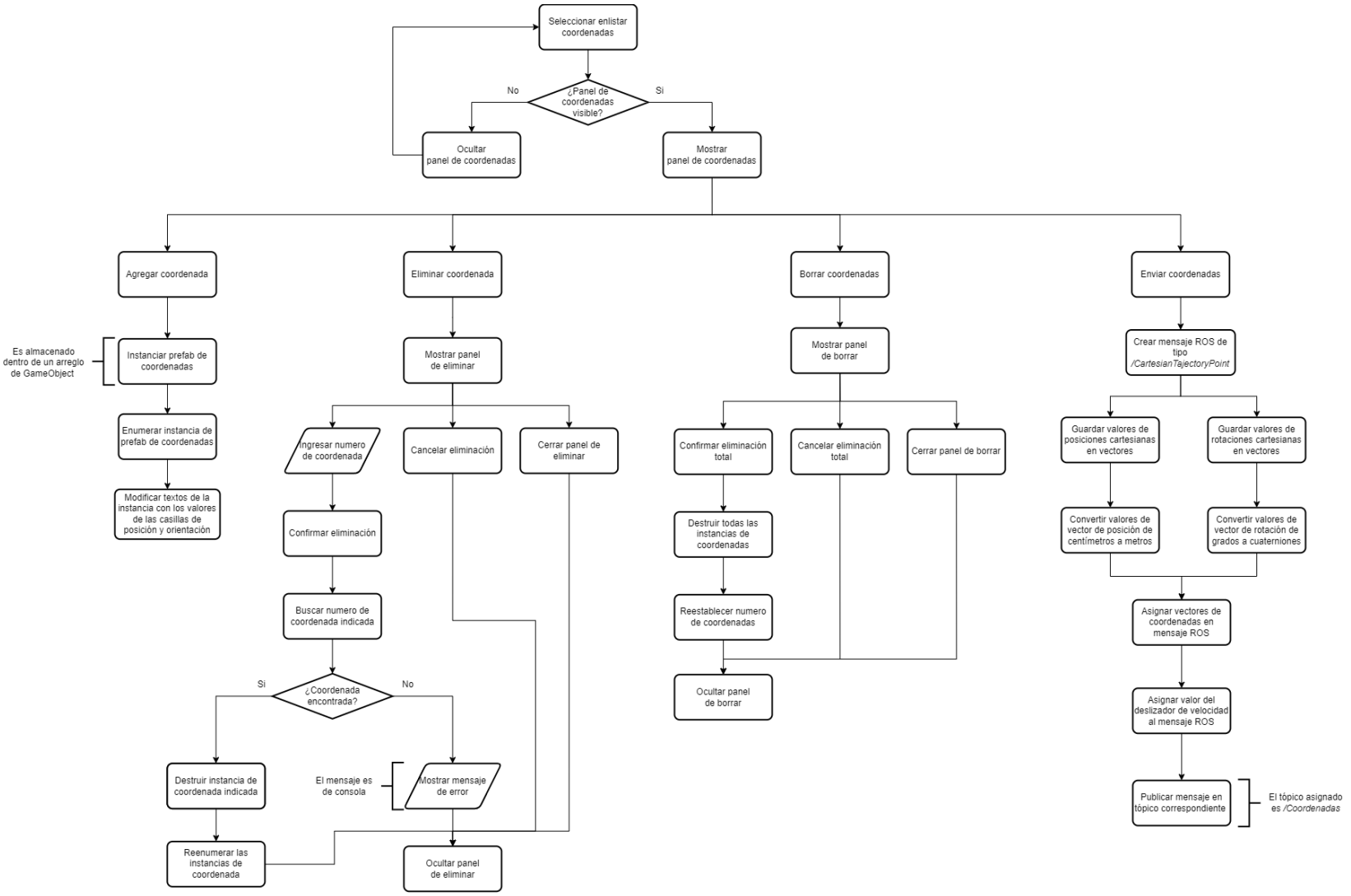
Anexo A.39.: Diagrama de flujo para enlistar las posiciones.



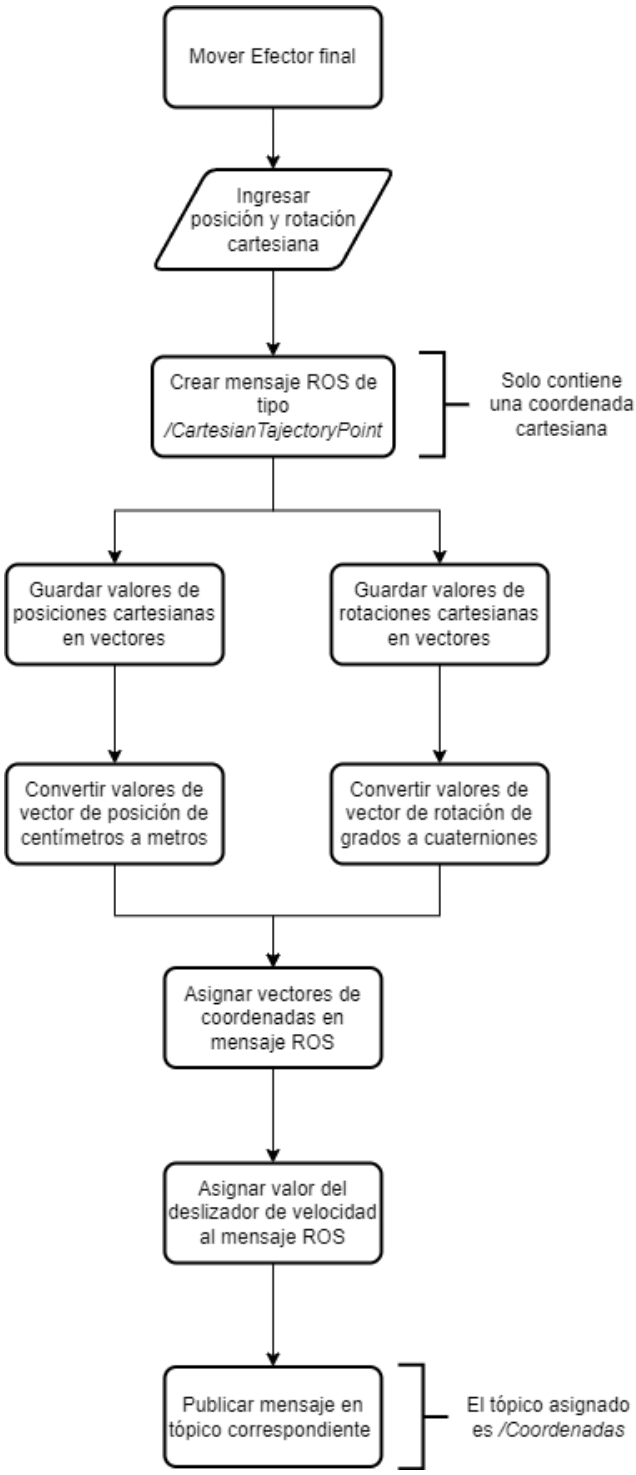
Anexo A.40.: Diagrama de flujo para mover las articulaciones.



