UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Diseño de tarea colaborativa empleando el robot UR3e y realización de pruebas experimentales

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Martín Doñas, Alejandro

Tutor/a: Gracia Calandin, Luis Ignacio

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MÁSTER DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Diseño de tarea colaborativa empleando el robot UR3e y realización de pruebas experimentales

---

Alumno

Alejandro Martín Doñas

Tutor

Luis Ignacio Gracia Calandin

2022-2023

Resumen

En el presente Trabajo de Fin de Máster, se muestra el desarrollo de diversas pruebas experimentales empleando el robot UR3e del fabricante Universal Robots. Este trabajo aborda temáticas propias de la titulación MAII, como son la robótica, informática industrial, visión artificial, etc.

La primera parte del trabajo consiste en la realización de una tarea colaborativa de pick & place entre dos robots colaborativos UR3e. Para la realización de esta parte se usaron elementos del laboratorio como una cinta transportadora, cámara de visión artificial… Además, se hizo uso del robot ABB IRB 140 para cargar un programa cíclico que envíe la información de los sensores y actuadores de la cinta a los robots UR3e.

Posteriormente, se probó el modo fuerza del UR3e haciendo una prueba sencilla en la que uno de los dos robots UR3e sujetaba una pizarra mientras que el otro aplicaba una fuerza constante con un rotulador. Al mover la pizarra, el segundo robot deberá adaptarse para seguir aplicando la fuerza impuesta.

Por último, se ha desarrollado un gemelo digital en Unity con las gafas de realidad virtual Oculus Quest 2 que nos permitirá controlar el robot UR3e en tiempo real en coordenadas cartesianas y articulares mostrándonos todos sus valores a través de una interfaz gráfica y pudiendo visualizar la posición del robot real. Además, se integra un control manual del robot que nos permitirá “agarrar” el robot y moverlo a nuestro gusto.

Palabras clave: Realidad virtual, UR3e, pick & place, PCH (punto central de herramienta), gemelo digital, visión artificial.

Abstract

In the present Master's Thesis, the development of various experimental tests using the UR3e robot from Universal Robots manufacturer is showcased. This work addresses topics related to the MAII degree, such as robotics, industrial computing, computer vision, and so on.

The first part of the project involves the execution of a collaborative pick & place task between two UR3e collaborative robots. For this part, laboratory elements such as a conveyor belt and an artificial vision camera were used. Additionally, the ABB IRB 140 robot was used to load a cyclic program that sends the information from the sensors and actuators of the conveyor belt to the UR3e robots.

Subsequently, the force mode of the UR3e was tested by performing a simple experiment in which one of the two UR3e robots held a whiteboard while the other applied a constant force with a marker. As the whiteboard is moved, the second robot must adapt to continue applying the imposed force.

Lastly, a digital twin has been developed in Unity using the Oculus Quest 2 virtual reality glasses. This digital twin allows us to control the UR3e robot in real time using both cartesian and joint coordinates. It displays all the robot's values through a graphical interface and enables visualization of the position of the actual robot. Additionally, a manual control of the robot is integrated, allowing us to "grab" the robot and move it according to our preference.

Keywords: Virtual reality, UR3e, pick & place, TCP (Tool Center Point), digital twin, computer vision.

Índice

Índice de figuras

1. Introducción

En este capítulo se van a presentar los diferentes objetivos que se intentan buscar con la realización de este Trabajo Fin de Máster así como la motivación que me ha llevado a realizarlo.

1.1. Objetivos

Con la realización de este trabajo se han buscado alcanzar dos objetivos principales. El primero ha sido conseguir realizar un pick & place colaborativo con dos robots UR3e haciendo uso de todos los instrumentos disponibles en el laboratorio.

Para cumplir este objetivo fueron necesarias semanas de documentación y de realizar cursos de la propia Universal Robots con el fin de aprender a programar este tipo de robots. Además, se han probado características como el modo fuerza del mismo y accesorios de otras marcas como la cámara de visión artificial y el uso de una cinta transportadora.

Otro de los principales objetivos de este trabajo fue el realizar un gemelo digital de un robot UR3e con las gafas de realidad virtual Oculus Quest 2 siguiendo principalmente otro trabajo que buscaba este mismo objetivo. [14]

1.2. Motivación

Hace relativamente pocos años se han empezado a implantar en la industria los robots colaborativos para trabajar junto al ser humano. Este tipo de robots eran totalmente desconocidos para mi hasta que nos hablaron de ellos en el máster.

Al ver el laboratorio de robótica y ver que poseían este tipo de robots quise aprender a programarlos e informarme lo máximo posible de los mismos. Esta fue mi principal motivación, ver por primera vez un robot para aprender lo máximo posible durante mi curso académico.

Además, realicé mi Trabajo Fin de Grado con otras gafas de realidad virtual para un estudio médico de la medición de los movimientos oculares para la detección de posibles enfermedades. Esto hizo que pensase que podría aplicar lo aprendido con la realidad virtual en este proyecto y me puse a investigar sobre cómo aplicar esta tecnología en un robot industrial.

Con mis primeras búsquedas me di cuenta de la importancia que está cobrando estos últimos años la implementación de la realidad virtual y aumentada en la industria. Por ejemplo, en una cadena de montaje si se detectan errores estos podrían verse con mayor rapidez con unas gafas de realidad aumentada.

Otro posible uso de la realidad virtual es la posibilidad de operar y programar un robot a distancia con la implementación de un gemelo digital. Como se puede ver, las aplicaciones con esta tecnología son casi ilimitadas lo que despertó mucho mi interés personal.

1.3. Planificación temporal

La realización de todo el proyecto tuvo una duración aproximada de 300 horas que se han repartido entre el estudio del campo de aplicación, la integración de las plataformas de hardware y software, el desarrollo, la extracción y el análisis de los resultados y la redacción de la memoria.

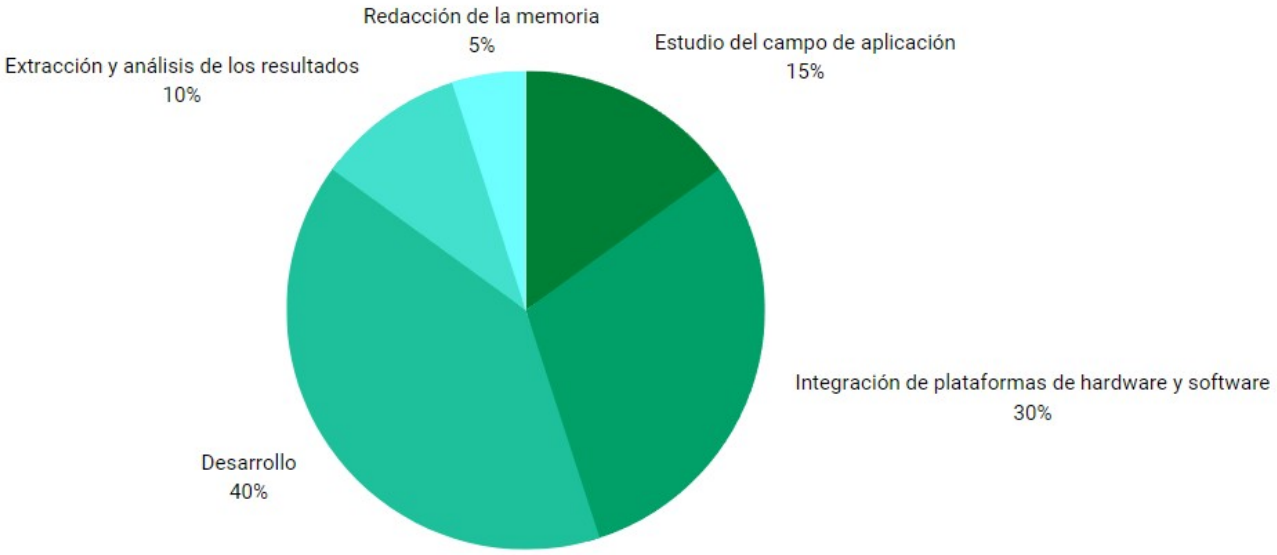

Figura 1: Planificación temporal

2. Estado del arte

2.1. Historia de la robótica

Es difícil situar históricamente el inicio de la robótica industrial pero podría fijarse alrededor siglo XVIII impulsado por la primera revolución industrial. La robótica ha sufrido una larga evolución a lo largo de los años que podemos dividir en cinco grandes generaciones:

- **Primera generación:** Aquí se encuentran los brazos mecánicos usados para la manipulación en entornos industriales. Estos requieren cierta supervisión y son usados en la automatización de procesos de fabricación.
- **Segunda generación:** En esta generación los robots tienen una inteligencia básica y la capacidad de aprender usando sensores para obtener información de su entorno.
- **Tercera generación:** La principal característica que define a los robots de esta generación es que son programables y autónomos incorporando sensores táctiles y visuales. Además se usan sistemas de control muy precisos que permiten al robot desplazarse sin una supervisión constante como pasaba en la primera generación.
- **Cuarta generación:** Estos robots empiezan a usar inteligencia artificial incorporando una gran cantidad de sensores permitiendo realizar tareas muy completas. Aquí se pueden situar robots humanoides…
- **Quinta generación:** Generación aún en desarrollo que espera que los robots sean capaces de imitar la conducta humana. En esta generación se habla de alcanzar la “singularidad” donde las máquinas posean una inteligencia igual o superior al del ser humano.

A pesar de la clasificación mostrada anteriormente, existen numerosas formas de clasificar a los robots ya sea por su nivel de inteligencia, lenguaje de programación...

2.2. Robots colaborativos

Para entender qué es un robot colaborativo será importante hablar de robots industriales y las desventajas de estos respecto a los colaborativos.

Un robot industrial puede ser definido como un dispositivo programable y autocontrolado que consta de unidades electrónicas, eléctricas o mecánicas para la realización de tareas complejas. Normalmente suelen estar aislados de los humanos y poseen su propio espacio de trabajo.

Los robots colaborativos o cobots en cambio están diseñados para trabajar junto a humanos compartiendo el mismo espacio de trabajo. Además, suelen ser mucho más ligeros en comparación a los industriales por lo que ofrecen una gran movilidad para su traslado. Todo esto hace que este tipo de robots puedan trabajar de manera segura y eficiente con humanos. [1]

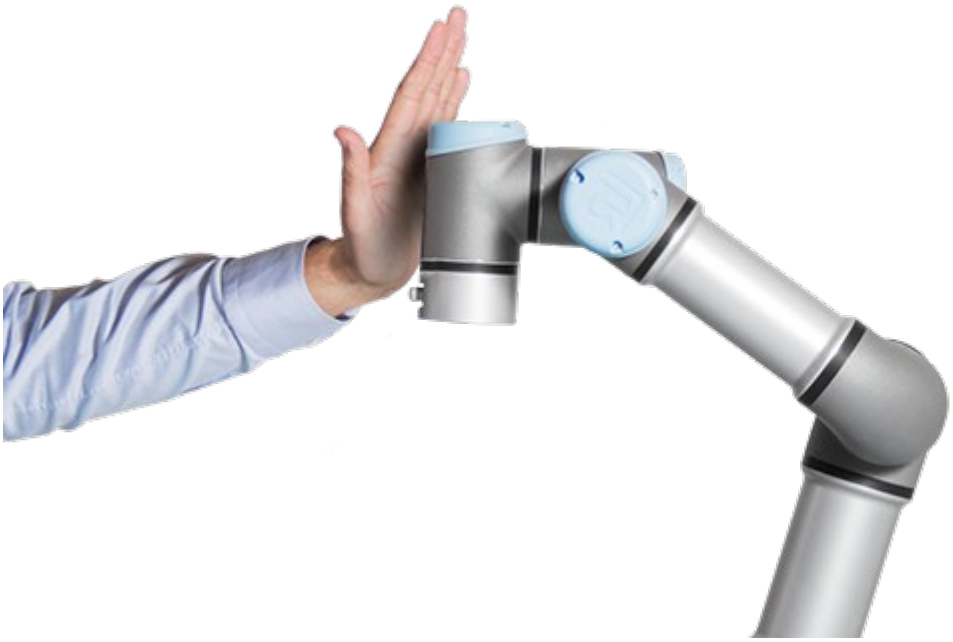

Figura 2: Robot colaborativo de Universal Robot

2.3. Visión por computador

La visión por computador trata de deducir de forma automática las estructuras y propiedades del mundo tridimensional a partir de una o varias imágenes bidimensionales. Las propiedades que se quieren deducir con la visión por computador podrían clasificarse en: [18]

- **Propiedades geométricas:** forma, tamaño, curvatura…
- **Propiedades de los materiales:** color, textura, composición…
- **Propiedades espaciales:** localización, orientación…
- **Propiedades cinemáticas:** velocidad, aceleración…

Este problema es difícil de resolver debido al gran volumen de información a procesar que depende de facturas que es imposible precisar.

2.3.1. Componentes básicos de un sistema de visión

Un sistema de visión se puede dividir en una parte de hardware que incluye el subsistema de adquisición, el computador con la tarjeta digitalizadora y el subsistema de visualización y otra parte de software en la que se incluye el software para controlar la adquisición y almacenamiento de la imagen, el software para manipular la imagen y el software para procesar e interpretar la imagen entre otras cosas.

Es importante destacar que en este tipo de sistemas es muy importante la iluminación y se suele decir que el éxito de un sistema de visión industrial depende más del diseño del sistema de iluminación que de un análisis sofisticado de la imagen. [18]

A continuación se muestran las distintas etapas para un proceso de visión 2D y 3D:

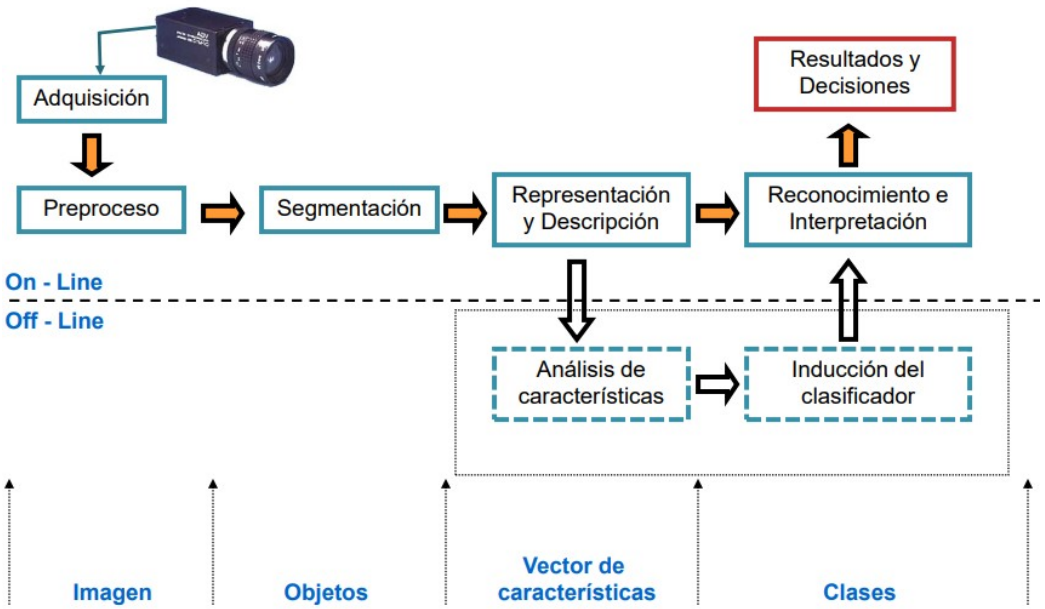

Figura 3: Etapas de un proceso de visión 2D [18]

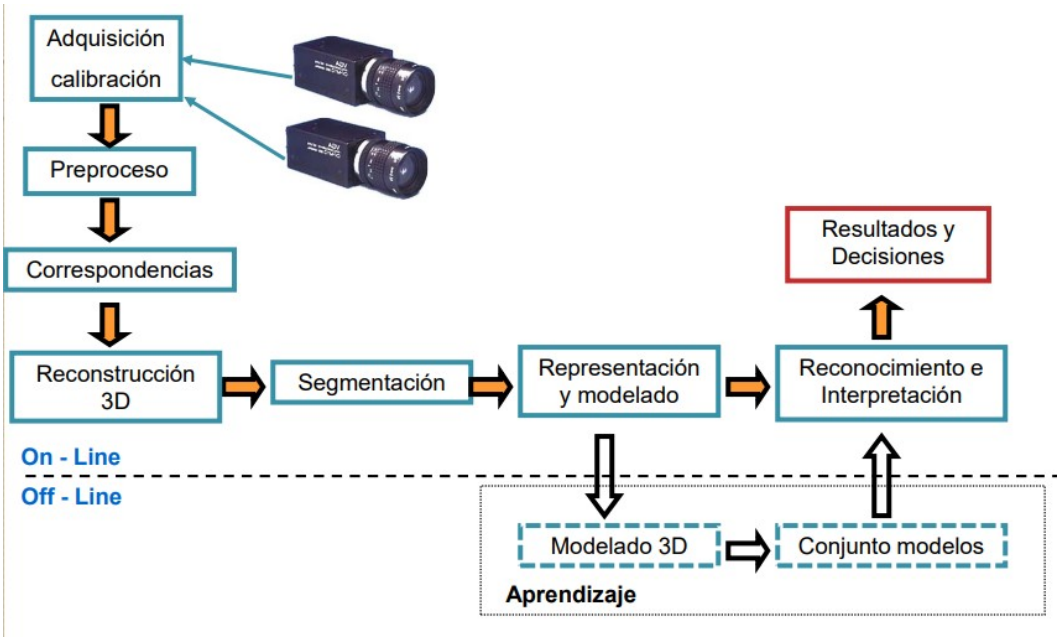

Figura 4: Etapas de un proceso de visión 3D [18]

3. Marco teórico

3.1. Cinemática

La cinemática es el estudio del movimiento sin considerar las fuerzas que lo producen y consiste en estudiar su movimiento con respecto a un sistema de diferencia. [2]

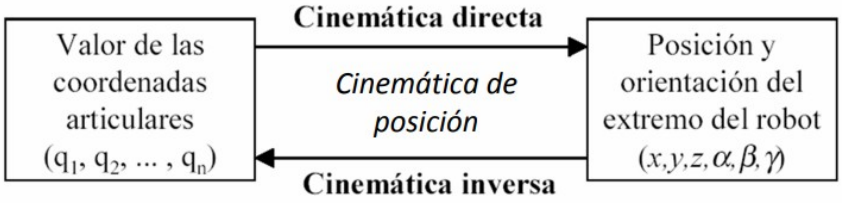

Figura 5: Relación entre las cinemáticas. [2]

3.1.1. Cinemática directa

El problema cinemático directo consiste en determinar la posición y orientación del extremo del robot con respecto a un sistema de coordenadas de referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot.

