Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática Industrial 21 (2024) 52-61

www.revista-riai.org

Generador 3D de trayectorias libres de colisiones para un manipulador UR3e con pinza blanda

Juan Sebastián Montenegro-Bravo[a,*], Juan David Ruiz-Flórez[a], Juan David Romero-Ante[b], Juliana Manrique-Córdoba[b], Oscar Andrés Vivas-Albán[a], José Maria Sabater-Navarro[b]

[a]*Departamento de Electrónica, Instrumentación y Control, Facultad de Ingeniería Electrónica y Telecomunicaciones, Universidad del Cauca, Calle 5 No. 4-70 Popayán, Colombia.*

[b]*Departamento de Ingeniería de Sistemas y Automática, Escuela Politécnica Superior de Elche, Universidad Miguel Hernández, Avda Universidad s/n, Elche, España.*

To cite this article: Montenegro-Bravo, J.S., Ruiz-Flórez, J.D., Romero-Ante, J.D., Manrique-Córdoba, J., Vivas-Albán, O.A., Sabater-Navarro, J.M. 2024. 3D collision-free trajectory generator for a UR3e manipulator with soft gripper. Revista Iberoamericana de Automática e Informática Industrial 21, 52-61. https://doi.org/10.4995/riai.2023.19332

Resumen

Las operaciones pick-and-place son las más frecuentes en aplicaciones robóticas, y muchas veces su diseño incluye la presencia de obstáculos. En este trabajo se presenta la construcción de una plataforma software que permite la manipulación de un robot colaborativo UR3e mediante la generación de trayectorias 3D fácilmente definidas por el usuario, además de una de pinza blanda capaz de sujetar objetos con diferentes geometrías. Para ello, se detalla el desarrollo de una interfaz gráfica en Unity, así como la incorporación del gemelo digital del robot UR3e. Del mismo modo, se exponen los diferentes módulos que permiten la comunicación de la plataforma con el manipulador a través de ROS. Los resultados muestran la creación de rutas adaptadas por el usuario ante diferentes casos en zonas de colisión y la disposición de la pinza para sujetar diferentes objetos. También se compara el error de precisión entre los datos enviados respecto a los que se reciben desde el robot durante el seguimiento de trayectorias definidas por el usuario.

Palabras clave: Planificación de trayectorias, pinzas blandas, ROS, robots colaborativos, ruta libre de colisión, manipulación robótica.

3D collision-free trajectory generator for a UR3e manipulator with soft gripper

Abstract

Pick-and-place operations are the most common in robotic applications, and often their design involves the presence of obstacles. This paper presents the development of a software platform that (allows-enables) the manipulation of a collaborative robot UR3e through the generation of 3D trajectories easily defined by the user, as well as a soft gripper capable of gripping objects with different geometries. For this purpose, the development of a graphical interface in Unity is detailed, as well as the incorporation of the digital twin of the UR3e robot. In the same way, the different modules that allow the communication of the platform with the manipulator through ROS are exposed. The results show the creation of user-adapted paths for different cases in collision zones and the arrangement of the gripper for gripping different objects. The accuracy error between the data sent and received from the robot during the tracking of user-defined trajectories is also compared.

Keywords: Path-planning, soft grippers, ROS, collaborative robots, free collision path, robotic manipulation.

*Autor para correspondencia: exlogam@unicauca.edu.co

1. Introducción

Los robots colaborativos o comúnmente denominados cobots, término que surge de la unión de las palabras colaboración y robot a finales de los años noventa (Peshkin and Colgate, 1999), son muy utilizados hoy en día en diferentes campos y con múltiples aplicaciones. Dentro de las más comunes están los procesos de *pick-and-place*, es decir, tomar objetos de un punto y llevarlos a otro. Aunque esto parece algo relativamente sencillo, detrás de ello hay una serie de implicaciones a tener en cuenta; como la manera de programar la trayectoria, la existencia de obstáculos estáticos o dinámicos, el tipo de robot a utilizar, la tarea a cumplir, los objetos que se manipularán, entre otros más.

En primer lugar, los cobots se caracterizan por incorporar mecanismos de seguridad soportados en normas o estándares internacionales, lo cual les permite el trabajo cooperativo con los humanos (Farsoni et al., 2019). Sin embargo, la planificación de movimientos en robots manipuladores presenta cierta dificultad cuando existen obstáculos, el robot podría entrar en colisión en caso de error (García et al., 2010), bien sea con objetos del entorno o con el trabajador humano.

Para el buen funcionamiento de los manipuladores robóticos es muy importante la planificación óptima del movimiento, con el fin de generar una trayectoria que pueda satisfacer el objetivo (Aguiar and Hespanha, 2004). Un planificador de trayectorias útil, debe ser capaz de generar rutas libres de colisiones, lo suficientemente rápido como para permitir reacciones eficientes a los cambios en el objetivo del entorno o en el estado del robot (Sanchez et al., 2019).

Los algoritmos basados en inteligencia artificial, si bien son utilizados en algunos casos, pueden presentar dificultades para la planificación de trayectorias dado que ofrecen soluciones infinitas para rutas libres de colisiones. Por otra parte, en los casos donde hay gran cantidad de objetos en el espacio de trabajo, aumenta el consumo de recursos computacionales puesto que se deben trazar rutas de alta complejidad para su evasión; es decir, a medida que el entorno en el que opera un robot se vuelve más caótico, por la cantidad de objetos presentes, el problema al que se enfrenta el manipulador se vuelve más complejo, generando restricciones que deben ser consideradas y respetadas para evitar una posible colisión (López et al., 2010).

El uso de bandas elásticas es una de las técnicas que se han implementado para solucionar este problema, donde se hace uso de un camino deformable sin colisión, el cual modifica su trayectoria cuando es sometido a fuerzas artificiales (Quinlan and Khatib, 1993). (Kot et al., 2022) presentan una nueva versión de banda elástica utilizada para encontrar trayectorias suaves, de longitud óptima y libre de colisiones. El algoritmo propuesto sitúa puntos de control sobre la trayectoria, los cuales modifican su posición dinámicamente en reacción a un obstáculo estático o dinámico por medio de fuerzas artificiales de repulsión entre los objetos y el robot; en cambio (Rösmann et al., 2012) presentan una extensión de banda elástica llamada *"timed elastic band"* (TEB), la cual convierte una ruta inicial compuesta de una secuencia de puntos dependiente del tiempo, para generar trayectorias óptimas en tiempo real.

En la actualidad, existen diversos algoritmos de control que implican la detección de objetos para generar trayectorias que eviten colisiones en sistemas robóticos. Estos suelen incorporar cámaras y sistemas sensoriales para garantizar su correcto funcionamiento como algoritmos de aprendizaje profundo. En (Lin et al., 2021) se plantean el uso de DDPG (*Deep deterministic policy gradient*) en conjunto con una red neuronal LSTM (*Long short-term memory*) y el reconocimiento de imágenes, con el fin de identificar el objeto a manipular y los obstáculos que lo rodean. A partir de esta información utilizan el DDPG recurrente propuesto para trazar una ruta libre de colisiones. Así mismo se destaca la herramienta MoveIt!, un software de código abierto con un alto rendimiento que trabaja con ROS y permite planificar movimientos sin colisiones para una amplia gama de sistemas robóticos.