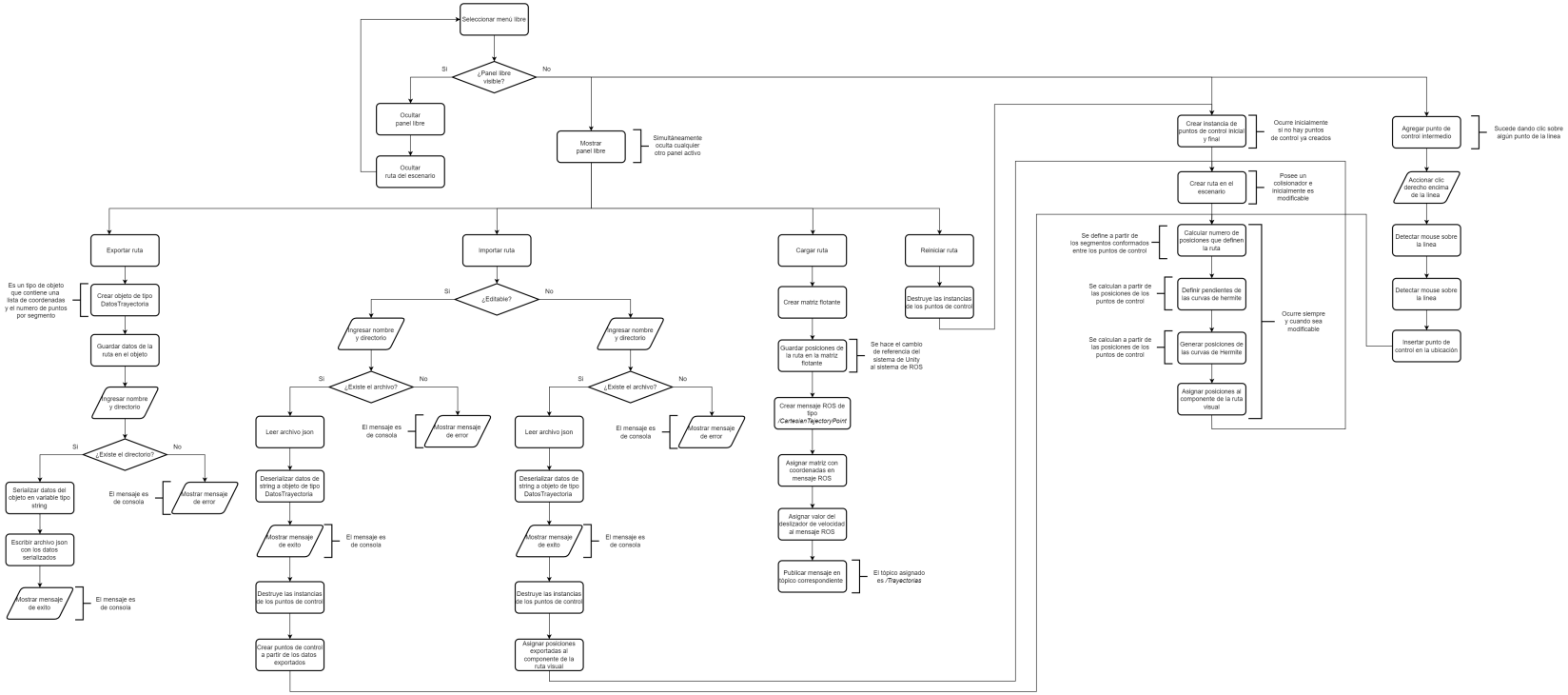
Anexo A.41.: Diagrama de flujo para el panel cartesiano.