Para resolver el problema de la cinemática directa se pueden usar métodos geométricos, obteniendo la posición y orientación del extremo del robot utilizando relaciones geométricas, mediante matrices de transformación homogénea o bien usando el método de Denavit-Hartenberg (D-H).

La resolución mediante D-H es un método sistemático para situar los sistemas de coordenadas asociados a cada eslabón y obtener la cadena cinemática del robot. Este método es válido para robots del tipo cadena cinemática abierta.

$$A_i = R_{z,\theta i}\,Trans_{z,di}\,Trans_{x,ai}\,R_{x,\alpha i} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i \alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Siendo θ_i una rotación alredor del eje Z_{i-1}, d_i una traslación a lo largo del eje Z_{i-1}, a_i una traslación a lo largo del eje X_i y α_i una rotación alrededor del eje X_i.

3.1.2. Cinemática inversa

Este es el problema opuesto a la cinemática directa. Consiste en determinar la configuración que deben adoptar las articulaciones del robot para alcanzar una posición y orientación del extremo conocidas. Este problema es redundante puesto que en general se obtienen varias soluciones o dándose el caso de haber ninguna solución.

Los métodos genéricos suelen tener problemas de cálculo en tiempo real pero, por regla general se suelen desacoplar los tres primeros ejes del robot para ajustar la posición y los tres últimos para la orientación. [2]

3.1.3. Cinemática de velocidades

La cinemática de velocidades busca la relación que existe entre las velocidades de las articulaciones y las velocidades en la punta del robot. [2]

Las velocidades en la punta del robot se descomponen en velocidades lineales y angulares.

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \vec{w} \end{bmatrix} = J(q) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \cdot \\ \cdot \\ \dot{q}_n \end{bmatrix}$$

Partiendo de un modelo conocido de cinemática, se puede derivar para saber la relación entre las velocidades de las articulaciones y la punta del robot y viceversa.

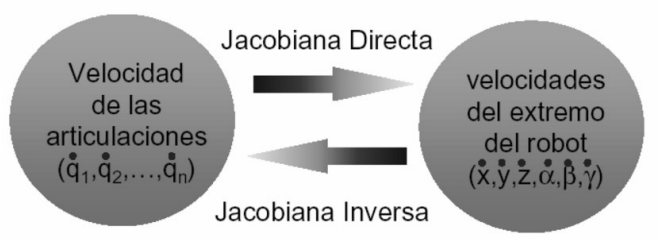

Figura 6: Jacobiana directa e inversa. [2]

En relación a la matriz jacobiana directa e inversa se pueden obtener las siguientes fórmulas:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = J \begin{bmatrix} \dot{q}_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \dot{q}_n \end{bmatrix} \qquad \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \cdot \\ \cdot \\ \cdot \\ \dot{q}_n \end{bmatrix} = J^{-1}(q) \begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix}$$

4. Hardware

4.1. Robot colaborativo UR3e

4.1.1. Introducción al UR3e

Para el desarrollo de este trabajo se han hecho uso de dos brazos robóticos UR3e pertenecientes a la empresa Universal Robots que, como característica principal respecto al UR3 es que posee un sensor de fuerza integrado del que se hará uso.

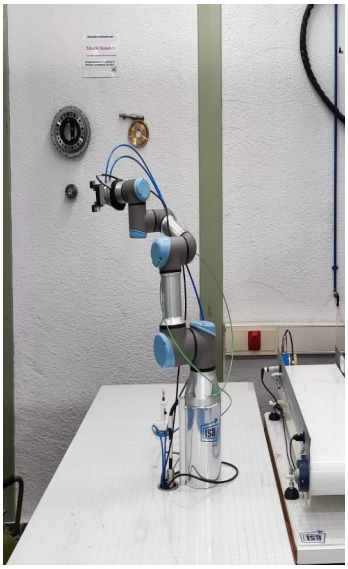

Figura 7: Robot colaborativo UR3e

El UR3e es un robot industrial colaborativo ultraligero y compacto ideal para la aplicación sobre mesas de trabajo. Este robot al contar con un tamaño muy reducido es ideal para implementarse directamente dentro de maquinaria o espacios muy reducidos de trabajo. [3]

A continuación veremos sus especificaciones más importantes:

Tabla 1: Especificaciones UR3e

Carga útil	3 kg
Alcance	500 mm
Grados de libertad	6 articulaciones giratorias
Programación	Pantalla táctil de 12” con interfaz gráfica de usuario Polyscope

Tabla 2: Movimiento UR3e

Repetibilidad según la norma ISO 9283	± 3 mm	
Movimiento de ejes:	**Rango de trabajo**	**Velocidad máxima**
Base	± 360º	± 180º/s
Hombro	± 360º	± 180º/s
Codo	± 360º	± 180º/s
Muñeca 1	± 360º	± 360º/s
Muñeca 2	± 360º	± 360º/s
Muñeca 3	Infinito	± 360º/s
Velocidad TCP estándar	1 m/s	

Además, el robot UR3e incluye una caja de control con clasificación IP44 del que hablaremos de sus especificaciones principales.

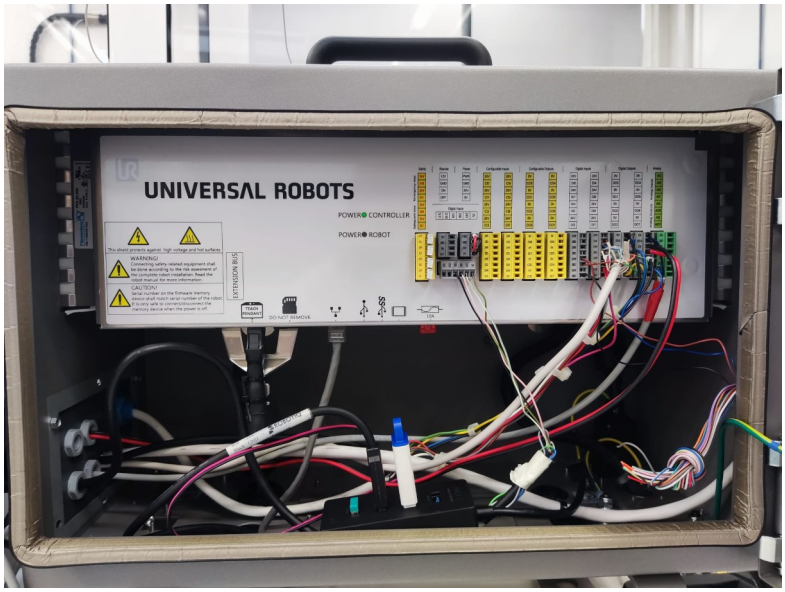

Figura 8: Caja de control UR3e

Tabla 3: Especificaciones principales caja de control UR3e

Puertos de E/S:	
Entradas digitales	16
Salidas digitales	16
Entradas analógicas	2
Salidas analógicas	2
Entradas digitales en cuadratura	4
Comunicación	Frecuencia de control 500 Hz Modbus TCP PROFINET Ethernet/IP USB 2.0, USB 3.0

4.1.2. Análisis cinemático del robot UR3e

Como bien se ha comentado en el marco teórico, será importante conocer la cinemática del robot con el que estamos trabajando, en este caso el robot UR3e.

Para ello, aplicaremos el método de Denavit-Hartenberg para hallar los parámetros de nuestro robot [4]. Se calcula la cinemática directa del robot por motivos académicos y para tener un mayor conocimiento del mismo, aunque para el desarrollo de pruebas experimentales no ha hecho falta utilizarla.

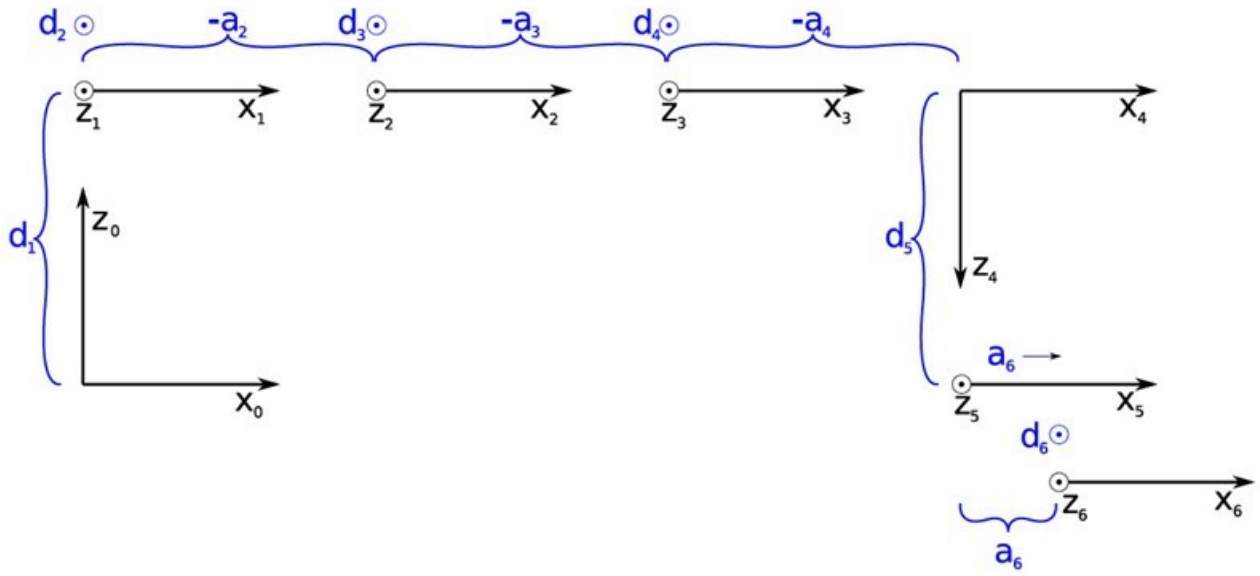

Figura 9: Ejemplo 1. Diagrama Denavit-Hartenberg UR3e. [4]

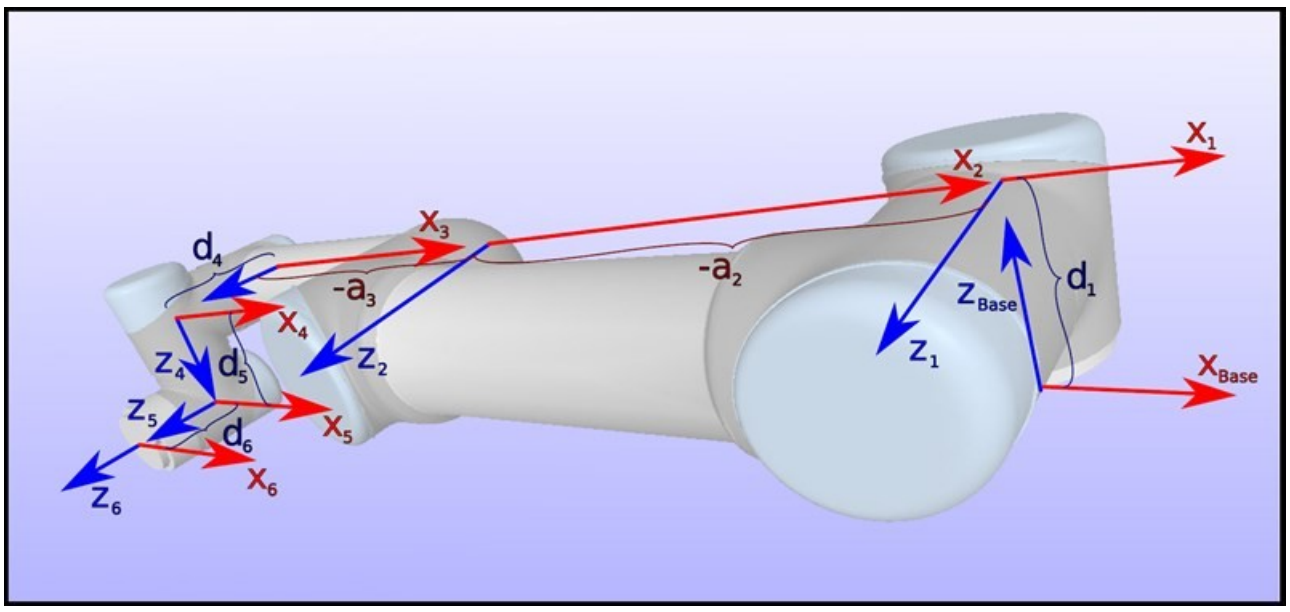

Figura 10: Ejemplo 2. Diagrama Denabit-Hartenberg UR3e. [4]

En la siguiente tabla se muestran los parámetros de Denavit-Hartenberg para el robot UR3e.

Tabla 4: Parámetros D-H para la cinemática del robot UR3e [4]

Cinemática	theta [rad]	a [m]	d [m]	alpha [rad]
Articulación 1	0	0	0.15185	π/2
Articulación 2	0	-0.24355	0	0
Articulación 3	0	-0.2132	0	0
Articulación 4	0	0	0.13105	π/2
Articulación 5	0	0	0.08535	-π/2
Articulación 6	0	0	0.0921	0

4.2. Robotiq Wrist Camera

Para la realización del trabajo se hizo uso de una cámara de visión de la empresa Robotiq para el UR3e, la “Wrist Camera”.

Figura 11: Robotiq Wrist Camera [5]

Esta cámara utiliza un sensor con tecnología CMOS para la captación de imágenes teniendo una resolución de entre 0.3 y 5 Mpx. Además, los fotogramas de la cámara varían entre 2 a 30 FPS.

El campo de visión que posee es otra característica interesante de la cámara siendo de 36x27 centímetros su campo de visión máxima y 10x7.5 centímetros el mínimo.

Para el uso de esta cámara se necesita instalar un software especial en Polyscope, lo que se denomina un “URCap”.

Una vez instalado todo el software será el momento de definir la “snapshot position” que será la posición que el robot tomará de referencia para determinar el campo de visión de la cámara y, por lo tanto, el espacio de trabajo utilizado posteriormente.