No obstante, el uso de las herramientas mencionadas puede ser un reto para aquellas personas sin conocimientos avanzados en robótica, ya que se requiere tener entendimientos previos sobre cinemática, planificación de trayectorias y programación con ROS, generando un curva de aprendizaje lenta para investigadores novatos (Vivas and Sabater, 2021). O simplemente el investigador desea probar rápidamente determinadas trayectorias complejas, por lo que una alternativa es la representación visual del comportamiento de los manipuladores por medio de una plataforma digital interactiva.

A pesar de que Unity no está diseñado propiamente para la robótica, posee cierta flexibilidad que lo convierten en un entorno óptimo, no solo para producir videojuegos, sino para crear simuladores y escenarios virtuales que contribuyen al desarrollo de diferentes campos como la medicina, la industria o la educación. En (Potkonjak et al., 2016) se presenta una revisión literaria de varios laboratorios virtuales creados a partir de Unity con un enfoque educativo. También se han explorado interfaces basadas en realidad virtual para el control de manipuladores robóticos como lo presenta (Wonsick and Padir, 2020), donde se realiza una clasificación de este tipo de interfaces. Adicionalmete, Unity posee un componente llamado *Line Renderer* y su función básica es crear una línea recta a partir de un conjunto de puntos, que contienen una coordenada en el espacio tridimensional. Esta línea posee una serie de características que se pueden modificar tales como el color, grosor, textura, entre otros. Por lo tanto, este componente permite dibujar en el escenario de Unity desde una línea recta hasta una más compleja como una espiral, lo cual la hace perfecta para trazar rutas.

Es por ello que en este trabajo se destaca el uso de Unity para construir un escenario virtual que contiene el gemelo digital del robot UR3e, así como la interfaz gráfica que permite la visualización de movimientos e interacción del usuario con el robot real a partir de 3 modos de control: articular, cartesiano y libre. Se hace especial énfasis en este último, donde el usuario puede generar trayectorias y deformarlas con el mouse del computador, con el fin de que el manipulador la pueda seguir y que evite colisionar con un objeto en un entorno de trabajo conocido.

Los sistemas de planificación y el sistema de actuación del robot deben trabajar en conjunto. En operaciones de pick-and-place de objetos con geometrías complejas, las pinzas comunes pueden tener dificultades debido a su área de contacto reducida y a los puntos de agarre limitados (Yeong et al., 2022). Para abordar estas limitaciones, se han desarrollado las pinzas universales, que se adaptan a la forma del objeto y tienen una mayor área y puntos de contacto (Giannaccini et al., 2014). Estas se dividen en dos tipos: activas y pasivas (Pham and Yeo, 1991). Las uni-

versales activas tienen diseños antropomórficos complejos y un control sofisticado y costoso, mientras que las pasivas constan de un cuerpo elástico que se deforma al entrar en contacto con el objeto. La fabricación de estas pinzas puede requerir procesos como el modelado, ensamblaje y fusión en cera, pero también pueden ser impresas en multimaterial.

Dentro de las pinzas universales pasivas, destaca la subcategoría de interferencia granular (Fitzgerald et al., 2020), las cuales se basan en la transición de materiales granulares de baja densidad a un cuerpo rígido de alta densidad (Gómez et al., 2020). De los diferentes materiales granulares probados, el café molido es el más utilizado en este tipo de pinzas. Además, el sistema óptimo de fabricación para las pinzas de interferencia granular emplea una presión negativa y positiva para modular la transición de interferencia (Amend et al., 2012). En este trabajo se realiza la construcción e implementación de este tipo de pinza con el fin de realizar la manipulación de elementos de distinta geometría en operaciones de pick-and-place.

2. Componentes del sistema desarrollado

En esta sección se presentan una descripción de los elementos más relevantes que conforman el proyecto desarrollado. Estos componentes abarcan el hardware, el software y las técnicas lógicas utilizadas para la construcción del sistema, así mismo se detalla la función que cumple cada uno.

2.1. Robot UR3e

En esta investigación, se utilizó por disponibilidad el brazo robótico colaborativo llamado UR3e (Universal Robots, 2021), fabricado por Universal Robots. Este robot es conocido por su tamaño reducido, lo que lo hace adecuado para trabajar en espacios pequeños. Tiene seis grados de libertad y las articulaciones de la muñeca pueden girar 360°. Las dos principales ventajas del UR3e son su flexibilidad y su colaboración segura. La flexibilidad se refiere a la capacidad de personalizar algoritmos o interfaces para controlar el robot en tareas específicas, mientras que la colaboración segura se basa en un sistema de seguridad que utiliza sensores de proximidad para evitar colisiones y mantener un ambiente de trabajo seguro (Vicente et al., 2019). Además este robot dispone de una tableta intuitiva (Teach Pendant) que permite programar tareas básicas.

2.2. Robot Operating System (ROS)

ROS es un middleware (lógica de intercambio de información entre aplicaciones) de código abierto para el desarrollo de sistemas robóticos complejos (Koubâa, 2016), que permite crear robots con funciones o ficheros que ya vienen predefinidos, con lo cual se puede reutilizar código y ajustarlo a las necesidades propias. Principalmente se basa en una arquitectura de comunicación mediante canales denominados tópicos, los cuales contienen información de un tipo particular de mensaje. ROS permite trabajar de forma seccionada con distintos subsistemas de un sistema robótico, como controladores, sensores, actuadores, entre otros, para luego poder interconectarlos.

Este trabajo fue desarrollado sobre la distribución de Linux Ubuntu 18.04, junto con la versión ROS Melodic. Igualmente fue probado en Ubuntu 20.04 con la versión ROS Noetic, donde se puede constatar que su funcionamiento no se ve afectado por el cambio de versiones.

2.2.1. ROS Bridge

El paquete ROS Bridge nace con la intención de extender la funcionalidad de éste en la web, permitiendo establecer comunicación entre sistemas ROS y sistemas que no son propiamente de ROS (por ejemplo, Unity) (Figura 1). Este complemento se divide en dos partes importantes, la primera corresponde a los protocolos que se usan para comunicar los sistemas ROS con los sistemas que no son directamente compatibles, esto se hace mediante objetos en formato *JSON* (JavaScript Object Notation). El segundo componente tiene que ver con la implementación específica, es decir, incluye la obtención y modificación de parámetros (rosapi), las librerías necesarias para el tratamiento de las cadenas *JSON* (rosbridgelibrary), y la capa de transporte de la información (rosbridgeserver). Este paquete permitió generar un puente por medio de un *WebSocket* de conexión persistente entre la plataforma y el robot, donde ambas partes envían datos en cualquier momento con latencias muy bajas (tiempo real).

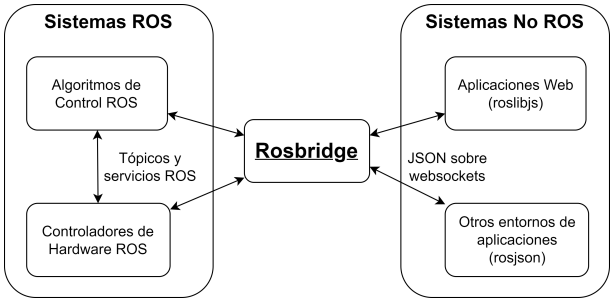


Figura 1: Esquemático del funcionamiento de ROS Bridge.