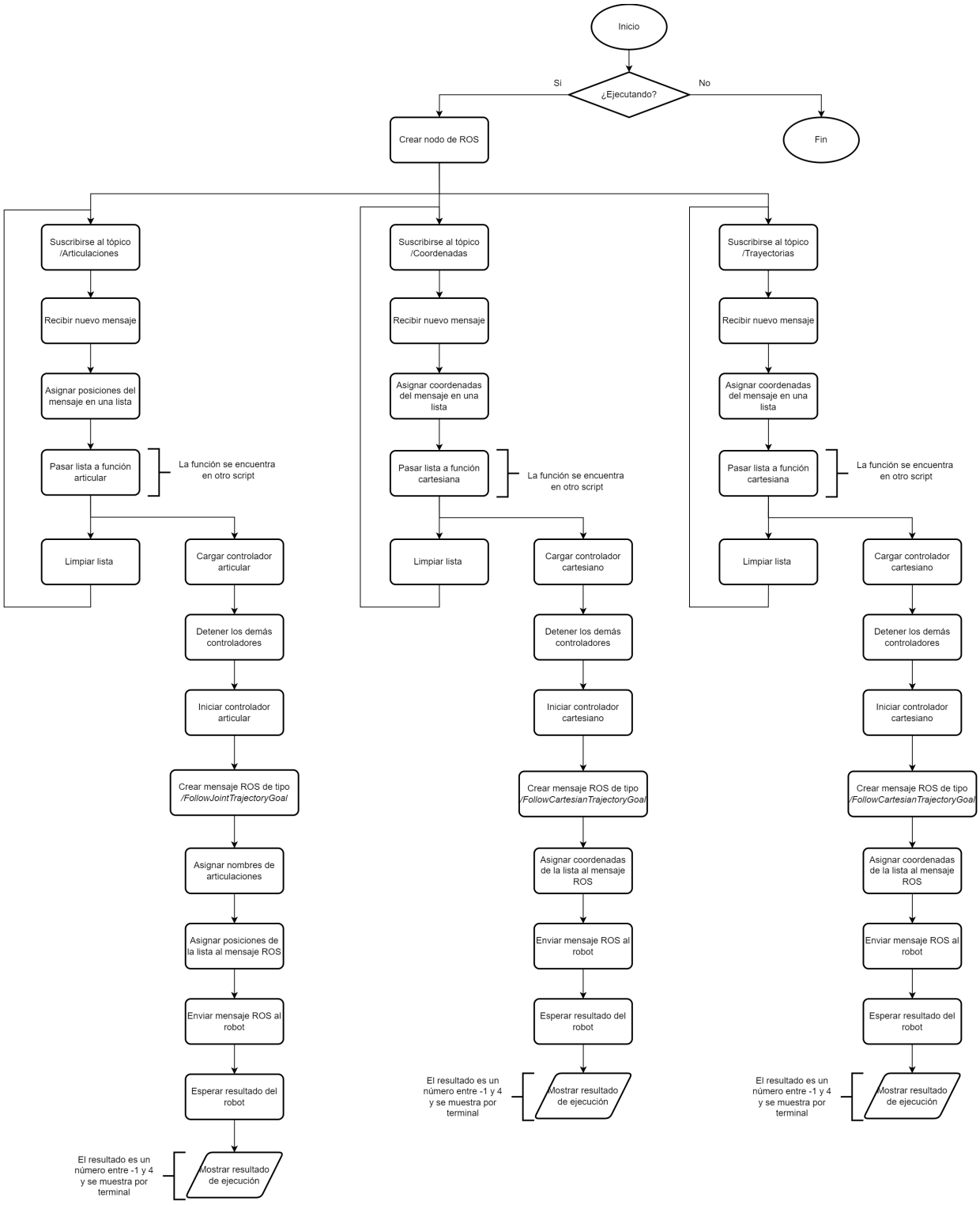
Anexo A.42.: Diagrama de flujo para enlistar las coordenadas.



Anexo A.43.: Diagrama de flujo para mover el efector final.



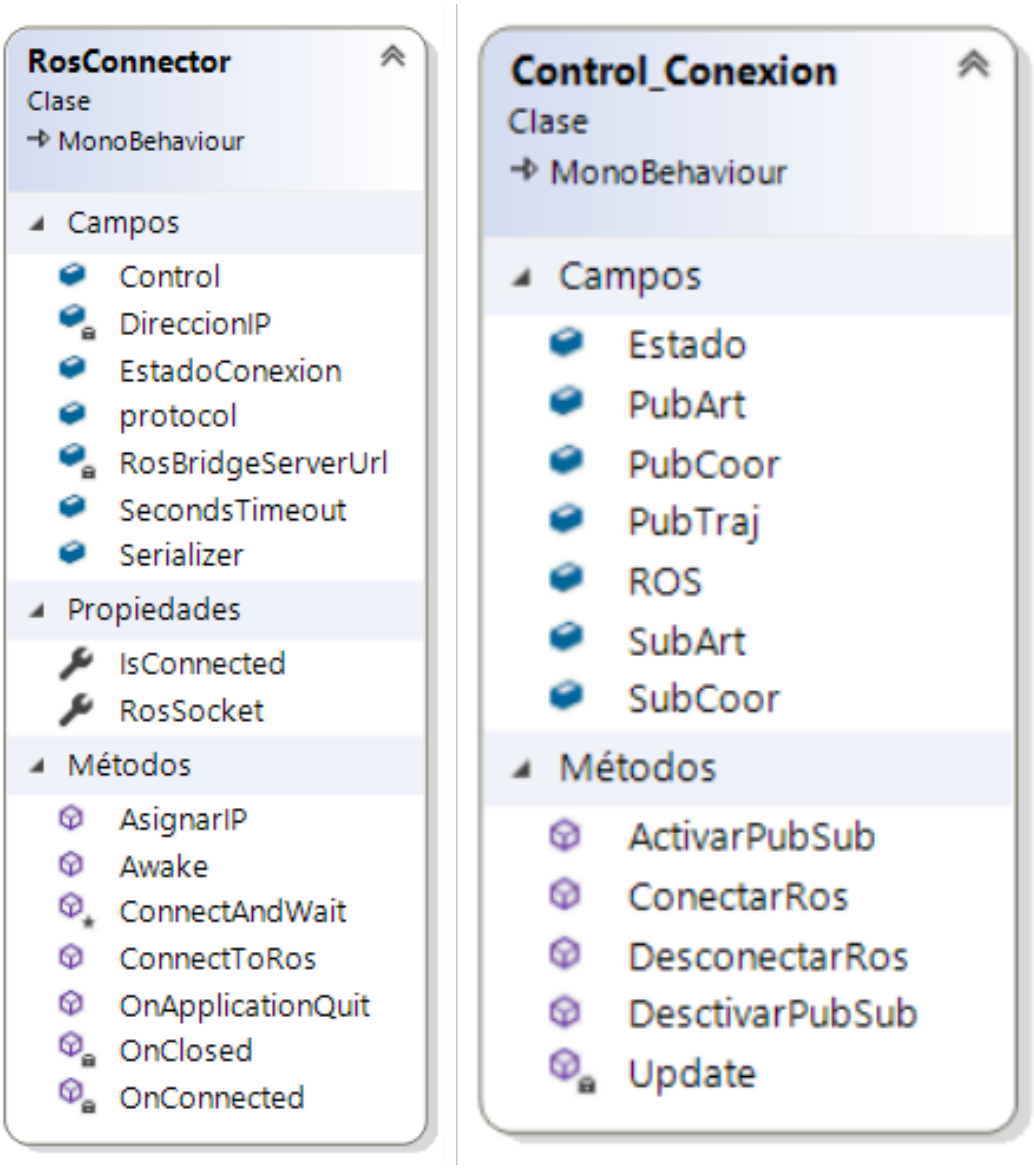
Anexo A.44.: Diagrama de flujo para las trayectorias libres.