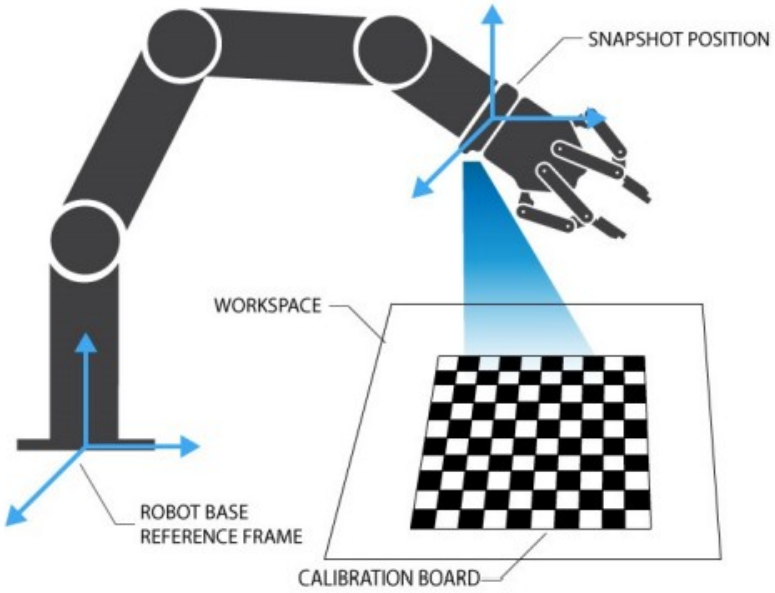

Figura 12: Representación esquemática del sistema de visión [5]

Al definir la posición de referencia habrá que calibrar la cámara usando un patrón de calibración que nos proporciona la propia empresa.

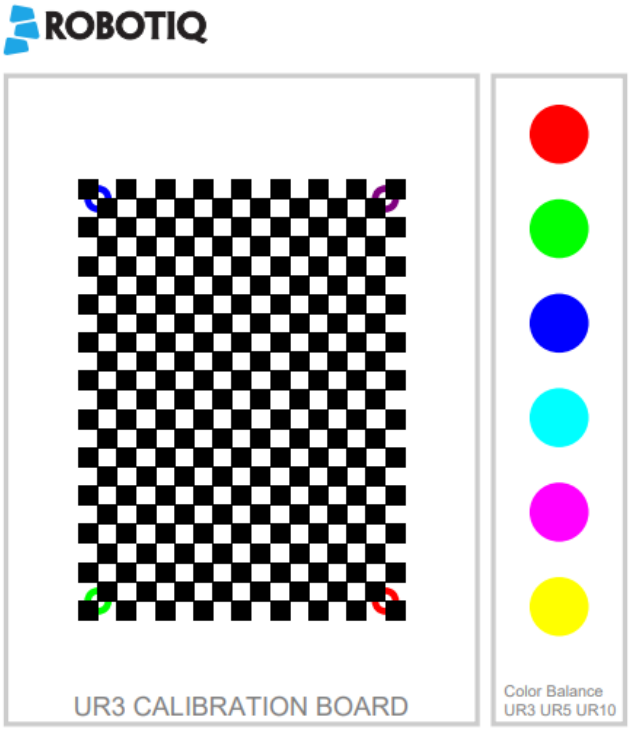

Figura 13: Calibración Wrist Camera UR3e [5]

El robot tomará 27 fotos con poses diferentes y 9 más de validación. Una vez terminado nos las mostrará para comprobar si se ha detectado bien el patrón o es necesario volver a calibrar la cámara.

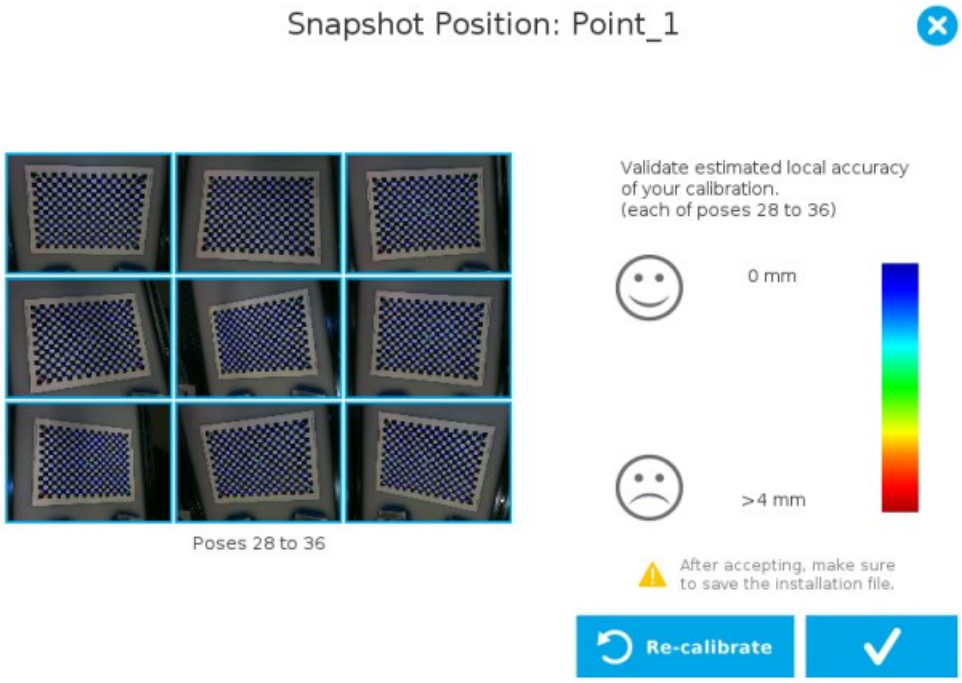

Figura 14: Validación de la calibración de Wrist Camera [5]

Para el aprendizaje de objetos existen dos métodos principales, el automático basado en fotos y un escaneo del objeto y el paramétrico que construye un modelo basado en parámetros de una forma 2D básica (círculo, anillo, cuadrado o rectángulo).

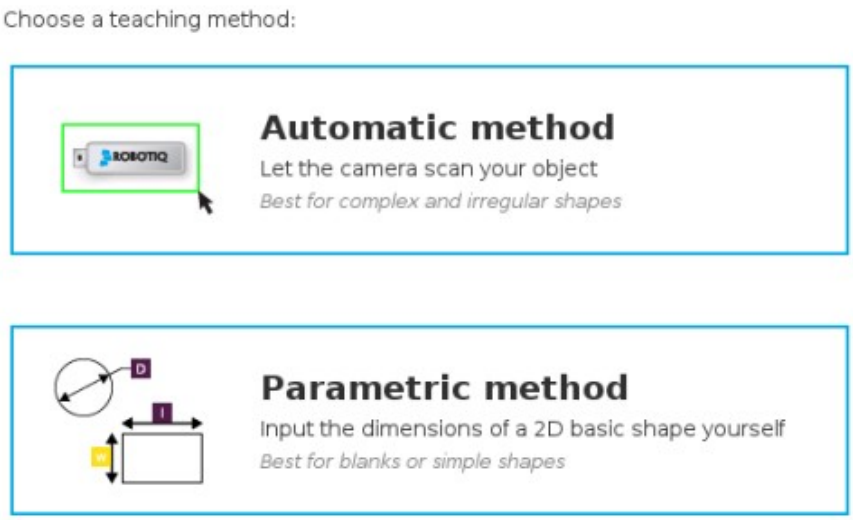

Figura 15: Selección del método de aprendizaje de objeto Wrist Camera [5]

El método automático es ideal para formas complejas e irregular mientras que el paramétrico es un método más rápido y permite que el sistema de visión reconozca y ubique con gran robustez objetos que tienen pocas características distintivas. Este último método suele dar mejores resultados para geometría simple y objetivos altamente reflectantes. [5]

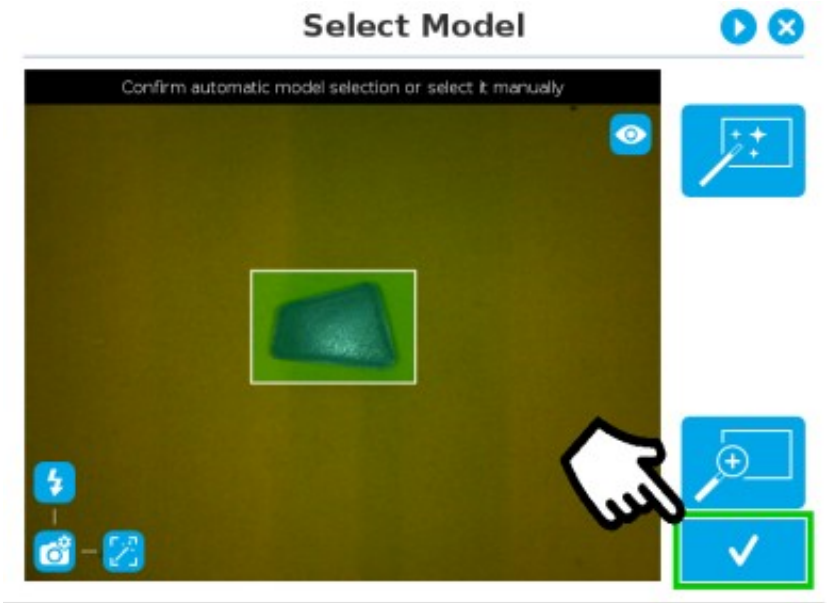

Figura 16: Aprendizaje de objetos Wrist Camera [5]

4.3. Robot industrial ABB IRB 140

El robot ABB ha sido necesario para el uso de la cinta transportadora para una tarea colaborativa de pick & place. Para el uso de la cinta, los robots UR3e no pueden acceder a sus actuadores ni a la información de los sensores ya que el conexionado de la misma está realizado en serie a través del ABB por lo que ha sido necesario cargar un programa cíclico que envíe esta información a los dos robots colaborativos.

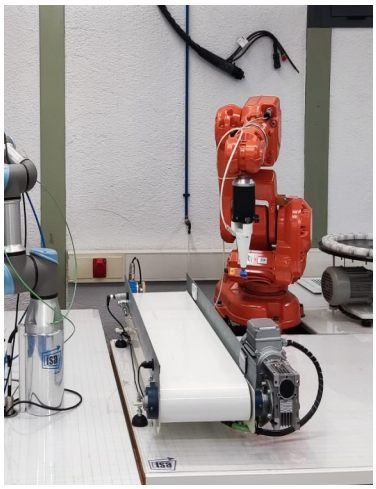

Figura 17: ABB IRB 140 y cinta transportadora

El código de RAPID diseñado se puede ver en el "*ANEXO A1*". Este código tendrá dos funciones principales:

- Si el ABB detecta que el UR3e ha activado la entrada digital correspondiente, pone en marcha la cinta en un sentido u otro (dependiendo de la entrada activada).
- Mandar la información de los sensores (detección de objetos) al robot UR3e para poder hacer la programación en consecuencia.

4.4. Gafas de realidad virtual Oculus Quest 2

Para el desarrollo del gemelo digital del UR3e fue necesario el uso de unas gafas de realidad virtual. Se han elegido las Oculus Quest 2 por ser una de las opciones más baratas actuales del mercado y su portabilidad, siendo capaces del seguimiento del movimiento sin necesidad de tener instaladas estaciones base como el resto de competidores.

Además, cabe destacar que para conectarlas al ordenador ha sido necesario un cable USB-C de alta velocidad. Una vez terminada la aplicación sería posible instalarlas en las mismas gafas sin necesidad de un ordenador ya que cuenta con un procesador suficientemente potente en su interior.

Figura 18: Gafas de Realidad Virtual Oculus Quest 2

A continuación veremos algunas de sus características principales:

Tabla 5: Principales características de Oculus Quest 2 [6]

Seguimiento	**Seis grados de libertad** Gracias a la tecnología de seis grados de libertad, estas gafas realizan un seguimiento de los movimientos tanto de cabeza como del cuerpo. No se necesitan sensores externos.
Óptica	**Especificaciones** Pantalla LCD de cambio rápido 1832 x 1920 de resolución en cada ojo Frecuencia de actualización admitida: 60, 72, 90 Hz Compatible con el uso de gafas
Sonido	**Audio posicional** El audio posicional 3D está integrado directamente en las gafas y te permite escuchar todo lo que te rodea. Además, cuenta con un puerto de audio de 3,5 mm para poder conectar unos auriculares
Almacenamiento	128 GB

5. Software

5.1. Polyscope

La programación de los robots UR3e se realizarán a través del software “Polyscope” al que accederemos a través de la consola de programación.

Figura 19: Consola de programación UR3e [7]

Dentro de la consola de programación encontraremos diferentes funciones a las que podremos acceder como los comandos de movimiento, ejecución de scripts, subprogramas… A continuación veremos algunos de estos.

5.1.1. Comandos de movimiento

Gracias a este tipo de comandos se logra controlar la trayectoria que genera el robot a través de puntos de paso especificando parámetros como la velocidad y la aceleración del movimiento deseado. A continuación se muestran los movimientos más habituales. [7]

MoveJ: Este tipo de movimiento se usa cuando no importa la trayectoria de la herramienta y se encuentra en un espacio de trabajo libre sin riesgo a colisión. Es un movimiento no lineal y es mucho más rápido que un “MoveL”.

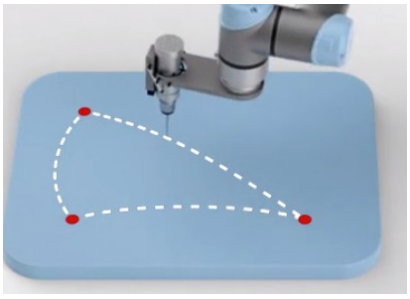

Figura 20: MoveJ de un UR3e

MoveL: Movimiento lineal y preciso en un espacio cerrado. Además, la trayectoria es calculada en el espacio cartesiano por lo que es un movimiento más lento que el "MoveJ".

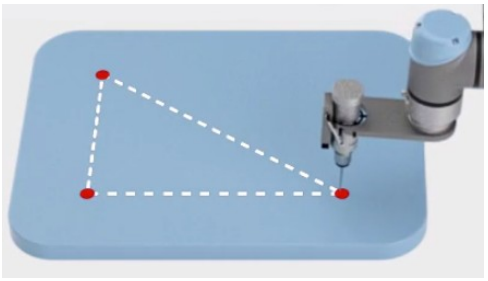

Figura 21: MoveL de un UR3e

MoveP: Movimiento lineal que mantiene una velocidad constante dando lugar a transiciones circulares entre los puntos de paso. Este tipo de movimientos se suelen usar en soldaduras por su distribución regular.

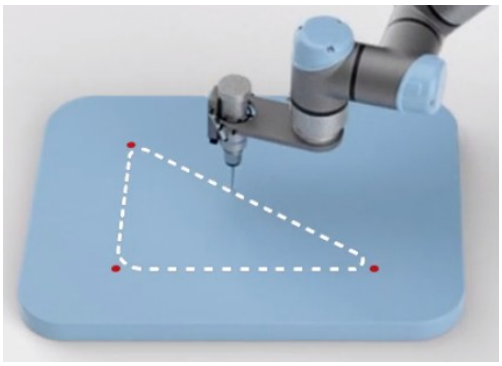

Figura 22: MoveP de un UR3e

5.2. Unity

Figura 23: Logo Unity

Para el desarrollo del gemelo digital se hizo uso de la herramienta "Unity" que es un motor usado en videojuegos ampliamente utilizado en la industria del desarrollo de videojuegos y aplicaciones interactivas.

Unity ofrece un gran conjunto de herramientas y recursos para implementar nuestros entornos junto con su correspondiente programación en C#.

Una de las grandes ventajas de Unity es su desarrollo multiplataforma permitiendo desarrollar una aplicación y ejecutarla en diferentes plataformas. En nuestro caso es útil para realizar la aplicación del gemelo digital en Android que es la plataforma soportada por las gafas de realidad virtual Oculus Quest 2.