2.2.2. ROS Sharp

ROS-Sharp o ROS# son un conjunto de librerías y bibliotecas de código abierto en C# creadas por Siemens. Específicamente posibilita la comunicación entre ROS y aplicaciones desarrolladas en .NET. Este paquete requiere la ayuda de ROS Bridge para establecer dicha comunicación, ya que este actúa como puente entre el hardware y software (Robot y Unity 3D respectivamente), donde encapsula la información proveniente de Unity para que esta llegue a los tópicos que permiten enviar ordenes al robot y viceversa (Figura 2). Se ha utilizado este paquete dentro del proyecto ya que facilita comunicar la plataforma de Unity con ROS, además de que contiene ficheros en C# ya prescritos de distintos tipos de mensajes ROS que se pueden utilizar de manera sencilla. Este complemento se encuentra disponible en *Unity Asset Store* de manera gratuita.

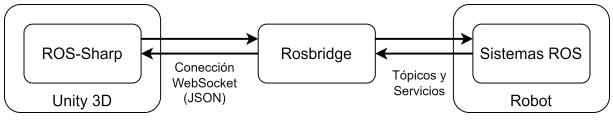


Figura 2: Implementación de ROS-Sharp.

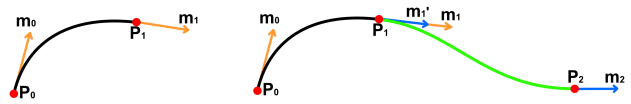
2.3. Unity

Es un motor de desarrollo creado principalmente para la realización de videojuegos en dos y tres dimensiones, permitiendo también crear aplicaciones en realidad virtual y realidad aumentada. Cuenta con grandes capacidades y funciones integradas, como los motores gráficos, los motores físicos, y los efectos de

sonido e iluminación, utilizando C# como lenguaje de programación. Es compatible con múltiples plataformas (Windows, Linux, Android, iOS, etc.) y diversos dispositivos (web, smartphones, consolas, ordenadores, etc.).

2.4. *Curvas de Hermite*

Una curva cúbica de Hermite define un tipo de línea a partir de dos puntos (P_0 y P_1) y dos tangentes (m_0 y m_1). La línea comienza en el punto P_0 inicial en dirección de la tangente m_0 y cambia de rumbo en dirección a la tangente m_1 hasta llegar al punto final P_1 (Figura 3(a)). Por otro lado, es posibles conectar dos o más curvas de Hermite, formando una spline cúbica de Hermite, esto se logra uniendo el vector tangente final de una curva con el vector tangente inicial de la curva siguiente (Figura 3(b)). Una ventaja de las splines de Hermite es que su línea pasa por los puntos de control (P_n), a diferencia por ejemplo de Spline de Bezier, donde no sucede esto. En este trabajo se implementa la spline de Hermite para ilustrar las trayectorias y permitir la modificación de la ruta del manipulador mediante el movimiento de los puntos de control.



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

Figura 3: Curva y spline de Hermite.

2.5. *Unified Robot Description Format (URDF)*

Este archivo en lenguaje XML que se compone principalmente de enlaces y uniones, define la geometría, las mallas visuales, los atributos cinemáticos y dinámicos de un sistema robótico. En conjunto con ROS, URDF permite modelar robots para su simulación y análisis.

Este formato es utilizado para describir virtualmente tanto el robot UR3e como la pinza, para ser incluidos en la plataforma. Para lograrlo se recurre a componentes como *URDF-Importer* y *Fusion2URDF* (El primero permite importar cualquier URDF a un proyecto de Unity y el segundo permite exportar en este tipo de formato las piezas diseñadas en Fusion 360).

Universal Robots proporciona la descripción de sus robots en formato *xacro (ur3e.xacro o ur3e.urdf.xacro)*, por lo que para importar el modelo URDF a Unity se realizó la conversión o supresión de la última extensión *(.xacro)* del archivo. Para lograr esto, una vez ubicado desde una terminal en la ruta donde se encuentra el archivo, se corre la siguiente instrucción:

$$\$\ rosrun\ xacro\ xacro\ model.xacro > model.urdf$$

3. Desarrollo del proyecto

Esta sección contempla una descripción detallada del proceso de construcción y funcionamiento de la plataforma. Aquí se expone la construcción del entorno en Unity, los modos de control creados, la arquitectura interna del proyecto y la red de comunicación con el robot.

La plataforma desarrollada se encuentra estructurada en tres bloques principales. El primer bloque, elaborado en Unity, corresponde a la interfaz gráfica que posibilita la interacción con el usuario. El segundo bloque realiza el procesamiento los de datos y la selección de control, se basa en archivos de Python. Finalmente, el último bloque lo conforma el robot físico que se va a manipular. Cabe aclarar que la comunicación entre bloques está basada en tópicos de ROS.

3.1. *Comunicación de la plataforma*

Para establecer la comunicación entre la plataforma de Unity con los programas de Python y el robot UR3e se utiliza ROS. Debido a que Unity no es un sistema ROS, es necesario adicionar el paquete de ROS# en el proyecto. Para lograr esto se descarga el paquete desde el repositorio de Siemens (Siemens, 2023) y se guarda dentro del *asset* del proyecto de Unity, o bien se puede descargar gratuitamente desde el *Unity Assets Store*. Este paquete de ROS-Sharp ofrece varias funciones dentro de las cuales se usaron la generación de mensajes de tipo ROS en el lenguaje de C#, las librerías que permiten la creación de publicadores y suscriptores, y algunos códigos de ejemplos ya prescritos. Dentro de estos últimos se encuentra un fichero que en conjunto con ROS-Bridge realiza la comunicación mediante un *WebSocket* usando la dirección IP del computador que inicializa ROS. Para que la interfaz sea ejecutable en cualquier ordenador o cualquier red, se crea una ventana de conexión (Figura 4) que contiene un cuadro de texto y permite al usuario especificar su dirección IP.

Figura 4: Panel de conexión.

3.2. *Escenario de Unity y gemelo digital*

La interfaz gráfica cuenta con un escenario virtual, en el se ubica un gemelo digital que copia los movimientos del robot en tiempo real. También cuenta con un tablero lateral que despliega los 3 modos que permiten mover el robot (articular, cartesiano o trayectoria libre).

El gemelo digital del robot se crea para que el usuario pueda observar los movimientos del manipulador en la escena virtual de Unity. En este caso se obtiene el modelo URDF del robot UR3e desde el repositorio oficial de Universal Robots (Universal Robots, 2023), el cual se agrega dentro del *asset* del proyecto en Unity. Luego, con ayuda del componente *URDF-Importer*, se importa el archivo del robot en formato URDF, el cual al finalizar crea un objeto en la jerarquía del editor con los componentes del robot y todas sus características cinemáticas, así como también construye el robot digital en el centro de la escena. Este mismo

proceso de importación se utiliza para agregar el URDF de la pinza y acoplarla al robot digital.