Anexo A.45.: Diagrama de flujo de Python.

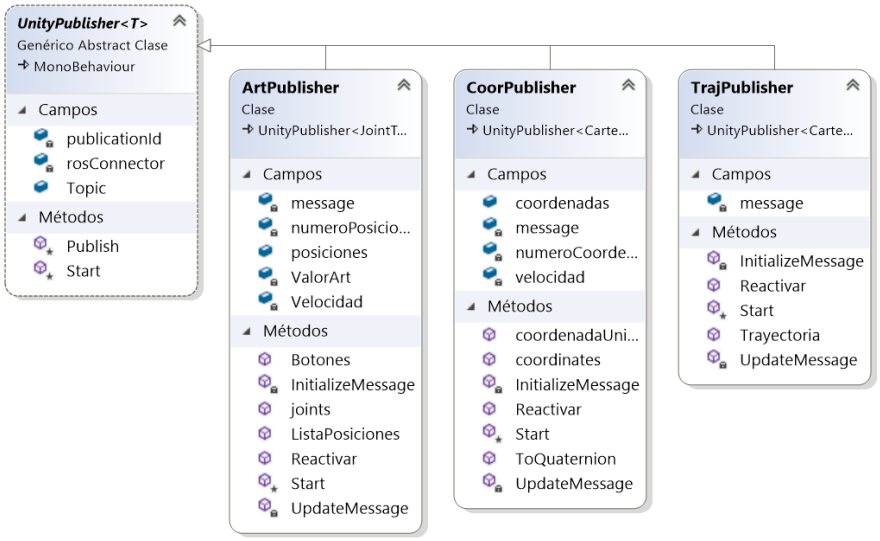
A.8. Diagramas de clases

A.8.1. Comunicación ROS

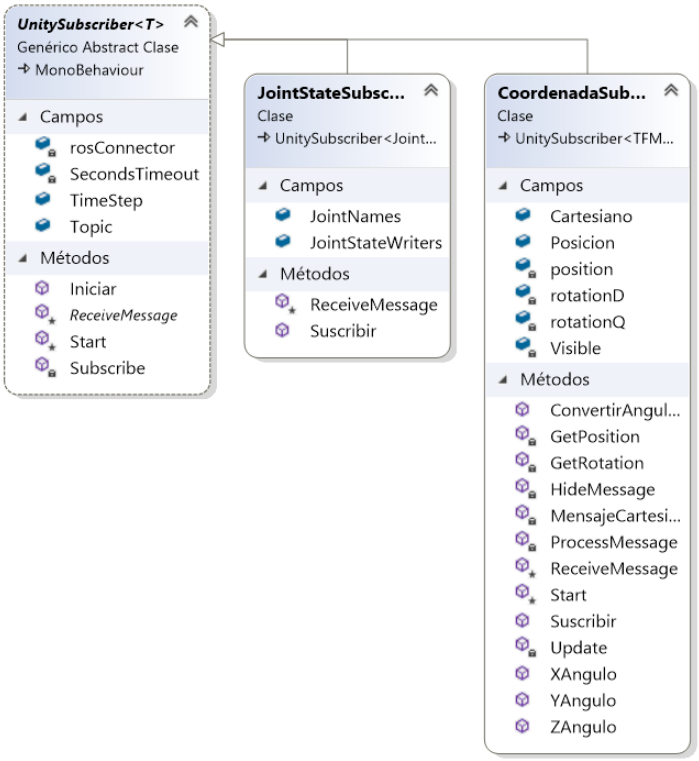


Anexo A.46.: Comunicación ROS.

A.8.2. Publicadores y suscriptores ROS

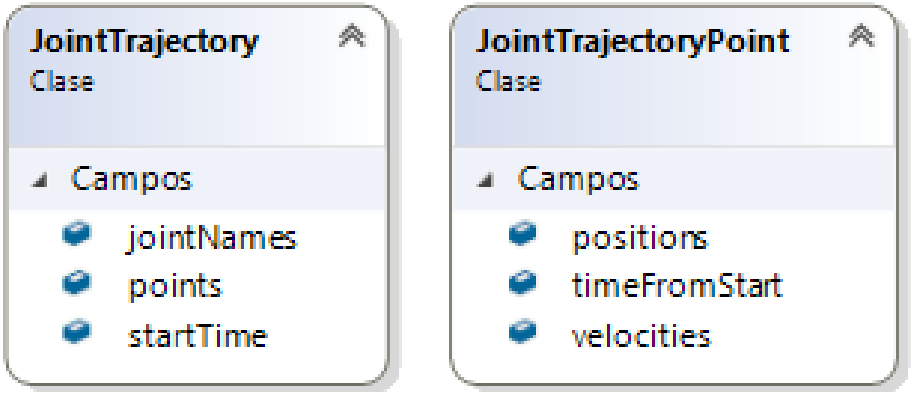


Anexo A.47.: Publicadores ROS.