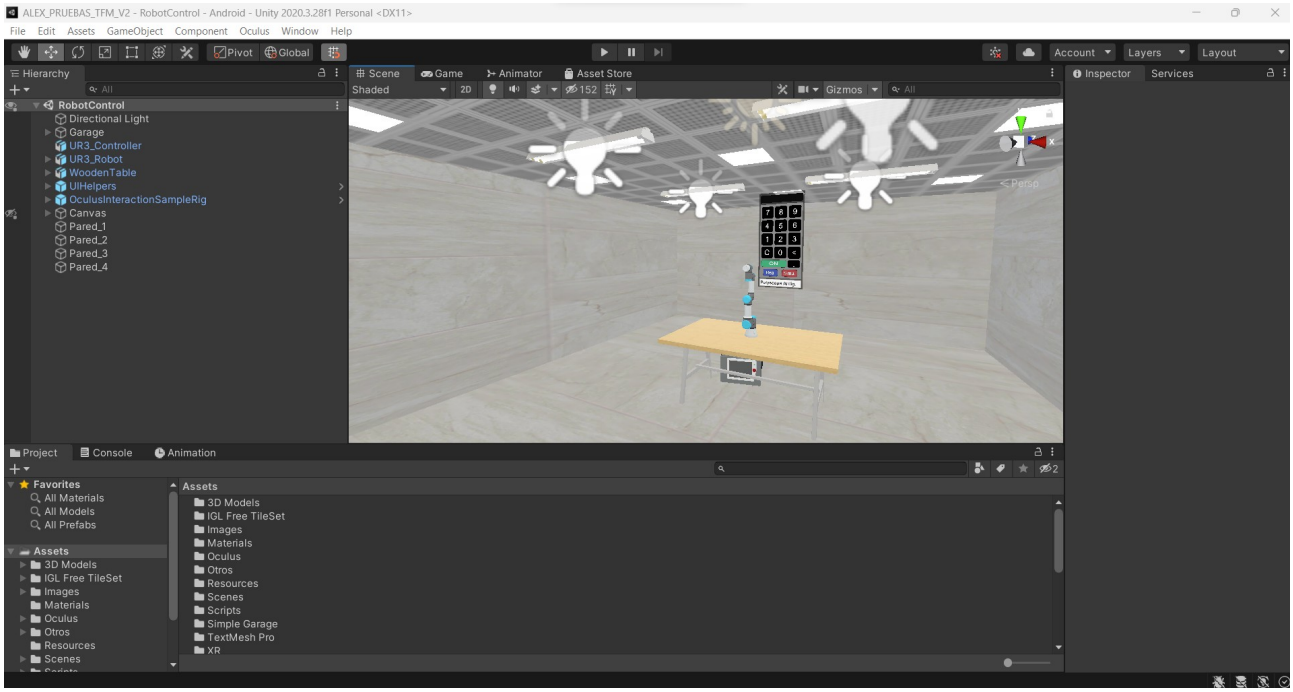

Figura 24: Interfaz de Unity

6. Diseño de pruebas experimentales

En este capítulo se tratarán varias temáticas.

- Pick & Place colaborativo con visión.
- Prueba modo fuerza UR3e.
- Gemelo digital del robot UR3e.
- Gemelo digital del robot UR3e con gafas de realidad virtual.

6.1. Pick & Place colaborativo con visión

El objetivo de esta parte del trabajo será realizar una tarea colaborativa de pick & place en la que el primer robot localizará un objeto enseñado a través de una cámara de visión

6.1.1. Configuración Wrist Camera

Como bien se comentó en el apartado **4.2**, para el uso de la cámara de visión artificial, será necesario configurarla. Lo primero que se hizo fue definir el “snapshot_position” que será la posición que nuestro robot tome como referencia.

Figura 25: Definición de la Snapshot Position

Una vez definida esta referencia, el robot se moverá automáticamente para hacer la calibración con el patrón y tomará las fotografías correspondientes en distintas posiciones. Una vez termine de tomar las fotografías, se nos mostrarán para validar que se ha conseguido captar el patrón de calibración en todas ellas.

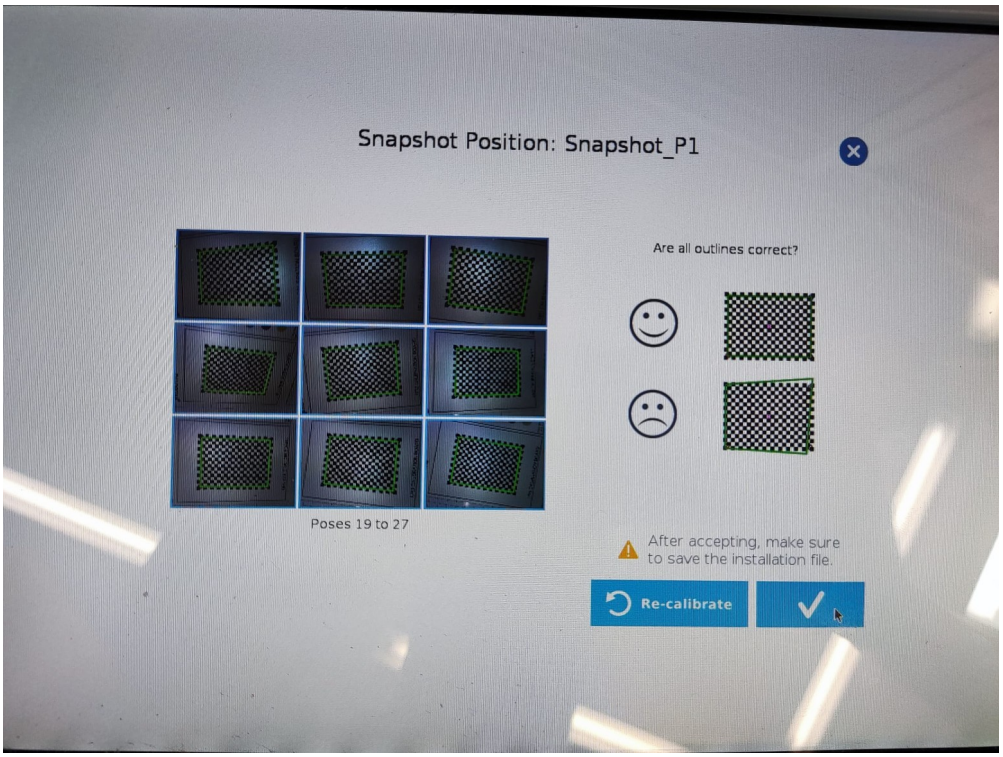

Figura 26: Validación calibración UR3e

Una vez hecho esto pasaremos al aprendizaje de objetos de la cámara. En nuestro caso se ha realizado a través del método automático explicado con anterioridad de un cubo verde.

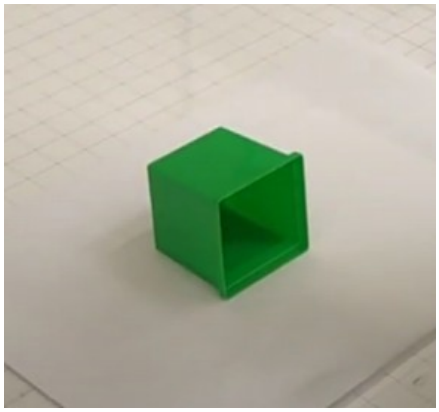

Figura 27: Objeto enseñado Wrist Camera

A continuación se enseña el proceso de aprendizaje de la pieza seleccionada así como el ajuste de la detección de los lados y del umbral de detección del color.

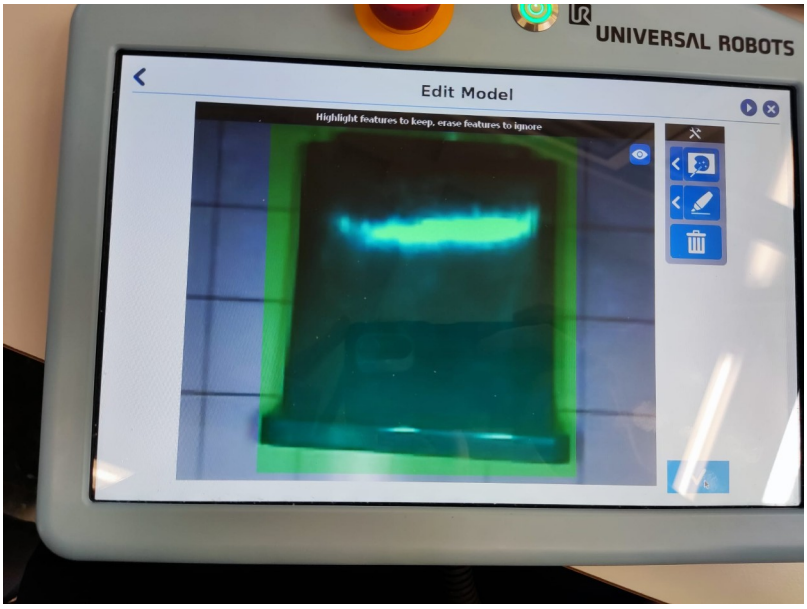

Figura 28: Aprendizaje automático pieza

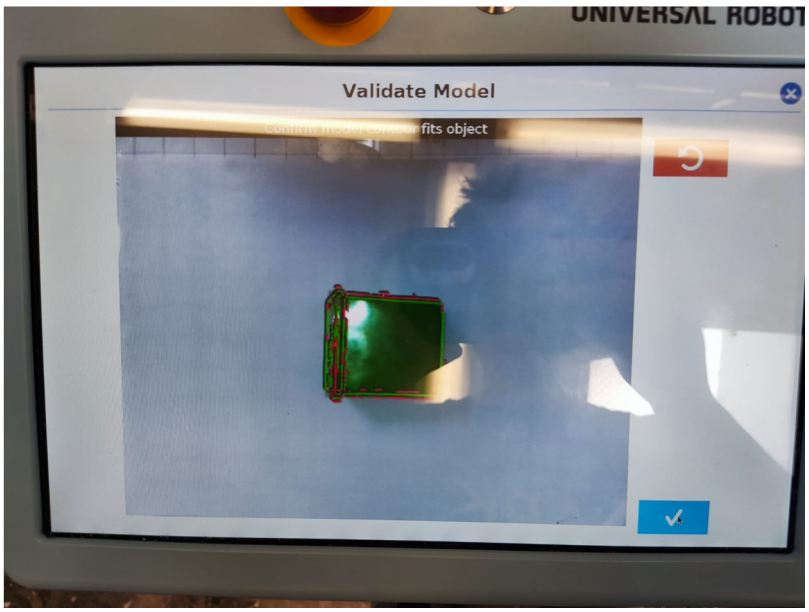

Figura 29: Validación modelo

También se debe ajustar el umbral de la detección de los lados del objeto así como de la detección del color que en la figura sería el verde. Los umbrales escogidos fueron los siguientes:

- **Detección de lados:** 75%
- **Detección de color**: 60%

El umbral de detección de color fue escogido así debido a los cambios de iluminación en nuestro entorno que podía afectar a la correcta localización de la pieza.

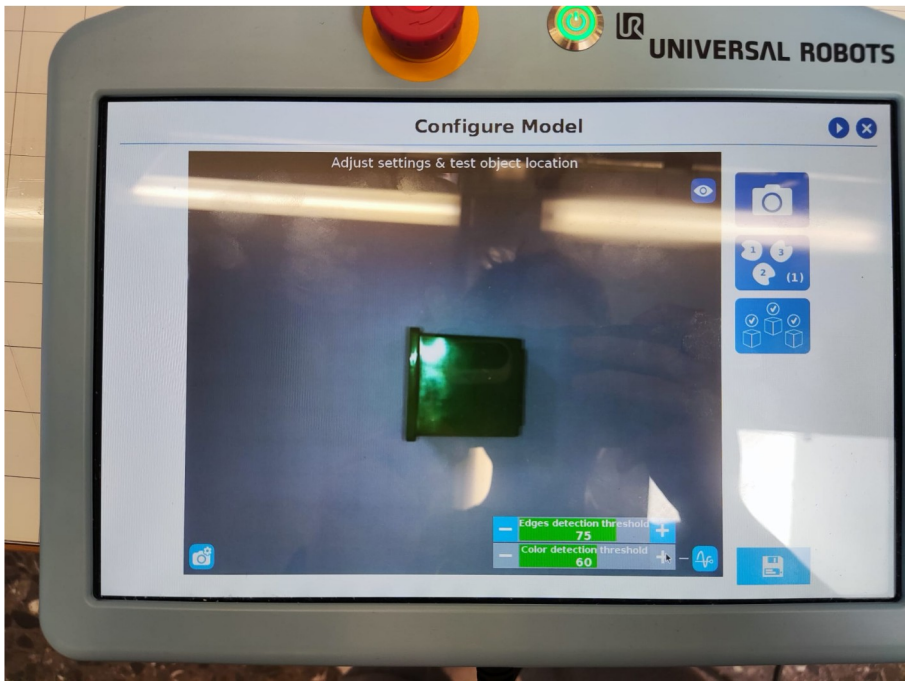

Figura 30: Configuración del modelo

Una vez calibrada toda la cámara y enseñado el objeto que queremos localizar, podremos pasar a añadirlo a nuestra programa del robot a través del módulo “Camera Locate”.

6.1.2. Configuración cinta transportadora y sensores

En este pick & place se hará uso de una cinta transportadora y de los sensores correspondientes para la detección del objeto. Las conexiones de la cinta y sensores del laboratorio estaban conectadas a través del robot ABB en serie y no en paralelo por lo que, para acceder a ellas fue necesario cargar un programa cíclico que enviase la información a los otros dos robots UR3e. El código se puede consultar en el “*ANEXO A1*”.

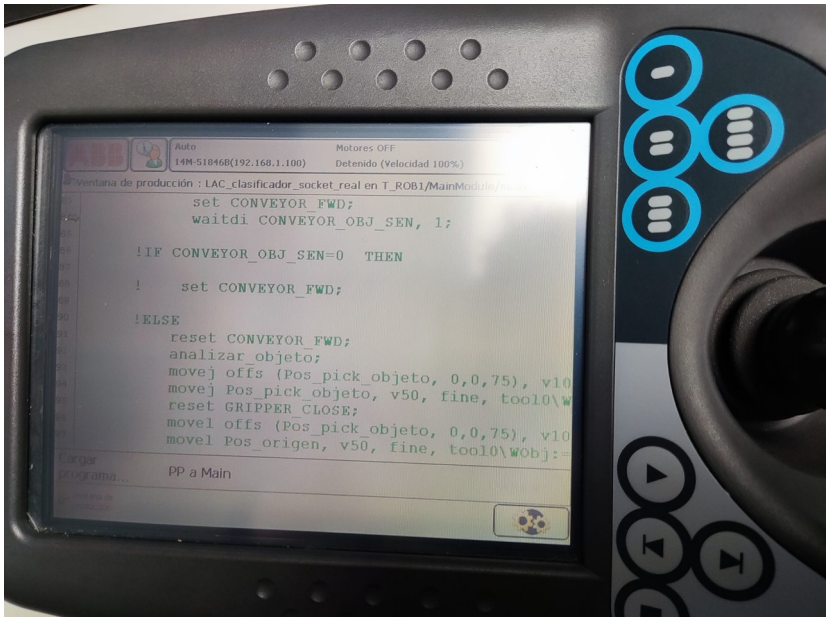

Figura 31: Configuración ABB para comunicación con UR3e

Si el conexionado de la cinta hubiese estado hecho en paralelo no hubiese sido necesario cargar este programa para poder usarla con los otros dos robots UR3e.

6.1.3. Programación y lógica de funcionamiento

Una vez hecha la configuración de la cámara de visión y del programa del robot ABB ya podremos pasar a configurar los dos robots UR3e para la tarea de pick & place. A continuación se muestra una imagen del montaje final junto con la explicación de funcionamiento.