Una vez se tiene el robot en la escena, se agrega a cada componente de articulación el archivo prescrito */JointStateWrite* del paquete ROS-Sharp. Este archivo permite modificar el valor de la articulación del gemelo digital con respecto a un valor recibido. Este valor es dado por un fichero suscriptor, que se suscribe un tópico que publica las posiciones articulares del manipulador físico para que el robot digital copie los movimientos en tiempo real. Posteriormente, se agregan más objetos a la escena para crear así un laboratorio virtual que permitirá trabajar con el gemelo digital y con el robot real.

3.3. *Control articular*

A continuación, se crea el tablero articular que contiene seis deslizadores que representan las seis articulaciones del robot, y seis casillas que muestran los valores numéricos de las mismas (Figura 5). El rango de cada slider o deslizador va desde -360 a 360 unidades. Al conectar la plataforma con el robot real mediante ROS, los deslizadores toman el valor en grados *RPY* de las articulaciones del robot. Si un deslizador es arrastrado a una posición diferente a la actual, automáticamente se activa una función dentro de un fichero publicador, el cual realiza la conversión de unidades y envía mediante un mensaje ROS los nuevos valores al controlador del robot. Así el controlador mueve simultáneamente cada articulación hasta alcanzar el valor indicado.

Así mismo, se crea otro tablero que permite listar varias posiciones articulares. Para lograr esto se diseña un *prefab* de casillas de texto con valores nulos. Al agregar un elemento a la lista, se genera una instancia del *prefab* y se modifica con los valores actuales de los deslizadores, permitiendo así crear un conjunto de posiciones que serán ejecutadas secuencialmente con una velocidad media preestablecida internamente. El propósito de este modo de control consiste en situar al manipulador en una postura especifica, así como desplazarlo rápidamente de una posición a otra, sin necesidad de garantizar gran precisión en la ubicación.

Figura 5: Ventana del control articular.

3.4. *Control cartesiano*

De forma similar se crea el tablero cartesiano, el cual contiene seis casillas de entrada y seis casillas de salida (Figura 6). Las primeras permiten ingresar los valores de la posición y rotación deseadas del efector final en los 3 ejes (XYZ), mientras que las segundas muestran la posición y rotación actuales. Cabe aclarar que la posición está dada en *cm* y la rotación en grados *RPY*. Al introducir una coordenada y presionar la opción "mover", se invoca una función de un fichero publicador, la cual envía los valores de la coordenada escrita al controlador del robot para que el efector final se dirija hasta el punto indicado. Otra característica de este tablero es que cuenta con un deslizador que indica la velocidad en cm/s con la que el efector final se moverá entre coordenadas. Este deslizador permite modificar la velocidad en un rango de $0{,}1cm/s$ a $10cm/s$.

Este modo también posee otro tablero que permite crear una lista de puntos para generar una trayectoria cartesiana. Para ello, se utiliza un *prefab* para los *ítems* cartesianos, de modo que cuando se adicione una coordenada, se cree una instancia del mismo con los valores ingresados en las casillas de posición y orientación. Posteriormente esta lista de coordenadas puede ser enviada al controlador del robot para que cree la trayectoria pasando por los puntos indicados en la lista, con la velocidad determinada por el deslizador de velocidad.

Figura 6: Ventana del control cartesiano.

3.5. *Control trayectoria libre*

Para este control se crea un tablero llamado trayectoria libre, el cual contiene un botón que crea una línea recta entre las dos esferas en el escenario de la plataforma (Figura 7). Esta línea está construida a partir del componente *Line Rederer*, cuenta con un colisionador de malla que le permite ser interactiva, es decir, que detecta cuando se presiona sobre ella desde el puntero del ratón. En su estado inicial esta línea solo cuenta con el punto inicial y un punto objeto pre-definido. Las esferas pueden trasladarse en el espacio tridimensional, generando que la trayectoria cambie de posición.

Al hacer clic sobre alguna parte de la línea, se creará un nuevo punto de control (*prefab* de esfera), el cual pasará a estar definido por una spline de Hermite, así como cada segmento de la línea, como se muestra en la ecuación (1).

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

Donde t es un parámetro de 0 a 1, P_n son los puntos de control y m_n las pendientes respectivas de cada punto.

Además, para que los segmentos se conecten con suavidad, se utiliza una interpolación *Catmull-Rom* que define los valores de las tangentes de los puntos de control por donde pasará la

línea. Esto se aplica tanto para los puntos intermedios (2) como para el punto inicial (3) y final (4) de la spline de Hermite. De esta forma el usuario podrá agregar tantos puntos de control como lo desee y ubicarlos en diferentes posiciones sobre el escenario, para formar visualmente la trayectoria que efectuará el robot.

$$m_k = \frac{P_{k+1} - P_{k-1}}{2} \tag{2}$$

$$m_{ki} = P_{k+1} - P_k \tag{3}$$

$$m_{kf} = P_k - P_{k-1} \tag{4}$$

El parámetro m_k representa la tangente de los puntos intermedios, mientras que m_{ki} y m_{kf} representan, respectivamente, la tangente del punto inicial y final. Además, la variable P_k representa el punto de control de la spline de Hermite.

Para que el robot realice la ruta planificada en la plataforma, se crea un botón "cargar" que llama una función del fichero publicador y envía al controlador del robot un mensaje ROS con las coordenadas de los puntos que conforman la línea del componente *Line Renderer*.

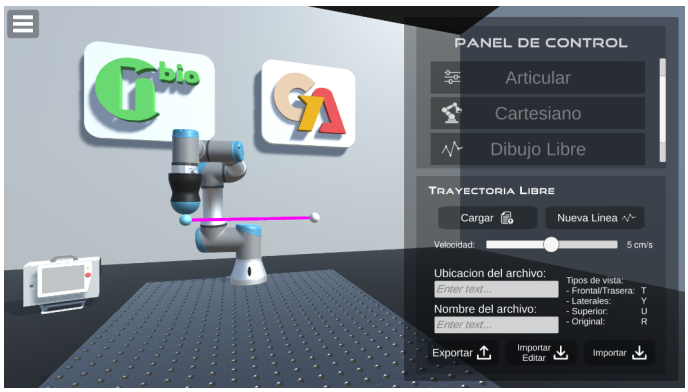


Figura 7: Ventana trayectoria libre.

Este tablero ofrece varias funciones adicionales que amplían sus capacidades. Entre ellas se encuentra un deslizador que permite ajustar la velocidad de desplazamiento del efector final, en un rango de 0,1 cm/s a 10 cm/s, lo que resulta especialmente útil en aplicaciones que requieren una mayor precisión y control. Además, se incluye la opción de exportar la ruta planificada en un archivo JSON portable, lo que facilita su uso y compartición. Los usuarios también pueden importar este archivo para replicar o modificar la trayectoria en cualquier momento y adaptarla a las necesidades específicas de su proyecto.

3.6. Arquitectura

El diagrama de la Figura 8 describe la arquitectura interna del proyecto desarrollado. Este diagrama muestra los tres bloques que componen la plataforma. El primer bloque corresponde a la interfaz gráfica generada en Unity, la cual contiene el gemelo digital junto con los tres modos de control, así como sus respectivos ficheros publicadores y suscriptores.