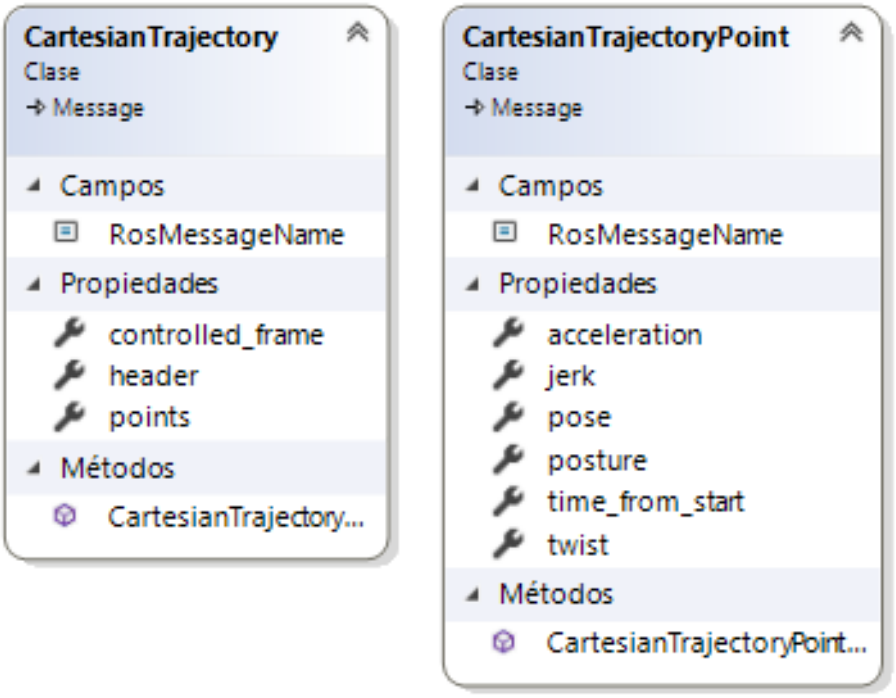


Anexo A.48.: Suscriptores ROS.

A.8.3. Mensajes ROS

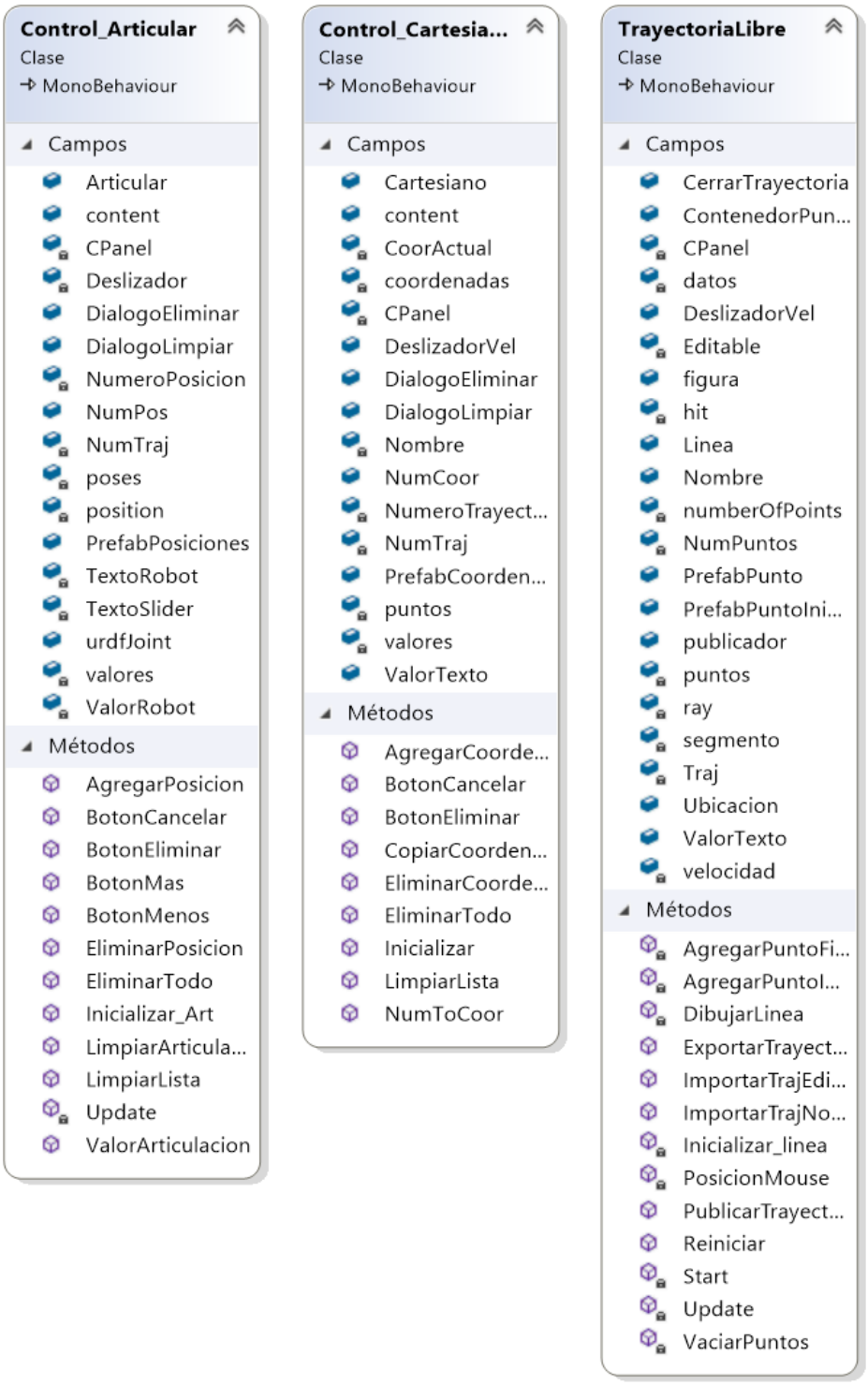


Anexo A.49.: Mensajes articulares.