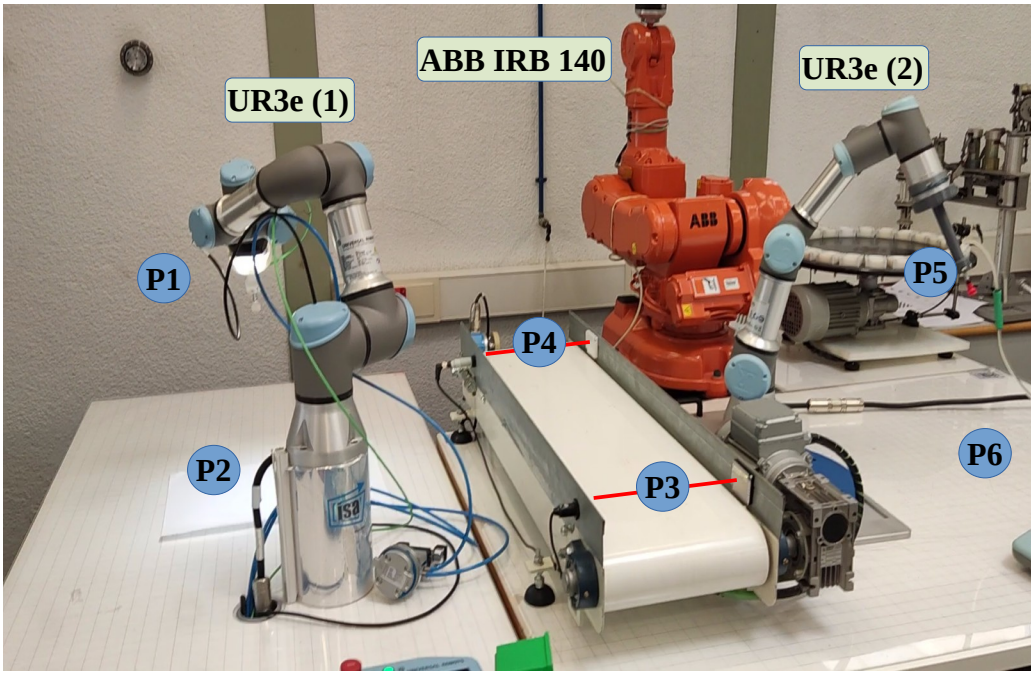

Figura 32: Esquema explicación pick & place

El primer robot UR3e es el que posee la cámara de visión artificial por lo que quedará esperando en **P1** hasta que logre detectar el objeto enseñado. Una vez se detecte el objeto en el espacio de **P2**, se dirigirá hacia esta posición relativa para agarrar el objeto a través de una ventosa que usa como herramienta. Al agarrar el objeto volverá a **P1** que es una posición segura libre de colisiones para dirigirse a **P3** donde se encuentra la cinta. Al dejar el objeto en la posición indicada, el robot volverá a su posición de reposo a la espera de encontrar otro objeto.

Al depositar el objeto en **P3**, se activará el primer sensor que hará que se active la cinta hasta la posición de **P4**. Una vez llegue a **P4** se parará la cinta a esperas de que sea recogida la pieza.

El segundo robot UR3e se activará al llegar la pieza a **P4** ya que se activará el sensor correspondiente e irá hasta esa posición para agarrar el objeto con otra ventosa. Una vez agarrado lo dejará en **P6.**

La programación de los dos robots UR3e y del robot ABB IRB 140 se puede encontrar en el "*ANEXO A*". Para una mejor comprensión de esta parte se recomienda visualizar el vídeo grabado a través de este enlace [8]. A continuación se muestran algunos fotogramas del vídeo:

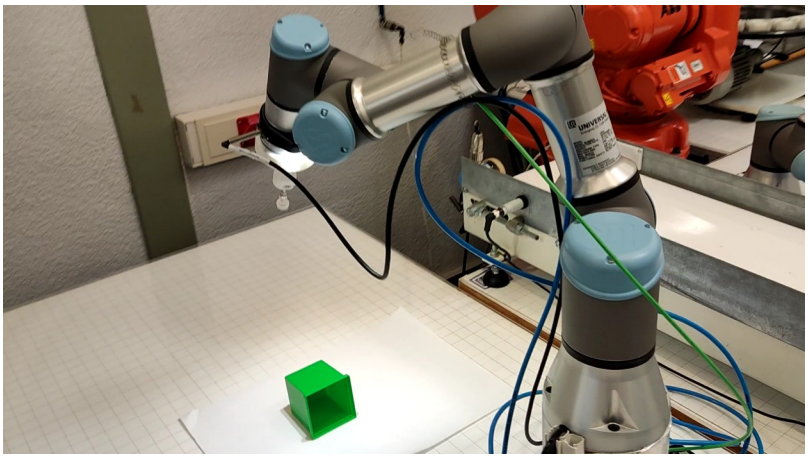

Figura 33: Prueba experimental Pick & Place (I)

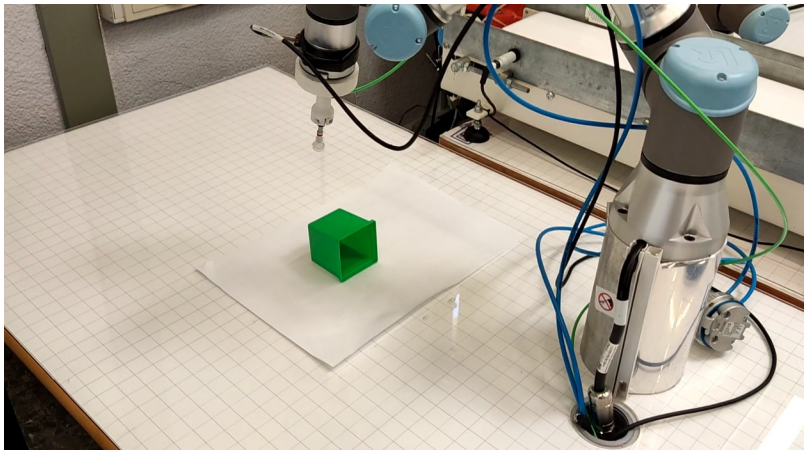

Figura 34: Prueba experimental Pick & Place (II)

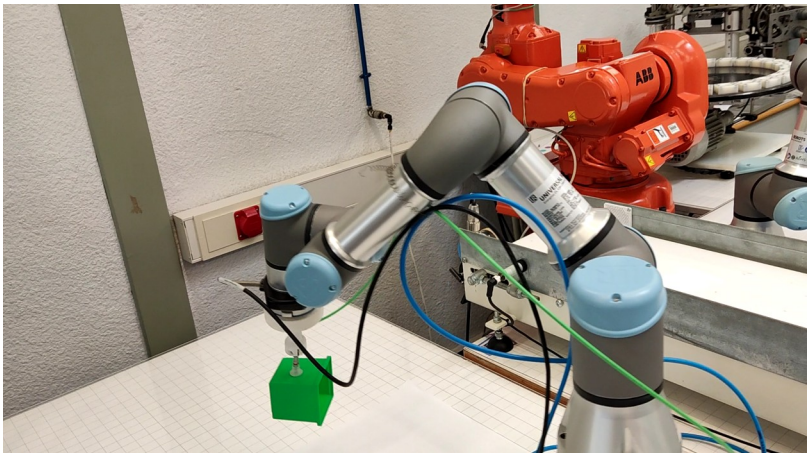

Figura 35: Prueba experimental Pick & Place (III)

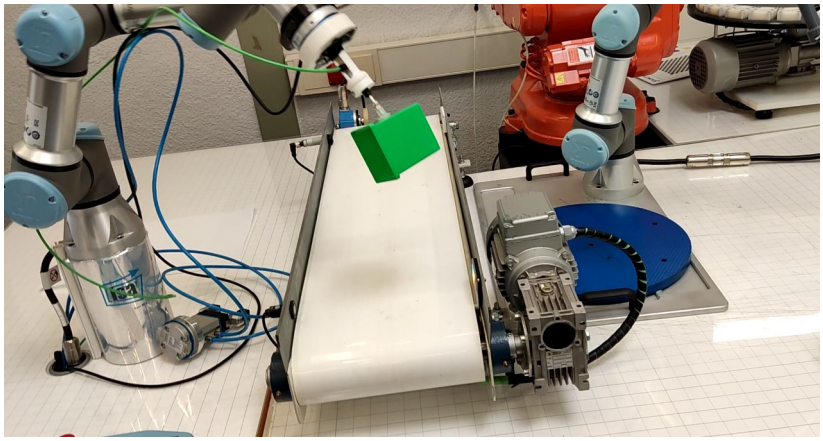

Figura 36: Prueba experimental Pick & Place (IV)

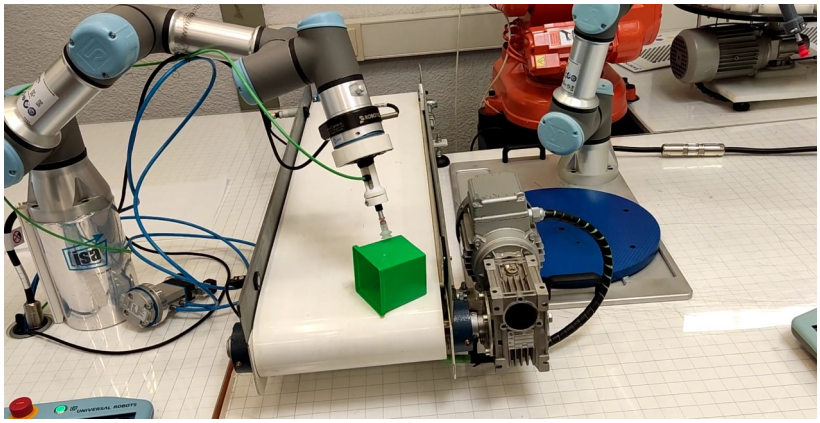

Figura 37: Prueba experimental Pick & Place (V)

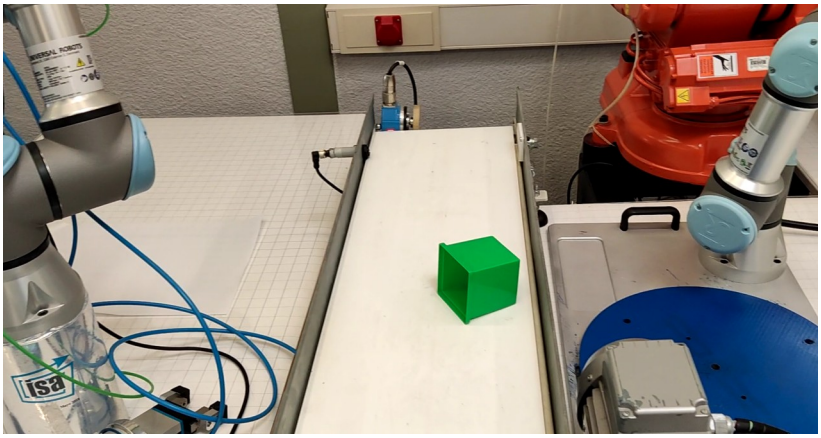

Figura 38: Prueba experimental Pick & Place (VI)

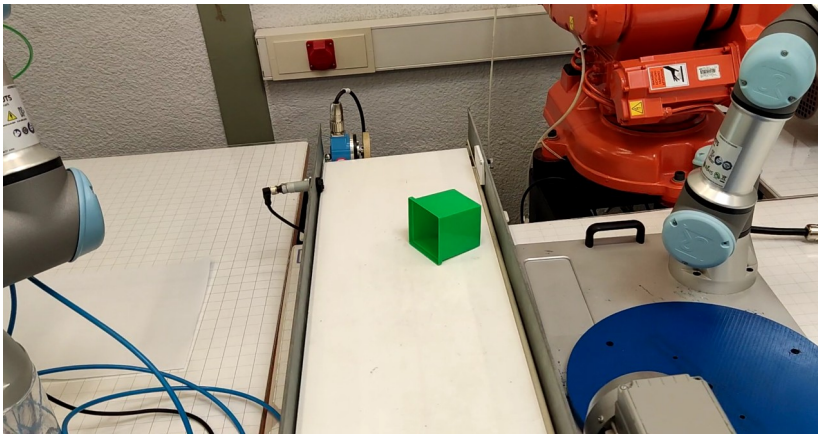

Figura 39: Prueba experimental Pick & Place (VII)

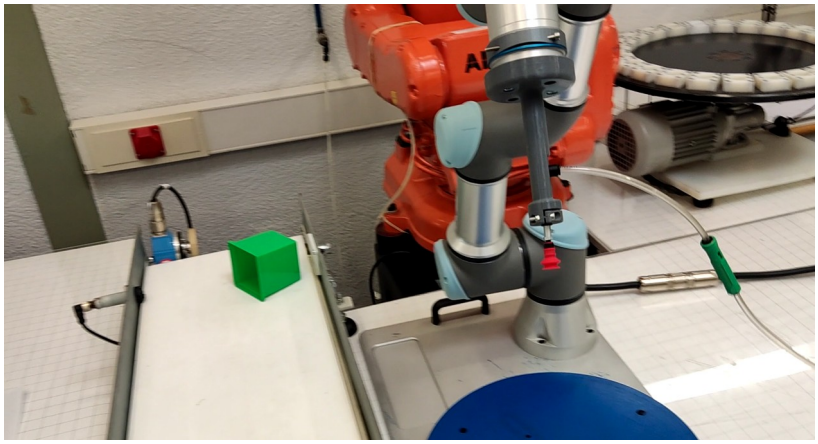

Figura 40: Prueba experimental Pick & Place (VIII)

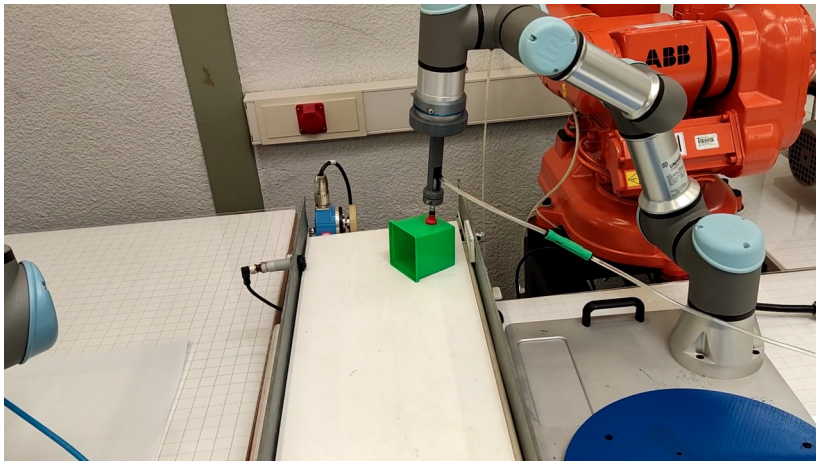

Figura 41: Prueba experimental Pick & Place (IX)

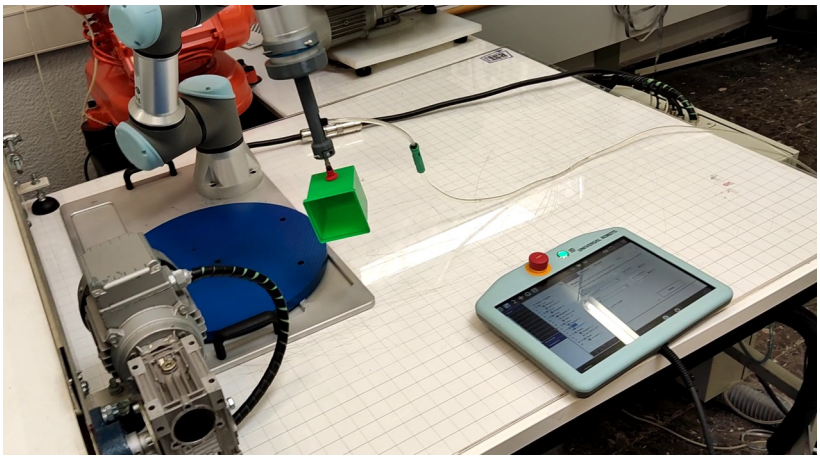

Figura 42: Prueba experimental Pick & Place (X)

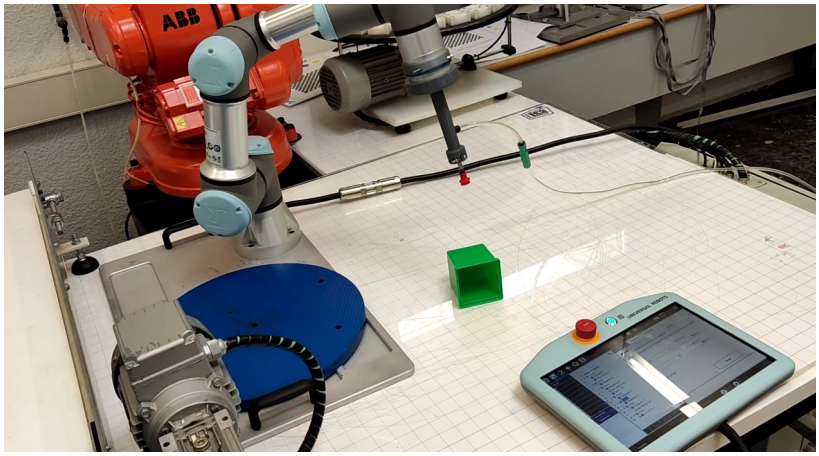

Figura 43: Prueba experimental Pick & Place (XI)

6.2. Modo fuerza UR3e

6.2.1. Introducción

El modo de fuerza que posee el robot UR3e es ideal para aplicaciones en las que la posición real del PCH a lo largo de un eje predefinido no sea importante pero que, en cambio, sea necesaria aplicar una fuerza deseada a lo largo de dicho eje (robot moviéndose por superficie curva, empujando o tirando de una pieza…)