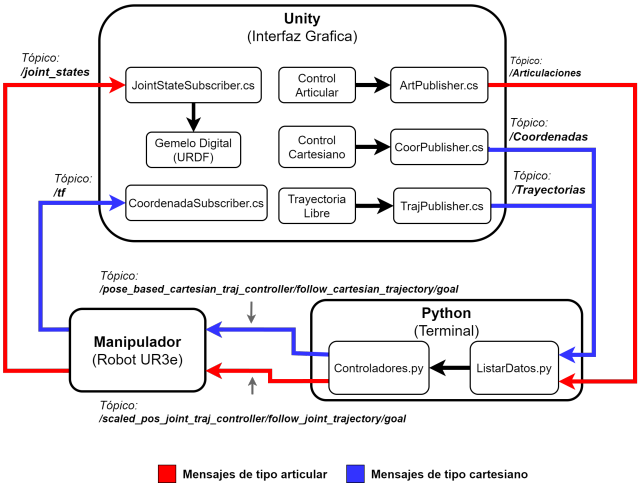


Figura 8: Arquitectura del sistema.

Cuando el usuario elige uno de los modos de operación e ingresa los datos a enviar al robot, cada modo activa su fichero publicador, que se encarga de convertir las unidades de los datos a los estándares de los mensajes ROS, para posteriormente publicar los mensajes con la información en el tópico correspondiente de cada modo, de manera que:

- Si los datos provienen del control articular, el fichero */Artpublisher* convertirá los valores articulares de grados *RPY* a radianes y los enviará a través de ROS al tópico */Articulaciones*, en el tipo de mensaje */JointTrajectory*.
- Si los datos provienen del control cartesiano, el fichero */CoorPublisher* convertirá las posiciones de centímetros a metros y las orientaciones de grados *RPY* a cuaterniones, para luego enviarlos vía ROS al tópico */Coordenadas*, en el tipo de mensaje */CartesianTrajectory*.
- Si los datos provienen de trayectorias libres, el fichero */TrajPublisher* los enviará a través ROS al tópico */Trayectorias*, en el tipo de mensaje */CartesianTrajectory*.

Mientras tanto en el bloque de Python se encuentra el archivo *ListarDatos.py*, el cual es un suscriptor en paralelo de los 3 tópicos ya mencionados. Al detectar un nuevo mensaje en alguno de estos tópicos se recibe la información, se organiza en forma de listas, y se invocan algunas de las funciones preestablecidas del archivo *Controladores.py* con el fin de pasar los datos como parámetros de entrada. Dentro de *Controladores.py* se carga e inicializa el controlador del robot requerido (articular o cartesiano) y se organizan las listas de datos en mensajes de tipo ROS para luego ser enviados a los tópicos que especifica el controlador:

- Si se selecciona el control articular del robot, los datos serán publicados en el tópico */scaled_pos_joint_traj_controller/follow_joint_trajectory_goal* en el tipo de mensaje */FollowJointTrajectoryGoal*.
- Si se selecciona el control cartesiano del robot, los datos serán publicados en el tópico */pose_based_cartesian_traj_controller/follow_cartesian_trajectory_goal* en el tipo de mensaje */FollowCartesianTrajectoryGoal*.

De esta manera, el controlador calcula internamente la trayectoria a realizar con el fin de alcanzar las posiciones o coordenadas especificadas desde la interfaz a la velocidad indicada. Luego, el robot real se desplaza según la trayectoria planificada y retroalimenta el movimiento, publicando su posición articular y cartesiana en tiempo real en los tópicos */joint_states* y */tf* respectivamente. El nodo de Unity, a su vez, se suscribe a estos tópicos a través de los ficheros suscriptores *JointStateSuscriber* y *CoordenadaSuscriber*. El primero permite que el gemelo digital copie los movimientos y se puedan visualizar los valores en el panel articular, mientras que el segundo muestra las coordenadas del efector final en el panel cartesiano.

3.7. *Conexión de la plataforma*

La comunicación del robot y plataforma software se caracteriza por dos nodos principalmente: el nodo de la plataforma software que envía comandos de movimiento, y el nodo del manipulador UR3e que recibe todas las órdenes enviadas.

El envío de datos se realiza por medio del driver *ur_robot_driver* de ROS, que se ejecuta por una orden de comando desde consola indicando la dirección IP del robot, y que lanza tanto nodos, tópicos y servicios que facilitan conocer el estado del mismo, así como los cálculos cinemáticos, los cuales posibilitan los movimientos articulares y cartesianos para que el manipulador se desplace a la posición deseada.

El usuario estable conexión con el robot y la plataforma por medio de la dirección IP del ordenador que ejecuta ROS (Figura 9).

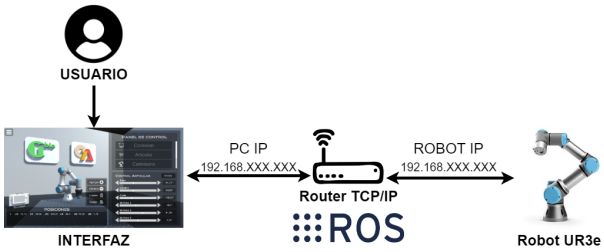


Figura 9: Conexión de la plataforma software con el robot UR3e.

3.8. *Desarrollo de la pinza*

Para poder manipular objetos con el robot real a partir de las órdenes enviadas desde la plataforma, se diseñó y construyó una pinza de interferencia granular. Para su construcción se realizó una revisión bibliográfica para lograr un diseño óptimo.

El diseño de la pinza granular implementada en este trabajo consta de varias partes (Figura 10). La primera es el cuerpo de la pinza, desarrollado en el software de Fusión 360 y creado en impresión 3D. En uno de sus extremos tiene un acople de rosca para sujetar la pinza al robot, en el lado opuesto tiene un globo de látex que internamente está lleno de café molido. El globo se ubica entre el cuerpo de la pinza y el acople del filtro, para así crear un sello hermético. El filtro neumático se encaja con el acople del filtro y evita que el material granular sea introducido en el sistema de succión.

El funcionamiento de la pinza de interferencia granular consiste en ubicar la pinza sobre el objeto, presionarla contra el mismo aplicando la denominada fuerza de activación, que hace que la pinza se envuelva alrededor del objeto adquiriendo su forma. Seguidamente se emplea una presión negativa para que el material granular interno se compacte y forme un cuerpo rígido que no se deforma. Para soltar el objeto se aplica una presión positiva a la pinza, para que la bola de látex se expanda soltando el objeto y el material granular vuelva a su estado inicial de baja densidad.

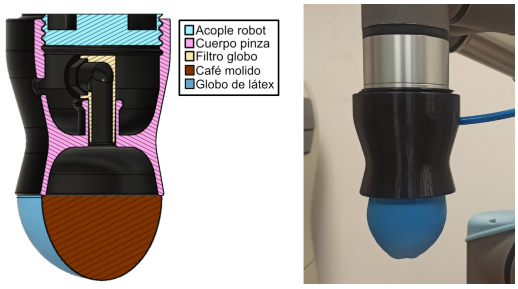


Figura 10: Pinza de interferencia Granular.

4. Resultados

Como resultado del trabajo aquí documentado se obtuvo la elaboración de una plataforma que permite controlar el robot UR3e a través de distintos modos, estos han sido validado mediante las siguientes pruebas.