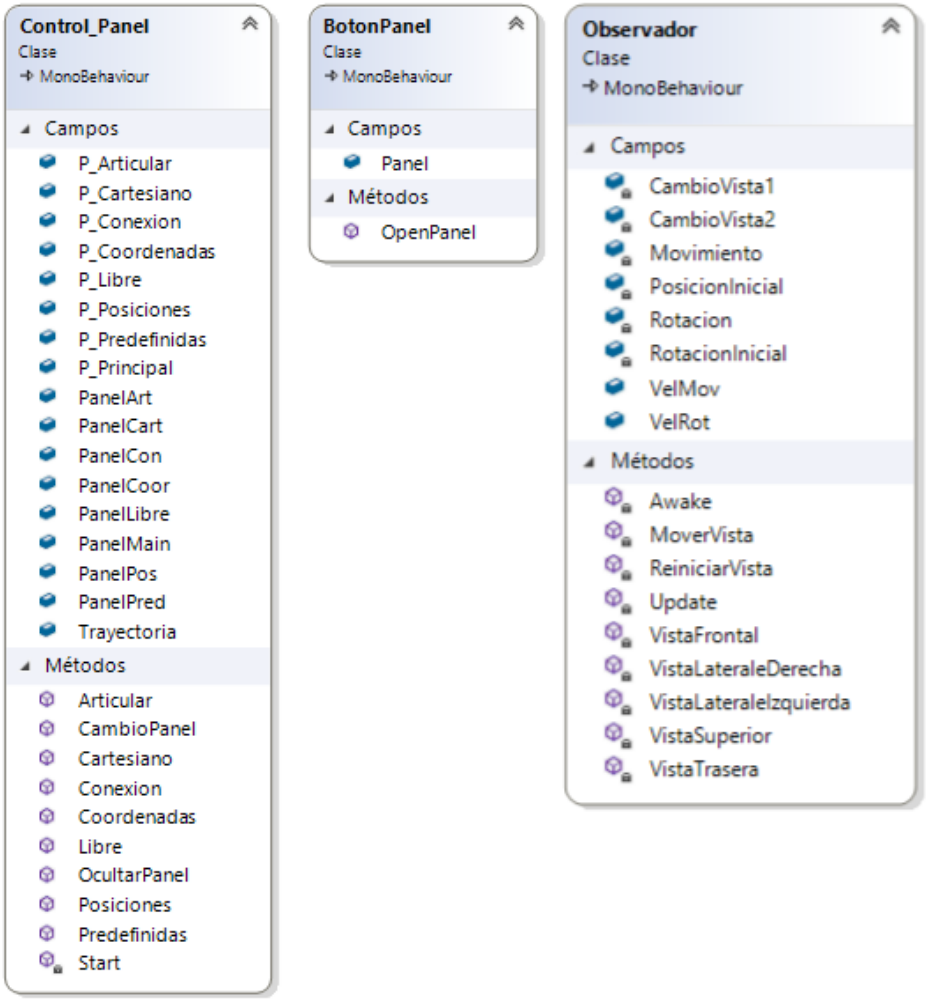


Anexo A.50.: Mensajes cartesianos.

A.8.4. Control de la interfaz

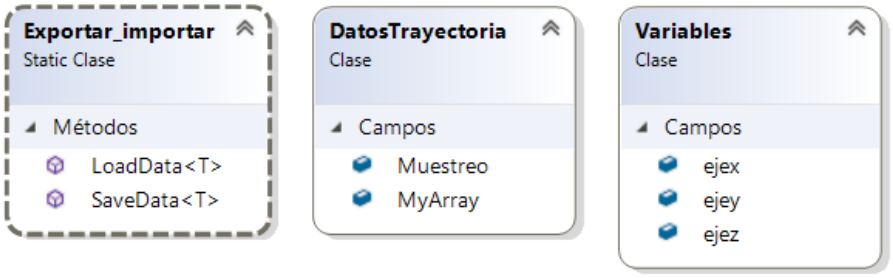


Anexo A.51.: Modos de control.



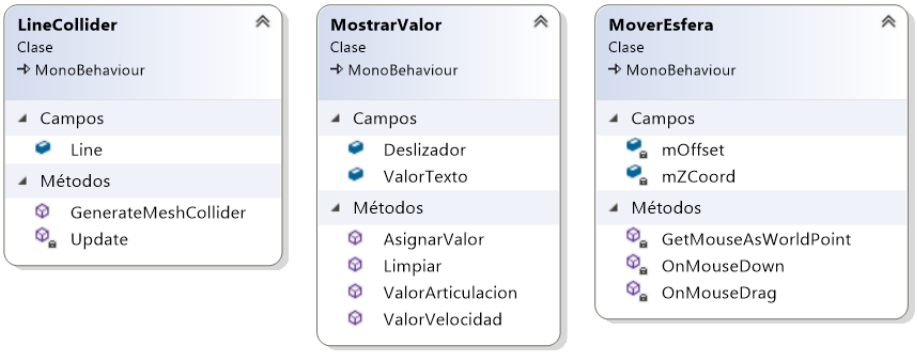
Anexo A.52.: Control de paneles y del observador.

A.8.5. Exportar e importar



Anexo A.53.: *Scripts* para exportar e importar rutas.

A.8.6. Secundarios



Anexo A.54.: *Scripts* secundarios.

Bibliografía

[1] P. Segura, O. Lobato, and A. Ramirez, "Human-robot collaboration systems: Components and applications," 11 2020, doi: 10.11159/cdsr20.150.

[2] H. Yun and M. Jun, "Immersive and interactive cyber-physical system (i2cps) and virtual reality interface for human involved robotic manufacturing," *Journal of Manufacturing Systems*, vol. 62, pp. 234–248, 2022, doi: 10.1016/j.jmsy.2021.11.018.

[3] G. Ajaykumar and C.-M. Huang, "User needs and design opportunities in end-user robot programming." Association for Computing Machinery, 2020, p. 93–95, doi: 10.1145/3371382.3378300.

[4] M. De Gier, "Control of a robotic arm: Application to on-surface 3d-printing," 2015, master of Science Thesis, Faculty of Mechanical, Delft University of Technology, Netherlands.

[5] U. Robots, "Polyscope manual," Último acceso 06-03-2023 a https://cfzcobots.com/wp-content/uploads/2018/06/UR3e_User_Manual_es_Global.pdf, 2018.

[6] A. Vivas and J. M. Sabater, "Ur5 robot manipulation using matlab/simulink and ros," in *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2021, pp. 338–343, doi: 10.1109/ICMA52036.2021.9512650.

[7] J. Li, X. Yang, X. Wu, and L. Zhong, "Research on path calculation and simulation system of variable parameter manipulator," *Journal of Physics: Conference Series*, vol. 1670, no. 1, p. 12002, 11 2020, doi: 10.1088/1742-6596/1670/1/012002.