Este modo además permite aplicar determinados pares de torsión alrededor de ejes predefinidos en los que intentará acelerar en la dirección de dicho ese si no se encuentra con obstáculos en un eje donde se tenga definida una fuerza distinta de cero. [12]

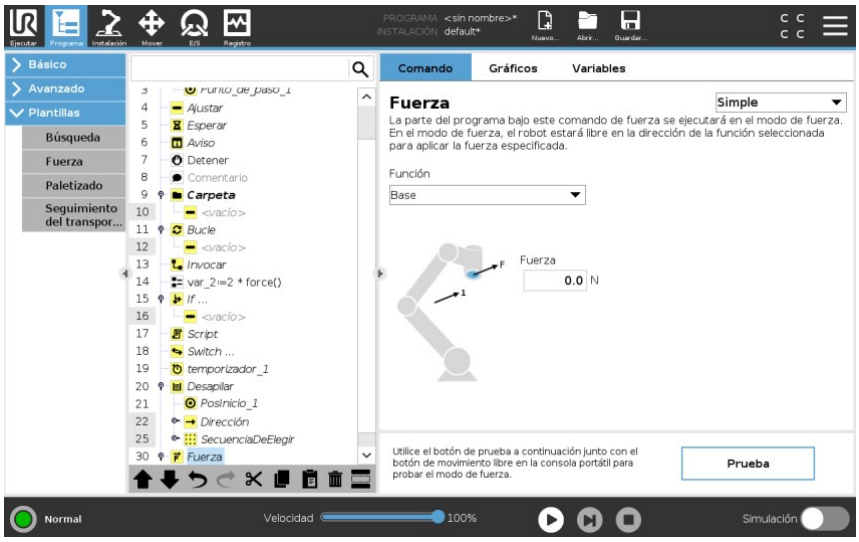

Figura 44: Interfaz programación UR3e plantilla fuerza [12]

Existen distintos tipos del modo de fuerza según la aplicación que se quiera realizar. A continuación veremos los principales: [12]

- **Simple:** Solo se aplica en un eje pudiéndose adaptar la fuerza a lo largo del mismo. La fuerza deseada siempre se aplica a lo largo del eje z de la función seleccionada. En el caso de coordenadas de líneas se aplicará a lo largo del eje y.
- **Marco:** Este modo permite seleccionar de forma independiente la adaptabilidad y las fuerzas en los seis grados de libertad.
- **Punto:** Cuando se selecciona Punto, el marco de tarea tiene el eje Y apuntando desde el PCH del robot hacia el punto de partida de la función seleccionada. La distancia entre el PCH del robot y el punto de partida de la función seleccionada ha de ser al menos de 10 mm. El marco de tarea cambia durante el tiempo de ejecución a medida que lo hace la posición del PCH del robot. El eje x y el eje z del marco de tarea dependen de la orientación original de la función seleccionada.

- **Movimiento:** Movimiento significa que el marco de tarea cambiará la dirección del movimiento del PCH. El eje x del marco de tarea será la proyección de la dirección del movimiento del PCH sobre el plano formado por los ejes x e y de la función seleccionada. El eje Y será perpendicular al movimiento del brazo robótico, y estará en el plano X-Y de la función seleccionada. Esto puede resultar útil al desbarbar a lo largo de una trayectoria completa donde haga falta una fuerza perpendicular al movimiento del PCH.

6.2.2. Plataforma experimental para pruebas del modo fuerza

Una vez sabemos qué es el modo fuerza, pasaremos a montar una plataforma experimental muy simple para comprobar el funcionamiento del mismo.

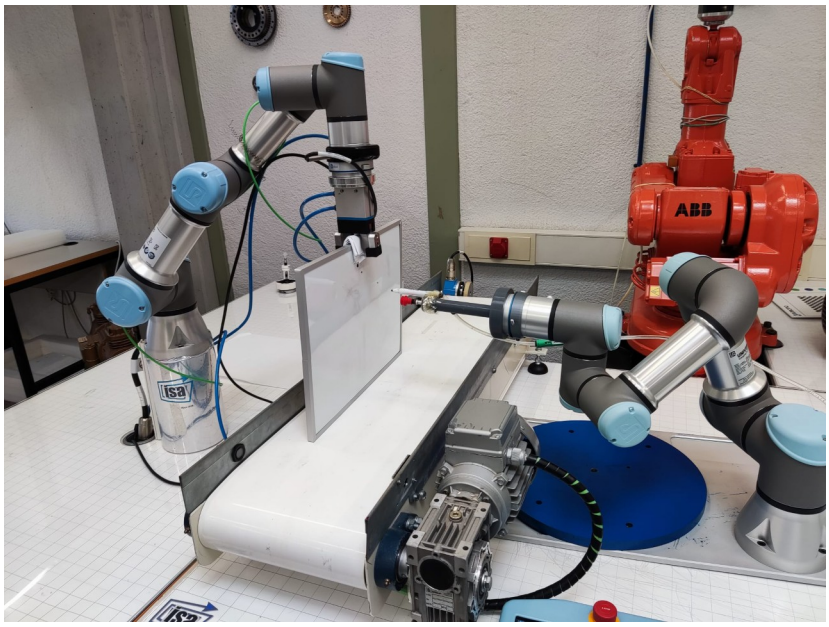

Figura 45: Plataforma experimental. Modo fuerza (I)

El primer robot usa como herramienta una pinza para agarrar una pizarra mientras que al segundo robot se le añadió un rotulador con cinta para aplicar una fuerza constante a lo largo del eje z en la pizarra.

Este planteamiento tiene doble utilidad:

- **Seguridad:** Evita romper la herramienta si el control de fuerza no se programa bien ya que simplemente se caería/desprendería el rotulador al verse vencido el “atado” por una fuerza exagerada.
- **Feedback visual:** Sirve de indicador visual de que la tarea de está haciendo bien, se consigue mantener una presión adecuada. La presión es mayor que un valor mínimo necesario porque vemos que el rotulador pinta pero, a su vez, no es excesiva ya que el rotulador no se cae.

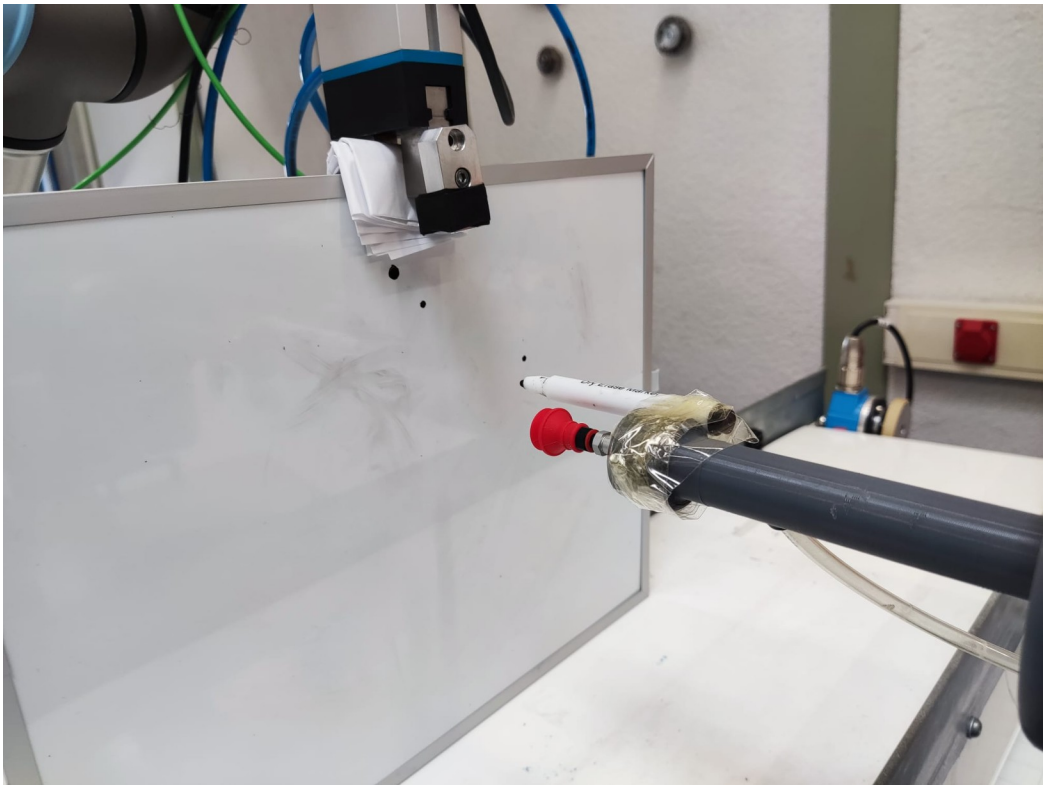

Figura 46: Plataforma experimental. Modo fuerza (II)

En este ejemplo se han realizado distintas pruebas en las que se ha logrado ver que una fuerza de 1 Newton es más que suficiente para aplicar una buena presión a la pizarra. Además, una vez se llega a la fuerza indicada se mueve la pizarra para que se vuelva adaptar el robot del rotulador y vuelva a aplicar una fuerza constante.

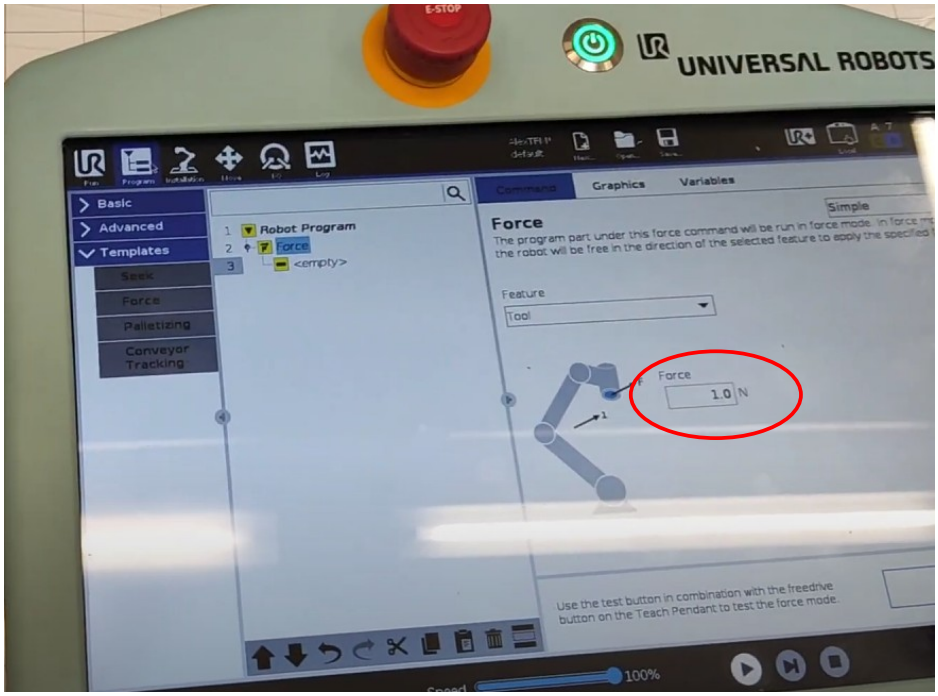

Figura 47: Configuración modo fuerza 1N

Para una mejor comprensión de esta parte se recomienda visualizar el vídeo grabado a través de este enlace [9]. A continuación se muestran algunos fotogramas del vídeo:

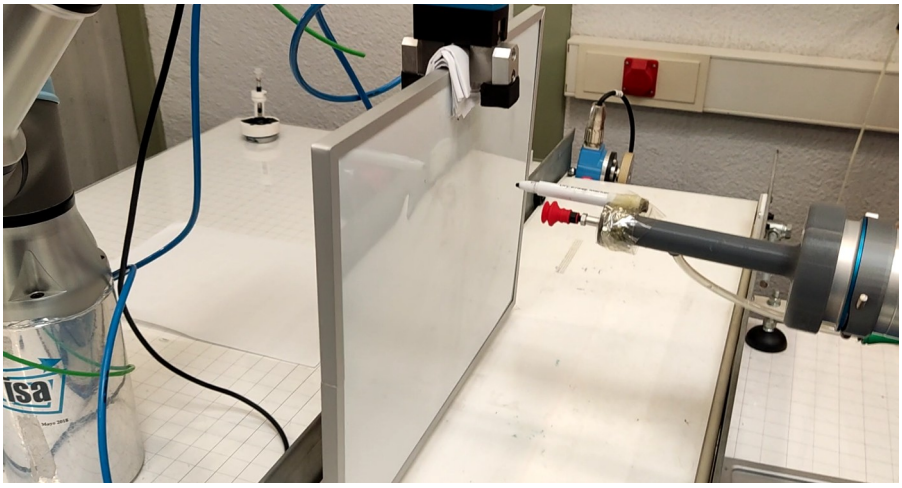

Figura 48: Prueba experimental Modo Fuerza (I)

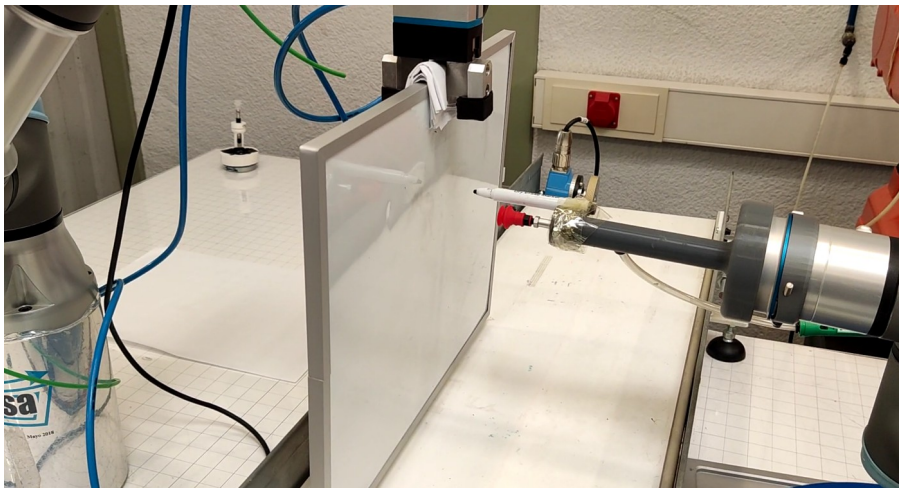

Figura 49: Prueba experimental Modo Fuerza (II)

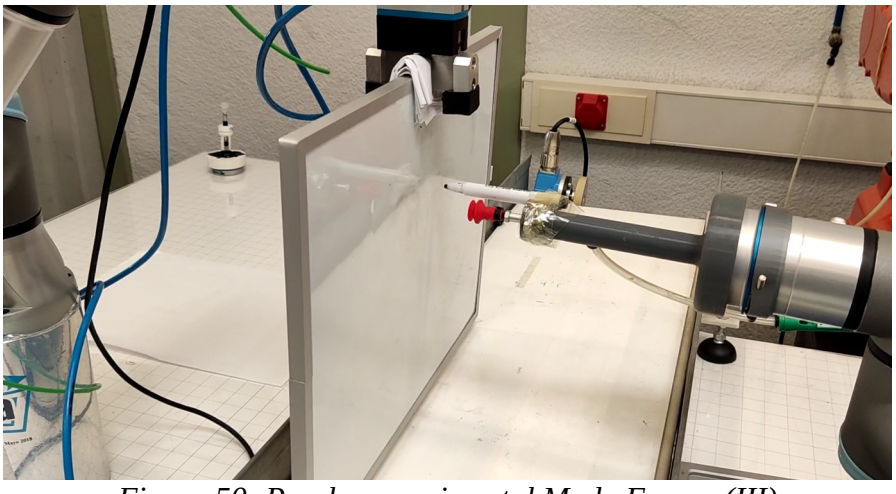

Figura 50: Prueba experimental Modo Fuerza (III)

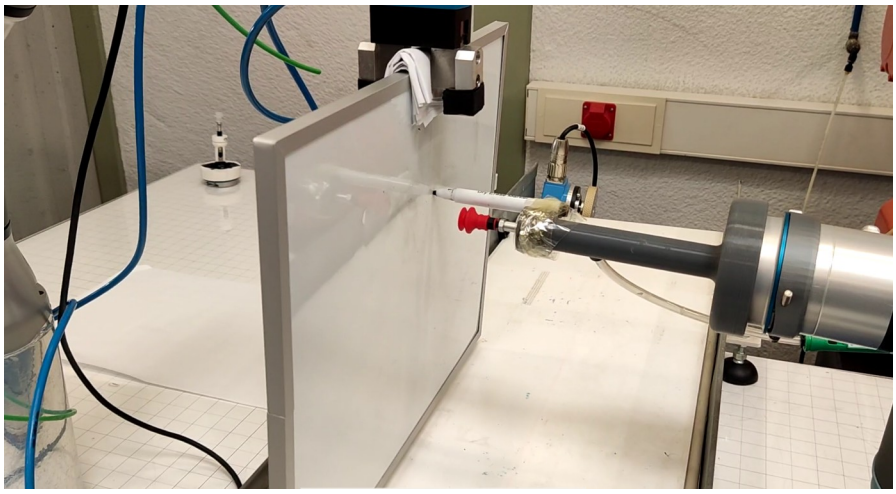

Figura 51: Prueba experimental Modo Fuerza (IV)

6.3. Diseño de un gemelo digital para el robot UR3e

6.3.1. Introducción - Primera versión sin realidad virtual

A la hora de querer desarrollar un gemelo digital para el robot UR3e se planteó la idea de desarrollarlo en la plataforma Unity ya que se encontró un proyecto en GitHub [13] que implementa una comunicación TCP/IP en Unity para comunicarse con el robot real.