4.1. *Prueba de precisión*

La prueba de precisión consistió en situar manualmente el efector final del robot sobre seis puntos de referencia, definidos en el espacio tridimensional mediante marcadores sobre una mesa plana. Estos marcadores estaban organizados en dos columnas con una separación de 7.5 cm y tres filas con una separación de 6 cm (Figura 12). Para obtener los datos del manipulador sobre cada punto, se utilizó el tópico */tf*, que proporciona las posiciones y orientaciones del efector final. Posteriormente, se ingresaron en el panel de control cartesiano las seis coordenadas registradas para que el robot alcanzara las mismas posiciones mediante la trayectoria descrita en la (Figura 11). Esta trayectoria es calculada por el controlador del robot a partir de los puntos que se envían. Para validar lo anterior, se capturaron los valores de salida de la plataforma desde el tópico */Coordenadas*, y se escuchó nuevamente el tópico */tf* para tomar las nuevas posiciones.

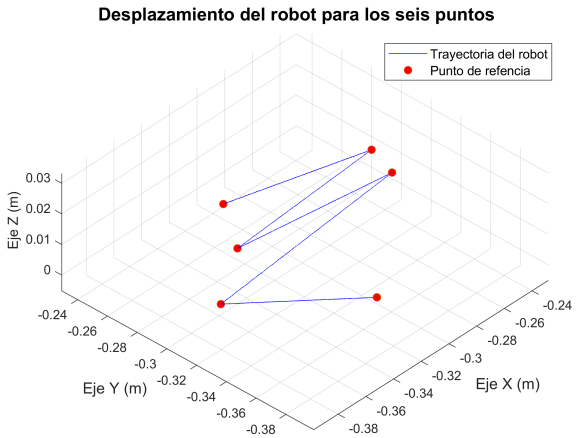


Figura 11: Trayectoria del robot para los seis puntos ingresados.

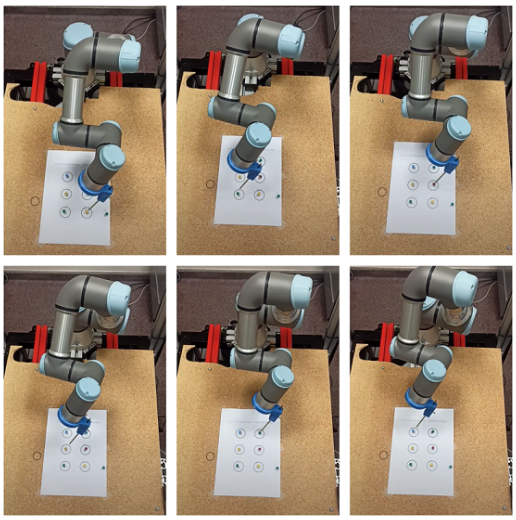


Figura 12: Toma de los puntos de referencia.

Los datos obtenidos se organizaron en una tabla de Excel y se compararon en un algoritmo de Matlab para hallar el error máximo mediante medición euclidiana. Los resultados mostraron un error de $4{,}43x10^{-06}m$ para la posición y $5{,}05x10^{-4}rad$ para la rotación (Figura 13).

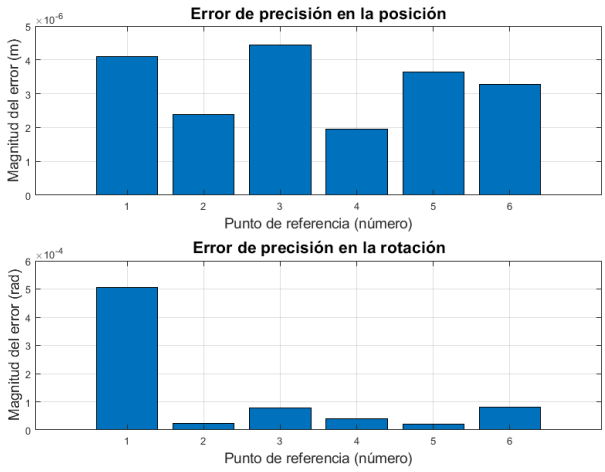


Figura 13: Error de precisión en la posición y rotación.

Es esencial tener en cuenta que, para lograr una evaluación exhaustiva del error, se requiere utilizar un sistema de medición externo al robot. Esto se puede lograr mediante el uso de dispositivos de visión o medición láser, que permitan comparar la señal deseada con la señal adquirida por dicho sistema de medición externo.

4.2. *Prueba trayectoria libre*

Se llevó a cabo una prueba para evaluar el control de trayectoria libre, la cual consistió en tres casos posibles. El propósito 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 objetivo, se empleó una ruta inicial en forma de línea recta, la cual se adaptó en función de cada caso. Cada ruta dibujada en los tres casos, estaba conformada por un conjunto de puntos cartesianos que variaba según el numero de segmentos.

Para determinar la precisión de la ruta planificada en la plataforma en comparación con la ruta 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 de Matlab para calcular esta distancia y se generaron gráficas para cada caso, analizando el error máximo obtenido.

- **Caso I:** En este caso se define la situación de tomar una pinza quirúrgica en una posición inicial determinada, para trasladarla del punto A hasta el punto B, evitando la zona de colisión delimitada por un prisma cuadrado de $19x14x9cm$ (obstáculo 1). Para lograrlo se modificó la ruta hasta el punto de adaptarla a una posición que esté por fuera del área de colisión y que el instrumento quirúrgico pueda ser llevado a su objetivo (Figura 14).

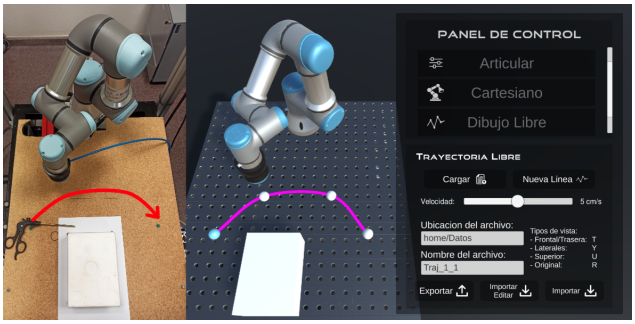


Figura 14: Evasión del obstáculo caso I.

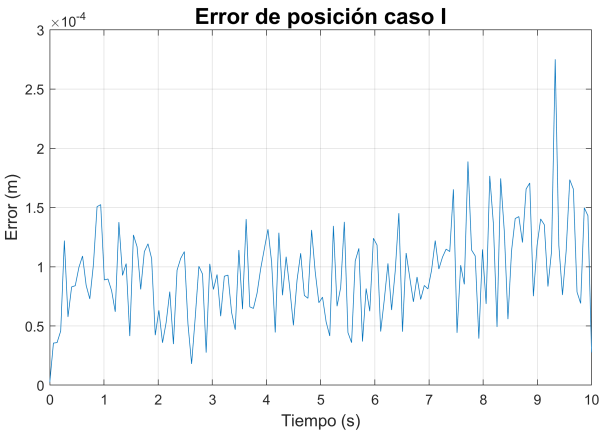


Figura 15: Error de trayectoria caso I.