[8] V. Popov, S. Ahmed, N. Shakev, and A. Topalov, "Gesture-based interface for real-time control of a mitsubishi scara robot manipulator," *IFAC-PapersOnLine*, vol. 52, pp. 180–185, 2019, doi: 10.1016/j.ifacol.2019.12.469.

[9] G. Lunghi, R. Marin, M. Di Castro, A. Masi, and P. J. Sanz, "Multimodal human-robot interface for accessible remote robotic interventions in hazardous

environments," *IEEE Access*, vol. 7, pp. 127 290–127 319, 2019, doi: 10.1109/ACCESS.2019.2939493.

[10] M. Abdeetedal and M. R. Kermani, "An open-source integration platform for multiple peripheral modules with kuka robots," *CIRP Journal of Manufacturing Science and Technology*, vol. 27, pp. 46–55, 2019, doi: 10.1016/j.cirpj.2019.08.003.

[11] A. Raviola, A. Coccia, A. De Martin, A. C. Bertolino, S. Mauro, and M. Sorli, "Development of a human-robot interface for a safe and intuitive telecontrol of collaborative robots in industrial applications," in *Advances in Service and Industrial Robotics*, A. Müller and M. Brandstötter, Eds. Springer International Publishing, 2022, pp. 556–563.

[12] H. Ashrafian, O. Clancy, V. Grover, and A. Darzi, "The evolution of robotic surgery: surgical and anaesthetic aspects," *BJA: British Journal of Anaesthesia*, vol. 119, pp. 72–84, 11 2017, doi: 10.1093/bja/aex383.

[13] R. Muradore, P. Fiorini, G. Akgun, D. Erol Barkana, M. Bonfé, F. Boriero, A. Caprara, G. De Rossi, R. Dodi, O. Elle, F. Ferraguti, L. Gasperotti, R. Gassert, K. Mathiassen, D. Handini, O. Lambercy, L. Li, M. Kruusmaa, A. Manurung, and E. Yantac, "Development of a cognitive robotic system for simple surgical tasks," *International Journal of Advanced Robotic Systems*, vol. 12, 04 2015, doi: 10.5772/60137.

[14] G. M. Fortunato, G. Rossi, A. F. Bonatti, A. De Acutis, C. Mendoza-Buenrostro, G. Vozzi, and C. De Maria, "Robotic platform and path planning algorithm for in situ bioprinting," *Bioprinting*, vol. 22, p. e00139, 2021, doi: 10.1016/j.bprint.2021.e00139.

[15] K. Schwaner, I. Iturrate, J. Holm, C. Rosendahl, P. Jensen, and T. Rajeeth, "Mops: A modular and open platform for surgical robotics research," in *2021 International Symposium on Medical Robotics (ISMR)*, 2021, pp. 1–8, doi: 10.1109/ISMR48346.2021.9661539.

[16] A. Bihlmaier, T. Beyl, P. Nicolai, M. Kunze, J. Mintenbeck, L. Schreiter, T. Brennecke, J. Hutzl, J. Raczkowsky, and H. Wörn, *ROS-Based Cognitive Surgical Robotics*. Springer International Publishing, 2016, doi: 10.1007/978-3-319-26054-9_12.

[17] A. Higueras, I. Delgado, L. Serrano, A. Principe, C. Enriquez, M. González, R. Rocamora, G. Conesa, and L. Serra, "Sylvius: A multimodal and multidisciplinary

platform for epilepsy surgery," *Computer Methods and Programs in Biomedicine*, vol. 203, p. 106042, 03 2021, doi: 10.1016/j.cmpb.2021.106042.

[18] D. Adair, K. Gomes, Z. Kiss, D. Gobbi, and Y. Starreveld, "Tactics: an open-source platform for planning, simulating and validating stereotactic surgery," *Computer Assisted Surgery*, vol. 25, pp. 1–14, 12 2020, doi: 10.1080/24699322.2020.1760354.

[19] E. Caudana, G. Baltazar, R. Acevedo, P. Ponce, N. Mazon, and J. Hernandez, "Robotics: Implementation of a robotic assistive platform in a mathematics high school class," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, 2019, pp. 1589–1594, doi: 10.1109/ISIE.2019.8781520.

[20] G. López, A. Romeo, and J. Guerrero, "Project based learning of robot control and programming," 1 2009, doi: 10.48550/arXiv.2305.11279.

[21] L. Pérez, E. Diez, R. Usamentiaga, and D. F. García, "Industrial robot control and operator training using virtual reality interfaces," *Computers in Industry*, vol. 109, pp. 114–120, 2019, doi: 10.1016/j.compind.2019.05.001.

[22] J. Chacon, D. Goncalves, E. Besada, and J. López, "Un laboratorio remoto de código abierto y bajo coste para el brazo robótico educativo dobot magician," *Revista Iberoamericana de Automática e Informática industrial*, vol. 20, no. 2, pp. 124–136, 2023, doi: 10.4995/riai.2022.17477.

[23] M. Peshkin and J. Colgate, "Cobots," *Industrial Robot: An International Journal*, vol. 26, no. 5, pp. 335–341, 1999, doi: 10.1108/01439919910283722.

[24] J. Fernández, D. Mronga, M. Günther, T. Knobloch, M. Wirkus, M. Schröer, M. Trampler, S. Stiene, E. Kirchner, V. Bargsten, T. Bänziger, J. Teiwes, T. Krüger, and F. Kirchner, "Multimodal sensor-based whole-body control for human–robot collaboration in industrial settings," *Robotics and Autonomous Systems*, vol. 94, pp. 102–119, 2017, doi: 10.1016/j.robot.2017.04.007.

[25] C. Rodriguez, "Universal robots® ur5. desarrollo de programación," 7 2020, tesis de pregrado en Ingeniería en Electrónica Industrial y Automática, Universidad de Valladolid, Valladolid, España.

[26] J. Vicente, I. Campos, B. Sanz, A. Rodríguez, J. Oñate, and J. Sabater, "Ejemplo de integración de alexa con un robot ur," in *XL Jornadas de Automática*. Universidade da Coruña, Servizo de Publicacións, 2019, pp. 360–365, doi: 10.17979/spudc.9788497497169.360.