Este proyecto incorpora un control cartesiano del robot mediante una interfaz que incluye botones y muestra los valores articulares del robot en grados junto con posición y orientación del extremo del robot. Al ser una comunicación TCP/IP fue necesario definir la IP del robot junto con los sockets de conexión.

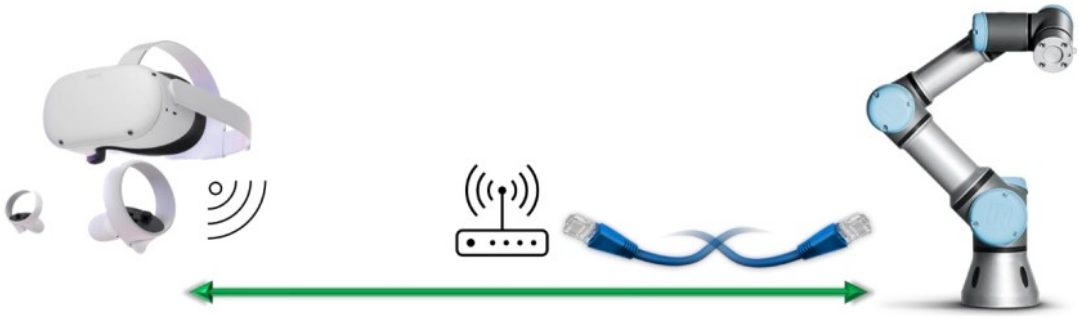

Figura 52: Comunicación por sockets directa con el robot [14]

Una vez hecha toda la configuración se probó con el robot para comprobar su correcto funcionamiento. Si queremos controlar el robot desde el PC será necesario activar el modo "Remote" en la consola de programación, de esta forma conseguiremos manejarlo. Si por el contrario, queremos mover el robot desde la consola de programación (pudiendo ver los cambios en tiempo real en el ordenador), será necesario activar el modo "Local Control".

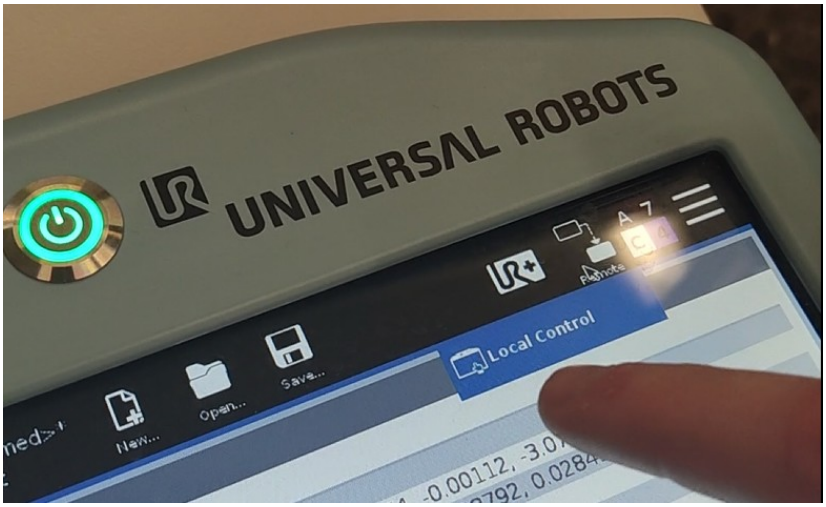

Figura 53: Modo Remote/Local Control robot UR3e

Una vez elegido el modo de funcionamiento solo tendremos que poner la IP del robot y darle al botón "Connect".

Figura 54: Conexión al robot

Una vez realizada la conexión entre nuestro gemelo digital desarrollado en Unity y el robot real, aparecerá una nueva interfaz que nos permitirá controlar el robot y visualizar los cambios en tiempo real.

Figura 55: Gemelo digital. Control UR3e desde el PC

Para una mejor comprensión de esta parte se recomienda visualizar el vídeo grabado a través de este enlace [10]. A continuación se muestran algunos fotogramas del vídeo:

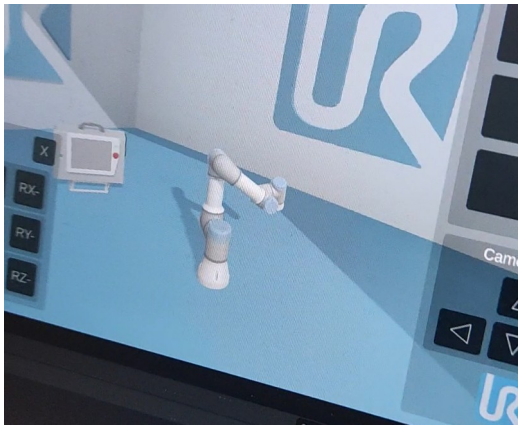

Figura 56: Prueba experimental Gemelo Digital (I)

Figura 57: Prueba experimental Gemelo Digital (II)

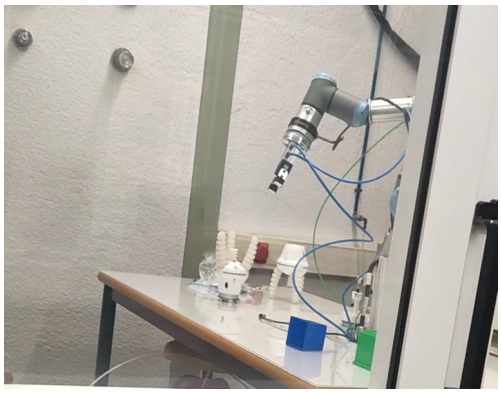

Figura 59: Prueba experimental Gemelo Digital (IV)

Figura 58: Prueba experimental Gemelo Digital (III)

6.3.2. Configuración proyecto para soporte de realidad virtual

Una vez probado el potencial del proyecto anterior se planteó la idea de realizar lo mismo pero para ser controlado con unas gafas de realidad virtual. Hay numerosas opciones en el mercado pero se optó por usar unas gafas Oculus Quest 2 ya que es una de las opciones más accesibles para entrar a la realidad virtual y se disponían de ellas.

Lo primero que se hizo fue crear un nuevo proyecto de Unity desde cero, teniendo en cuenta que las Oculus Quest 2 trabajan sobre el sistema operativo Android por lo que la base del proyecto tendrá que ser esta. Una vez creado el proyecto necesitaremos configurarlo para que tenga compatibilidad con las gafas Oculus instalando el "XR Plug-in Management".

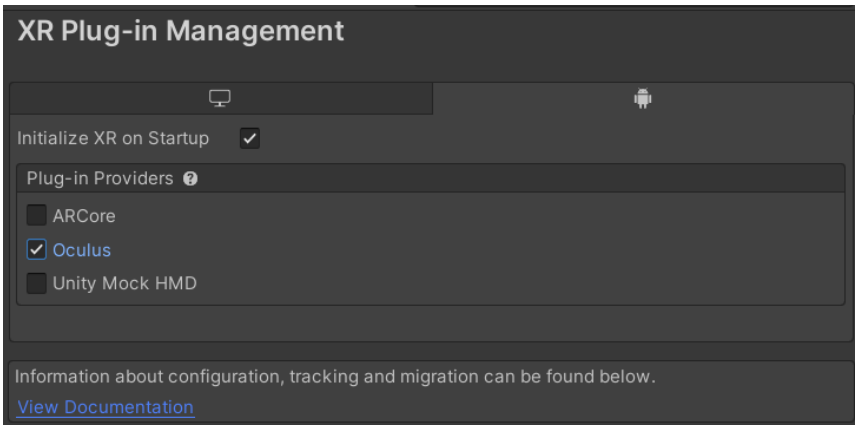

Figura 60: XR Plug-in Management (Integración con Oculus)

6.3.3. Creación entorno e integración de scripts

Unity además, ofrece una tienda en la que podemos descargar entornos creados por la gente entre los cuales algunos serán gratis. En este proyecto se ha descargado un “Asset” de un garaje el cual modificaremos posteriormente. [16]

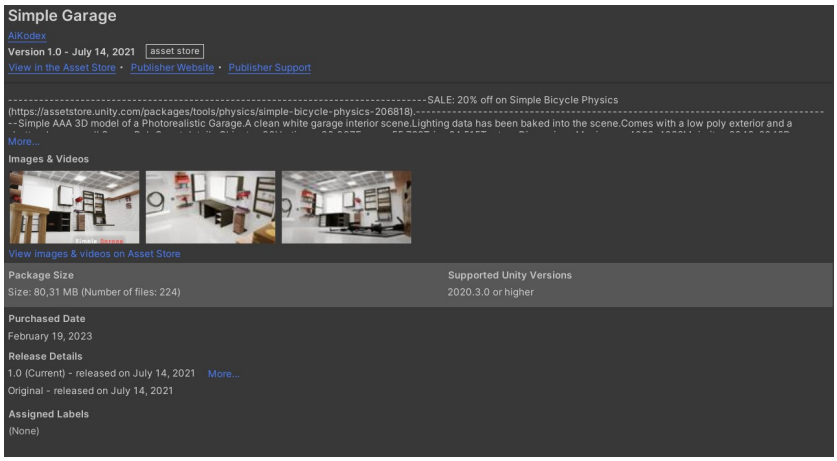

Figura 61: Importar Simple Garage [16]

Figura 62: Simple Garage entorno [16]

Una vez tenemos la escena creada (que se modificará posteriormente) insertaremos en el proyecto el “OculusInteractionSampleRig” que es una herramienta en la que vienen por defecto la configuración de controles y la cámara de las Oculus Quest 2.

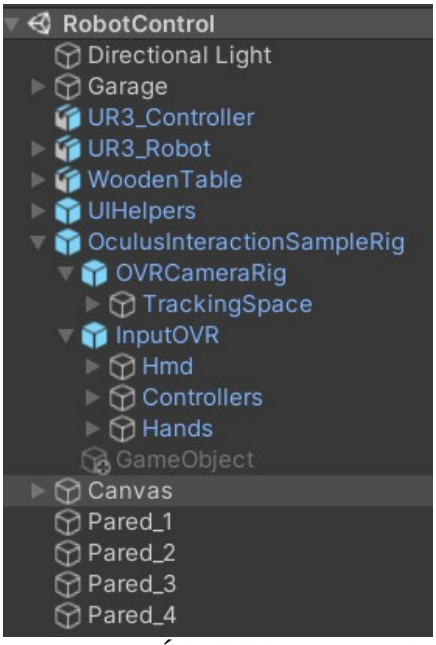

Figura 63: Árbol del proyecto de Unity

Además, será necesario añadir a la cámara del jugador un "Ridigbody" que dota de propiedades físicas permitiendo movernos y responder a fuerzas físicas como colisiones y de un "Capsule Collider" que se usa para calcular las interacciones de colisión con otros objetos y detectar colisiones mediante programación.

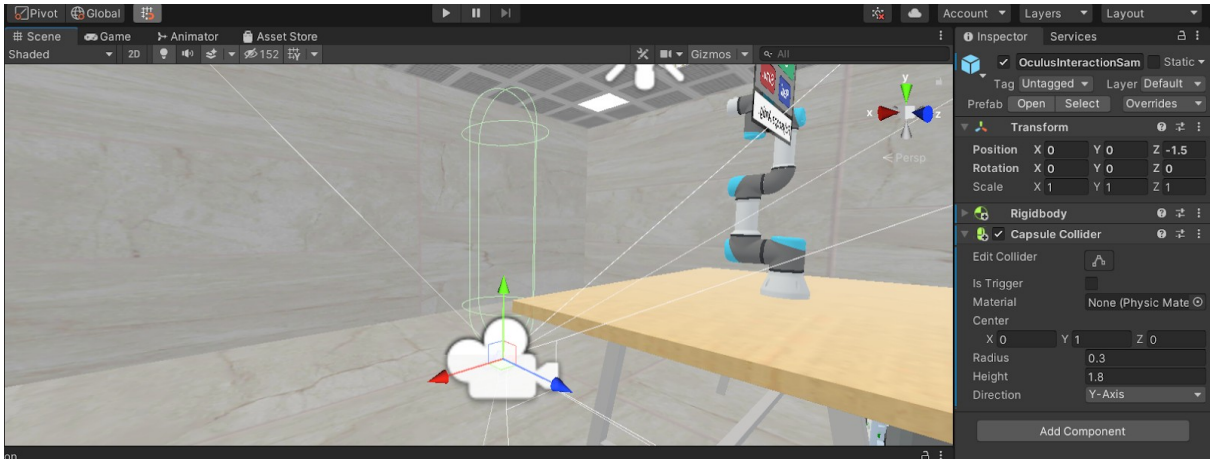

Figura 64: Añadir físicas y colisiones a la cámara (Unity)

Al entorno creado, añadiremos el modelo en blender de la caja de programación y del robot UR3e a Unity sacado del proyecto mencionado con anterioridad. [13]

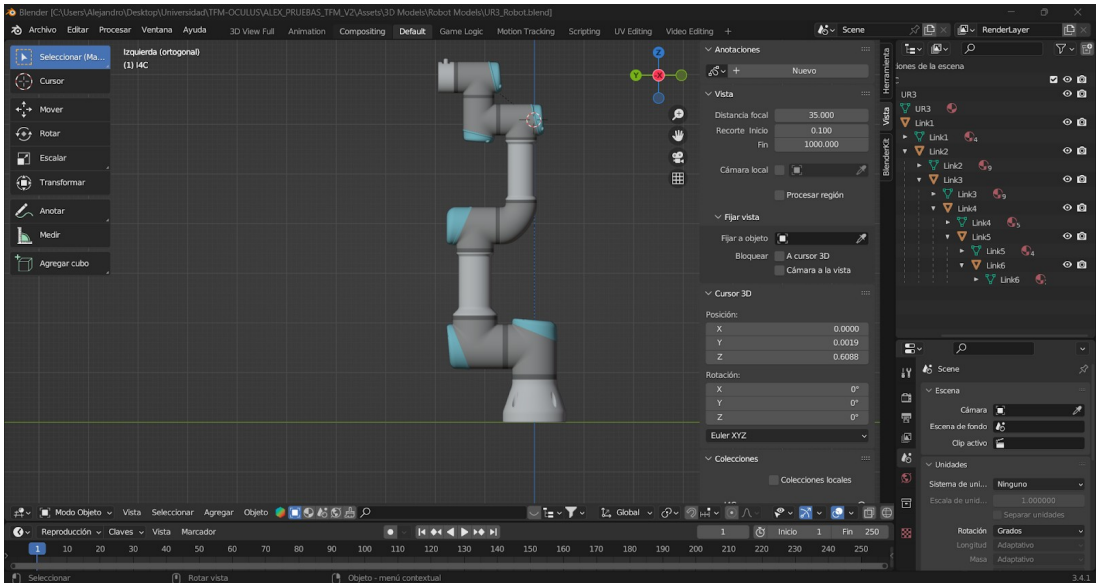

Figura 65: Modelo blender UR3e [13]

Para que el modelo del robot presente la misma configuración que el robot real se añadió a cada eslabón un script [13] que modifica la rotación del eslabón sobre sus propios ejes aplicando el valor articular de esa articulación del robot real. [14]

Además, se ha añadido una esfera visible que muestra el espacio de trabajo del robot. Esta esfera se volverá invisible cuando el extremo del robot está dentro de ella y se podrá visualizar cuando llegue a tocar la misma. Para terminar, se modificó el entorno del garaje [15] dejando los mínimos elementos posibles para afectar lo menos posible a los fotogramas.