La ruta utilizada en este caso estaba compuesta por 150 puntos y su trayecto se completó en un tiempo total de 10 *segundos*, a una velocidad constante de 5 cm/s. Con el objetivo de obtener posiciones precisas del robot en su trayectoria real, se realizó un muestreo a una frecuencia de 0.0666 *segundos* utilizando el tópico /tf para obtener las posiciones en cada punto. Posteriormente, se compararon estas posiciones obtenidas con las posiciones enviadas, utilizando la métrica de distancia euclidiana para determinar el error de seguimiento máximo, el cual se registró en $2{,}75x10^{-04}m$ (ver Figura 15).

- **Caso II:** De forma similar al caso anterior se mueve el objeto de un lado al otro, pero en esta ocasión la pinza quirúrgica se rotó 90° sobre su eje, provocando nuevamente una zona de colisión con el obstáculo. Se modificó entonces la trayectoria para evitar que el objeto chocara con el obstáculo (Figura 16).

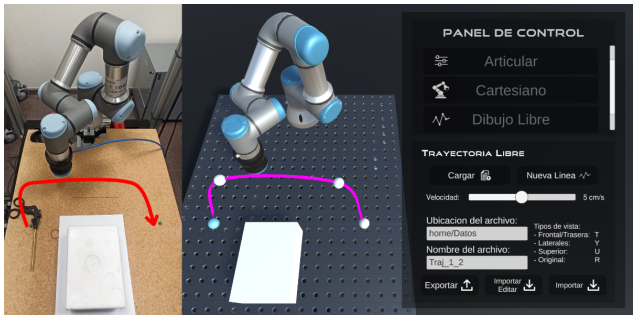


Figura 16: Evasión del obstáculo caso II.

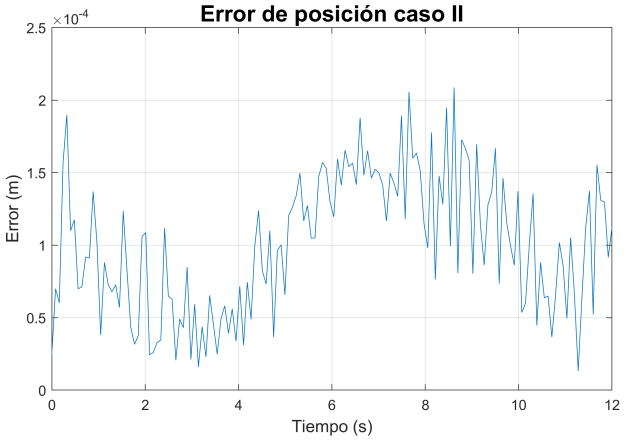


Figura 17: Error de trayectoria caso II.

En este segundo caso, la trayectoria utilizada no presentó modificaciones en cuanto al número de puntos utilizados. Sin embargo, se ha registro un aumento en el tiempo total necesario para completar el recorrido con la misma velocidad anterior, pasando a ser de 12 *segundos*. Debido a este cambio, se modifico el tiempo de muestreo utilizado para la obtención de las posiciones del tópico /tf, estableciéndolo en 0.08 *segundos*. Los resultados obtenidos muestran que el error de precisión más alto en esta ruta modificada fue de $2{,}08x10^{-04}m$, como se puede observar en la Figura 17.

- **Caso III:** Consiste en seguir la misma linea del caso II, pero agregando en esta ocasión un obstáculo adicional de dimensiones $19x6x14cm$ al espacio de trabajo. Se considera que este nuevo obstáculo presenta una zona de colisión en un tramo de la trayectoria anterior, entonces esta se modifica una vez más generando una trayectoria de mayor complejidad que evita la colisión con los dos obstáculos (Figura 18).

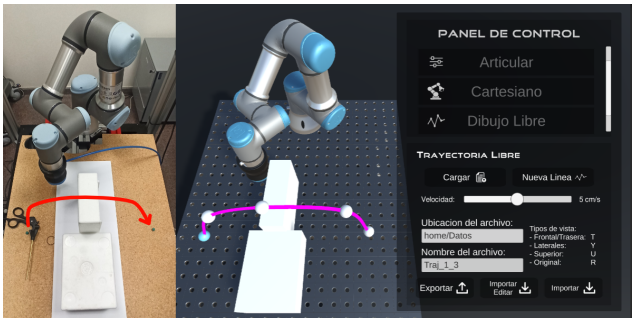


Figura 18: Evasión del obstáculo caso III.

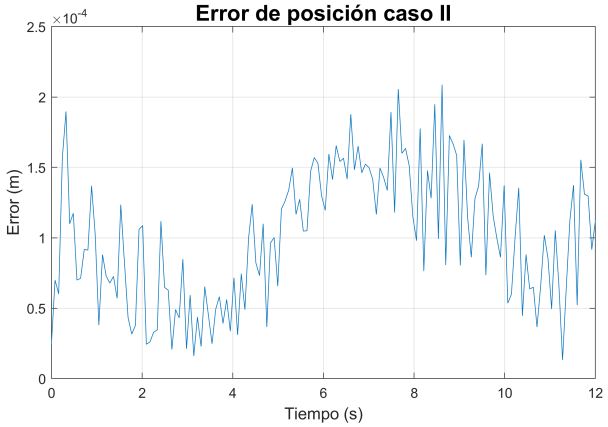


Figura 19: Error de trayectoria caso III.

En este último caso, la ruta planificada consistió en 200 posiciones y su tiempo total de recorrido fue de 13 *segundos* a una velocidad de 5 cm/s, lo que implica que el tiempo de muestreo utilizado para la obtención de los valores de la trayectoria real fue de 0.065 *segundos*. Al tener el mismo número de muestras que en los casos anteriores, se compararon los puntos enviados con la trayectoria efectuada por el manipulador mediante la medición de la distancia euclidiana. Como resultado de estas comparaciones, se registró un error máximo de precisión de $1{,}37x10^{-04}m$, como se puede apreciar en la Figura 19.

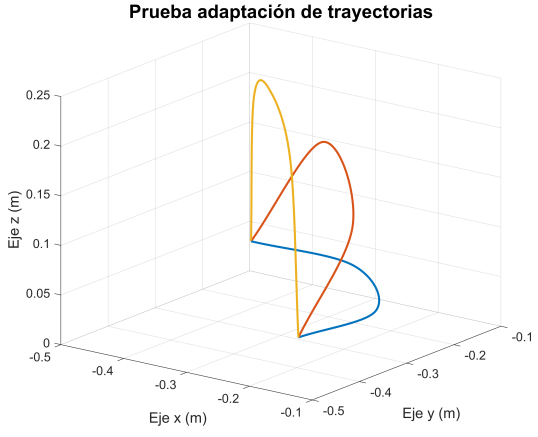


Figura 20: Gráfica de modificación de la trayectoria libre

5. Conclusiones

Este artículo presentó la implementación de una plataforma para manipular el robot UR3e, así como el diseño de una pinza de interferencia granular para su implementación. La plataforma fue construida en el motor gráfico de Unity donde se incluyó el gemelo digital del robot UR3e, el cual es manipulado por tres distintos tableros de control (movimiento articular, movimiento cartesiano o trayectoria libre). Para la comunicación entre la plataforma y el robot se utilizó ROS a través de una conexión por *WebSocket* establecida por ROS Bridge. Además, se construyó una pinza blanda de tipo granular, la cual fue creada con un globo de látex y con café molido en su interior; al ser sometida a una fuerza de succión que compacta los granos del interior, toma la forma del objeto deseado y lo agarra.