[27] U. Robots, "Data sheet ur3e," Último acceso 06-03-2023 a https://www.universal-robots.com/media/1807464/ur3e-rgb-fact-sheet-landscape-a4.pdf, 2021.

[28] ——, "Ur3e robot," Último acceso 06-03-2023 a https://www.universal-robots.com/es/productos/robot-ur3/, 2023.

[29] U. Technologies, "Plataforma de desarrollo en tiempo real de unity," Último acceso 06-03-2023 a https://unity.com/es, 2023.

[30] A. Koubâa, *Robot Operating System (ROS) The Complete Reference*. Springer, 2016, vol. 1.

[31] "Ros - basic concepts," Último acceso 06-03-2023 a http://wiki.ros.org/ROS/Concepts, 2023.

[32] A. Koubâa, *Robot Operating System (ROS) The Complete Reference*. Springer Cham, 2021, vol. 6, doi: 10.1007/978-3-030-75472-3.

[33] Y. Pyo, H. Cho, L. Jung, and D. Lim, *ROS Robot Programming (English)*. Robotics, 2017.

[34] E. Peña, "Interfaz inmersiva para misiones robóticas basada en realidad virtual," 7 2017, tesis de pregrado en Ingeniería en Tecnologías Industriales, Universidad Politécnica de Madrid, Madrid, España.

[35] "Ros: Rosbridge_suit," Último acceso 06-03-2023 a https://github.com/UniversalRobots/Universal_Robots_ROS_Driver, 2023.

[36] Siemens, "Ros-sharp," Último acceso 06-03-2023 a https://github.com/siemens/ros-sharp, 2023.

[37] "Universal robots: Ursim," Último acceso 06-03-2023 a https://www.universal-robots.com/download, 2023.

[38] F. Vargas, A. Vivas, and V. Muñoz, "Manipulación del robot ur3 mediante ros y ursim," *XVIII Semana Nacional de Ingeniería Electrónica, SENIE 2022, Ciudad de México, México*, 2020.

[39] "Github - urcaps : External control," Último acceso 06-03-2023 a https://github.com/UniversalRobots/Universal_Robots_ExternalControl_URCap, 2023.

[40] R. Rojas, M. Garcia, L. Gualtieri, E. Wehrle, E. Rauch, and R. Vidoni, *Automatic Planning of Psychologically Less-Stressful Trajectories in Collaborative Workstations: An Integrated Toolbox for Unskilled Users*, 09 2020, pp. 118–126, doi: 10.1007/978-3-030-58380-4_15.

[41] "Github - ur_robot_driver: Universal robots ros driver supporting cb3 and e-series," Último acceso 06-03-2023 a https://github.com/UniversalRobots/Universal_Robots_ROS_Driver, 2023.

[42] "Github - ros industrial - universal robots," Último acceso 06-03-2023 a https://github.com/ros-industrial/universalrobot, 2023.

[43] "Ros - xacro," Último acceso 06-03-2023 a https://wiki.ros.org/xacro, 2023.

[44] R. Parak, "A digital-twins in the field of industrial robotics integrated into the unity3d development platform," Último acceso 06-03-2023 ahttps://github.com/rparak/Unity3D_Robotics_Overview, 2021.

[45] Wikibooks, "Cg programming/unity/hermite curves," Último acceso 06-03-2023 a https://en.wikibooks.org/wiki/Cg_Programming/Unity/Hermite_Curves, 2020.

[46] Unity, "Line renderer component," Último acceso 06-03-2023 a https://docs.unity3d.com/Manual/class-LineRenderer.html, 2023.

[47] "Github - pose based cartesian traj controller causes robot to flip around and sometimes hit joint limits," Último acceso 06-03-2023 a https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/issues/540, 2023.

[48] "Github - cartesian ros controllers," Último acceso 06-03-2023 a https://github.com/UniversalRobots/Universal_Robots_ROS_controllers_cartesian, 2023.

[49] "Github - verificación de signos para garantizar ruta más corta," Último acceso 06-03-2023 a https://github.com/UniversalRobots/Universal_Robots_ROS_controllers_cartesian/pull/14, 2023.

[50] M. Giannaccini, I. Georgilas, I. Horsfield, B. Peiris, A. Lenz, A. Pipe, and S. Dogramadzi, "A variable compliance, soft gripper," *Autonomous Robots*, vol. 36, no. 1, pp. 93–107, 2014, doi: 10.1007/s10514-013-9374-8.

[51] J. Gómez, A. Santarossa, H. Bustos, and L. Pugnaloni, "Effect of the granular material on the maximum holding force of a granular gripper," *Granular Matter*, vol. 23, no. 1, pp. 1–6, 2020, doi: 10.1007/s10035-020-01069-z.

[52] J. Amend, E. Brown, N. Rodenberg, H. Jaeger, and H. Lipson, "A positive pressure universal gripper based on the jamming of granular material," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 341–350, 2012, doi: 10.1109/TRO.2011.2171093.

[53] S. Fitzgerald, G. Delaney, and D. Howard, "A review of jamming actuation in soft robotics," *Actuators*, vol. 9, no. 4, p. 104, 2020, doi: 10.3390/act9040104.

[54] E. Brown, N. Rodenberg, J. Amend, A. Mozeika, E. Steltz, M. Zakin, H. Lipson, and H. Jaeger, "Universal robotic gripper based on the jamming of granular material," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 107, 10 2010, doi: 10.1073/pnas.1003250107.

[55] J. Shintake, V. Cacucciolo, D. Floreano, and H. Shea, "Soft robotic grippers," *Advanced Materials*, vol. 30, p. 1707035, 05 2018, doi: 10.1002/adma.201707035.

[56] "Github - conflicto con la implementación de nuget.newtonsoft.json de unity," Último acceso 06-03-2023 a https://github.com/siemens/ros-sharp/issues/310, 2023.

[57] T. Kitamura, "Fusion2urdf," Último acceso 06-03-2023 a https://github.com/syuntoku14/fusion2urdf, 2020.