Se evaluó la precisión de la plataforma, realizando tres pruebas que medían la diferencia entre los datos deseados enviados desde la plataforma y el movimiento verificado por el robot, encontrándose un resultado aceptable puesto que se mantiene un margen de error pequeño en los tres casos.

Así mismo, se probó la adaptación manual de trayectorias libres para la evasión de obstáculos, donde sin requerir gran esfuerzo fue posible que el robot evadiera las zonas de colisión.

La plataforma propuesta está en la disponibilidad de incorporar cualquier manipulador de Universal Robots de la e-series, en las versiones anteriores como la CB-series se descarta ya que su controlador *ur_modern_driver* está obsoleto.

Trabajos futuros pretenden implementar con gran precisión los objetos del mundo físico dentro del escenario virtual, para de este modo realizar trayectorias quirúrgicas que requieren un mayor grado de complejidad. Además, esta plataforma podría ser escalada no solo a un escenario de pantalla sino a un mundo en realidad virtual, lo que posibilitaría gran número de aplicaciones.

Futuras investigaciones podrían desarrollar escenarios de trayectorias adaptativas en entornos desconocidos utilizando mediciones externas. Esto permitiría una mayor flexibilidad en la planificación de rutas y una mejor capacidad de adaptación a condiciones imprevistas. Además, esta funcionalidad también podría mejorar la seguridad en la operación del robot, ya que permitiría la detección y evasión de obstáculos en tiempo real.

Referencias

Aguiar, A., Hespanha, F., 2004. Logic-based switching control for trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty. In: Proceedings of the 2004 American control conference. Vol. 4. IEEE, pp. 3004–3010.
DOI: 10.23919/ACC.2004.1384369

Amend, J., Brown, E., Rodenberg, N., Jaeger, H., Lipson, H., 2012. A positive pressure universal gripper based on the jamming of granular material. IEEE Transactions on Robotics 28 (2), 341–350.
DOI: 10.1109/TRO.2011.2171093

Farsoni, S., Ferraguti, F., Bonfè, M., 2019. Safety-oriented robot payload identification using collision-free path planning and decoupling motions. Robotics and Computer-Integrated Manufacturing 59, 189–200.
DOI: 10.1016/j.rcim.2019.04.011

Fitzgerald, S., Delaney, G., Howard, D., 2020. A review of jamming actuation in soft robotics. Actuators 9 (4), 104.
DOI: 10.3390/act9040104

García, D., Gómez, F., Martín, J., López, I., Carbone, G., 2010. Algoritmo genético con evaluación borrosa multicriterio: aplicación a la planificación de movimientos en manipuladores paralelos. Universidad de Huelva, pp. 43–48.

Giannaccini, M., Georgilas, I., Horsfield, I., Peiris, B., Lenz, A., Pipe, A., Dogramadzi, S., 2014. A variable compliance, soft gripper. Autonomous Robots 36 (1), 93–107.
DOI: 10.1007/s10514-013-9374-8

Gómez, J., Santarossa, A., Bustos, H., Pugnaloni, L., 2020. Effect of the granular material on the maximum holding force of a granular gripper. Granular Matter 23 (1), 1–6.
DOI: 10.1007/s10035-020-01069-z

Kot, T., Wierbica, R., Oščádal, P., Spurný, T., Bobovský, Z., 2022. Using elastic bands for collision avoidance in collaborative robotics. IEEE Access 10, 106972–106987.
DOI: 10.1109/ACCESS.2022.3212407

Koubâa, A., 2016. Robot Operating System (ROS) The Complete Reference. Vol. 1. Springer.

Lin, G., Zhu, L., Li, J., Zou, X., Tang, Y., 2021. Collision-free path planning for a guava-harvesting robot based on recurrent deep reinforcement learning. Computers and Electronics in Agriculture 188, 106350.
DOI: 10.1016/j.compag.2021.106350

López, D., Gómez, F., Cuesta, F., Ollero, A., 2010. Planificación de trayectorias con el algoritmo rrt. aplicación a robots no holónomos. Revista Iberoamericana de Automática e Informática Industrial (RIAI) 3, 56–67.
DOI: 10.4995/riai.v3i3.8279

Peshkin, M., Colgate, J., 1999. Cobots. Industrial Robot: An International Journal 26 (5), 335–341.
DOI: 10.1108/01439919910283722

Pham, D., Yeo, S., 1991. Strategies for gripper design and selection in robotic assembly. International Journal of Production Research 29 (2), 303–316.
DOI: 10.1080/00207549108930072

Potkonjak, V., Gardner, M., Callaghan, V., Mattila, P., Guetl, C., Petrović, V., Jovanović, K., 2016. Virtual laboratories for education in science, technology, and engineering: A review. Computers & Education 95, 309–327.
DOI: 10.1016/j.compedu.2016.02.002

Quinlan, S., Khatib, O., 1993. Elastic bands: Connecting path planning and control. In: [1993] Proceedings IEEE International Conference on Robotics and Automation. Vol. 2. IEEE, pp. 802–807.
DOI: 10.1109/ROBOT.1993.291936

Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., Bertram, T., 2012. Trajectory modification considering dynamic constraints of autonomous robots. In: ROBOTIK 2012; 7th German Conference on Robotics. VDE, pp. 1–6.

Sanchez, J., Wang, M., Olivares, M., Molina, M., Voos, H., 2019. A real-time 3d path planning solution for collision-free navigation of multirotor aerial robots in dynamic environments. Journal of Intelligent and Robotic Systems 93, 33–53.
DOI: 10.1007/s10846-018-0809-5

Siemens, 2023. Ros-sharp. https://github.com/siemens/ros-sharp, accedido el 29 de abril de 2023.

Universal Robots, 2021. Data sheet ur3e. https://www.universal-robots.com/media/1807464/ur3e-rgb-fact-sheet-landscape-a4.pdf, accedido el 29 de abril de 2023.

Universal Robots, 2023. Ros industrial. https://github.com/ros-industrial/universal_robot, accedido el 29 de abril de 2023.

Vicente, J., Campos, I., Sanz, B., Rodríguez, A., Oñate, J., Sabater, J., 2019. Ejemplo de integración de alexa con un robot ur. In: XL Jornadas de Automática. Universidade da Coruña, Servizo de Publicacións, pp. 360–365.
DOI: 10.17979/spudc.9788497497169.360

Vivas, A., Sabater, J. M., 2021. Ur5 robot manipulation using matlab/simulink and ros. In: 2021 IEEE International Conference on Mechatronics and Automation (ICMA). pp. 338–343.
DOI: 10.1109/ICMA52036.2021.9512650

Wonsick, M., Padir, T., 2020. A systematic review of virtual reality interfaces for controlling and interacting with robots. Applied Sciences 10 (24), 9051.
DOI: 10.3390/app10249051

Yeong, W., Goh, G., Goh, G., Lee, S., Altherr, J., Tan, J., Campolo, D., 2022. 3d printing of soft grippers with multimaterial design: Towards shape conformance and tunable rigidity. Materials Today: Proceedings 70, 525–530.
DOI: 10.1016/j.matpr.2022.09.552