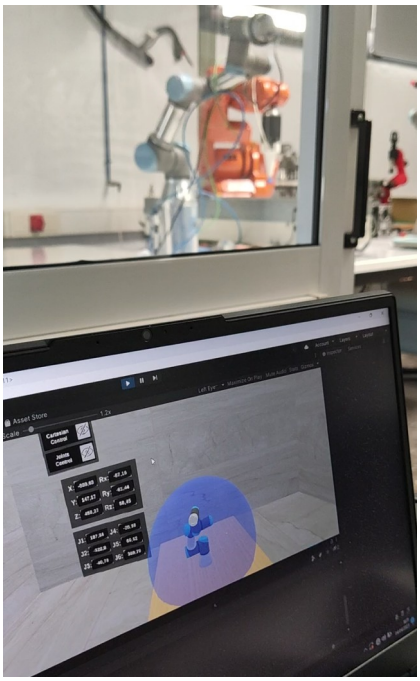

Figura 66: Espacio trabajo robot visible UR3e

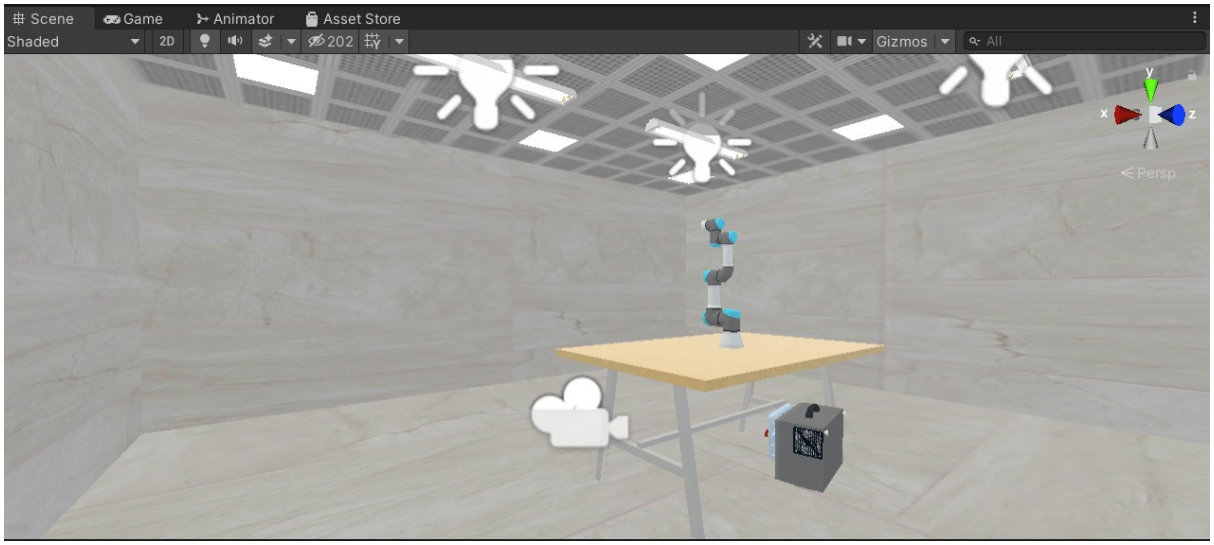

Figura 67: Entorno final Unity

6.3.4. Interfaz de usuario

Todas los ventanas de la interfaz gráfica se encuentran centrados sobre el robot y están contenidos dentro de un objeto “Canvas” de Unity. Estos son componentes utilizados para crear interfaces de usuario (UI) en aplicaciones y en ellos se pueden integrar elementos de la interfaz como botones, texto, imágenes… [14]

A continuación hablaremos de las principales ventanas definidas en nuestro proyecto:

- **Pantalla de conexión:** En ella introduciremos la IP del robot real y la versión de Polyscope usada. Para no tener que introducir la IP manualmente se introdujo un botón “Real” para insertar automáticamente la IP del robot (192.168.1.49). Además, se podría conectar a un robot de la máquina virtual proporcionado por la propia Universal Robots. Esto último no se probó por problemas con la máquina virtual.

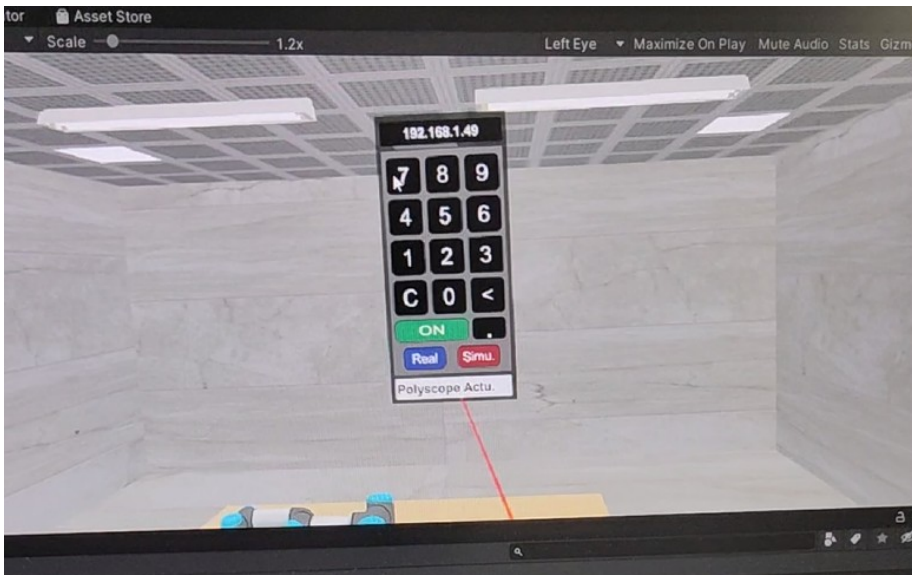

Figura 68: Pantalla de conexión Unity

- **Pantalla de pose del TCP y valores articulares:** Estos datos son proporcionados por la controladora del UR3e. Los valores serán pasados de metros a milímetros para la posición y de radianes a grados para la orientación.

Figura 69: Posición y orientación del robot UR3e

- **Pantalla control cartesiano:** El control cartesiano será útil para mover el robot en los ejes cartesianos de la base en la que controlaremos desplazamientos y rotaciones (positivos y negativos) en los tres ejes. Para lograr este control fue necesario usar la instrucción "speedl(xd, a , t)" del robot siendo "xd" la velocidad de la herramienta del robot, "a" la aceleración del codo del robot y "t" el tiempo de duración de la instrucción. Cuando se pulsa un botón la velocidad se mantiene fija en 0.1 m/s para el caso de desplazamientos y 0.1 rad/s en el caso de rotaciones. Para el caso del tiempo se ha fijado en 0.05 segundos. [14]

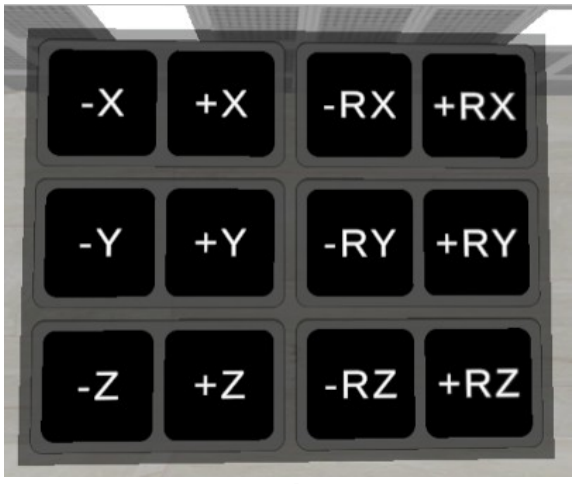

Figura 70: Control cartesiano UR3e

- **Pantalla control individual articulación:** Podemos controlar cada una de las seis articulaciones del robot por separado pudiendo variar la velocidad de las mismas con un slider entre 0.1 rad/s y 1 rad/s. Para este control se usa la instrucción “speedj(qd,a,t)” siendo “qd” la velocidad de cada articulación del robot en rad/s, “a” la aceleración del codo del robot en rad/s^2 y “t” el tiempo de duración de la instrucción. [14]

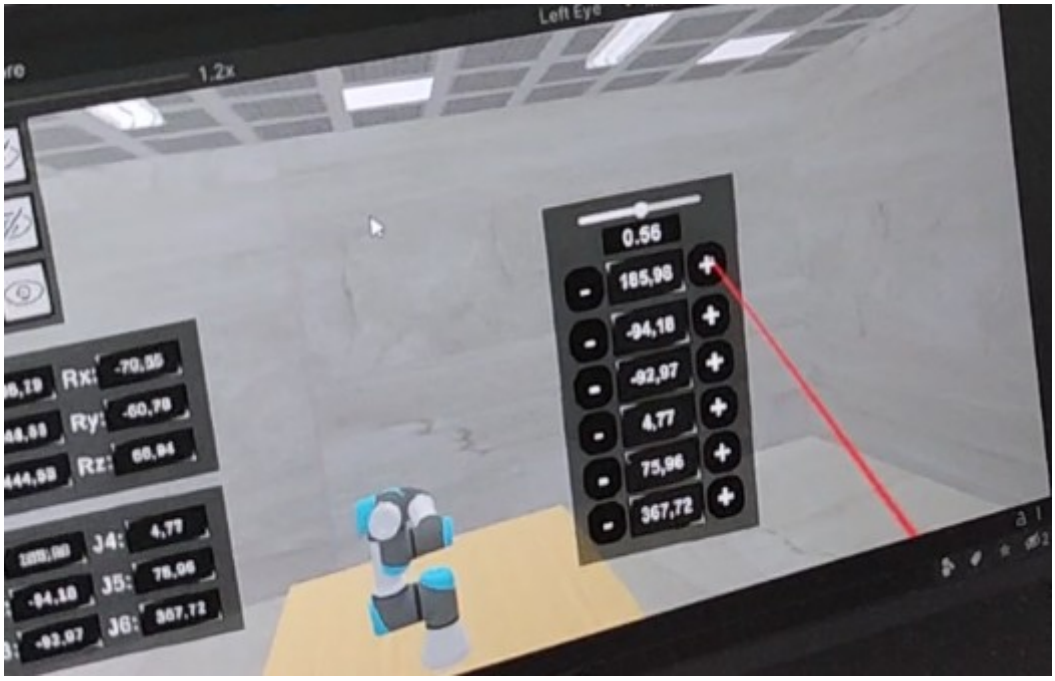

Figura 71: Control articulación UR3e

6.3.5. Parada de emergencia

Entre los valores que se manda el robot real existe un parámetro que indica el estado de seguridad “Safety Status” que toma distintos valores. [16]

	Value	Comment
SAFETY_MODE_UNDEFINED_ SAFETY_MODE	11	
SAFETY_MODE_VALIDATE_ JOINT_ID	10	
SAFETY_MODE_FAULT	9	
SAFETY_MODE_VIOLATION	8	
SAFETY_MODE_ROBOT_ EMERGENCY_STOP	7	(EA + EB + SBUS->Euromap67) Physical e-stop interface input activated
SAFETY_MODE_SYSTEM_ EMERGENCY_STOP	6	(EA + EB + SBUS->Screen) Physical e-stop interface input activated
SAFETY_MODE_SAFEGUARD_ STOP	5	(SI0 + SI1 + SBUS) Physical s-stop interface input
SAFETY_MODE_RECOVERY	4	
SAFETY_MODE_PROTECTIVE_ STOP	3	
SAFETY_MODE_REDUCED	2	
SAFETY_MODE_NORMAL	1	

Figura 72: Safety Status UR3e [16]

El valor del estado normal es 1 por lo que, si manda algún valor distinto a este se pararán todas las órdenes que enviemos y nos mostrará una ventana de aviso.

Figura 73: Ventana de parada de emergencia en Unity

6.3.6. Control manual del robot

El control manual se activará una vez se pulse el gatillo del control derecho quedando vinculada la posición del extremo del robot a la del mando. Al pulsar el gatillo se sustituirá la imagen del mando por la del extremo del robot y podremos pasar a moverlo con el movimiento de nuestra mano. [14]

Para lograr enviar la información se cogerá los valores de los sensores del mando y se usará la misma instrucción que en el control cartesiano.

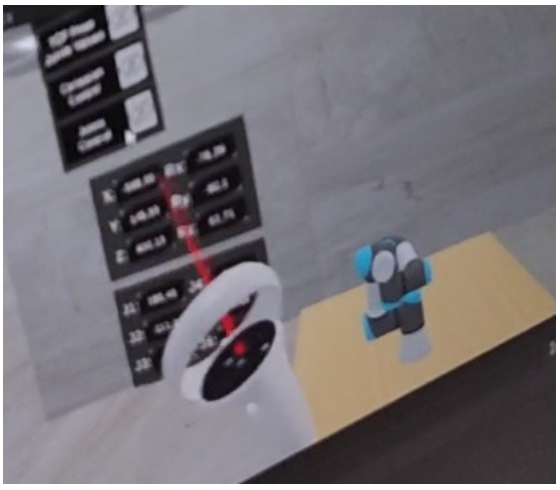

Figura 74: Vista del controlador en Unity

Figura 75: Control manual del robot UR3e

6.3.7. Comunicación con el robot TCP/IP

Para la comunicación TCP/IP entre el robot real y el gemelo digital se usarán los siguientes puertos:

- **Puerto 30002:** Para enviar instrucciones al robot.
- **Puerto 30003:** Para recibir información del robot.

e-Series							
	Primary		Secondary		Real-time		Real-time Data Exchange (RTDE)
Port no.	30001	30011	30002	30012	30003	30013	30004
Frequency [Hz]	10	10	10	10	500	500	500
Receive	URScript commands	-	URScript commands	-	URScript commands	-	Various data
Transmit	See attachment from the bottom		See attachment from the bottom		See attachment from the bottom		See RTDE Guide

CB-Series							
	Primary		Secondary		Real-time		Real-time Data Exchange (RTDE)
Port no.	30001	30011	30002	30012	30003	30013	30004
Frequency [Hz]	10	10	10	10	125	125	125
Receive	URScript commands	-	URScript commands	-	URScript commands	-	Various data
Transmit	See attachment from the bottom		See attachment from the bottom		See attachment from the bottom		See RTDE Guide

Figura 76: Puertos TCP/IP UR3 [17]

Además, hay que tener en cuenta que según la versión de polyscope usada, el tamaño de los mensajes variará.

6.3.8. Resultados finales

Una vez comentadas las distintas partes del proyecto del gemelo digital, se recomienda visualizar el siguiente vídeo grabado para una mejor comprensión del mismo. [11]

Por último, destacar que todos los códigos usados se pueden consultar en el "*ANEXO B*". [13][14]

Figura 77: Robot UR3e con Oculus Quest 2

A continuación se muestran algunos fotogramas del vídeo:

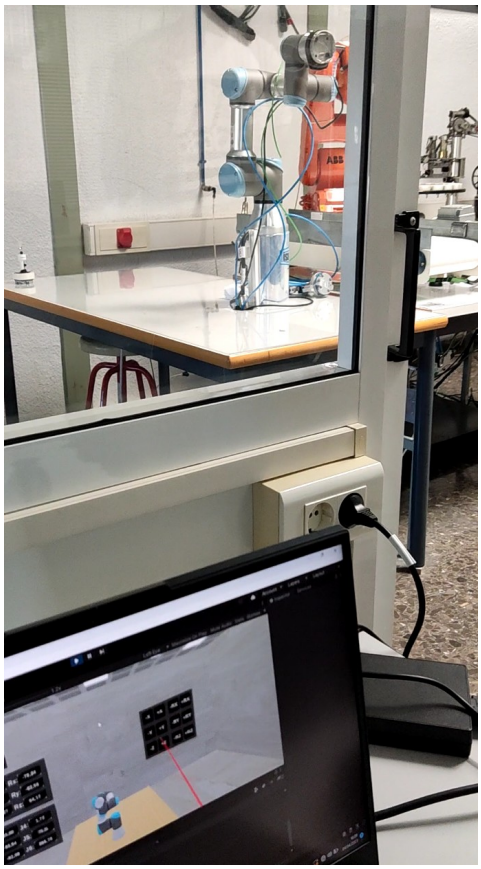

Figura 78: Prueba experimental Gemelo Digital VR (I)

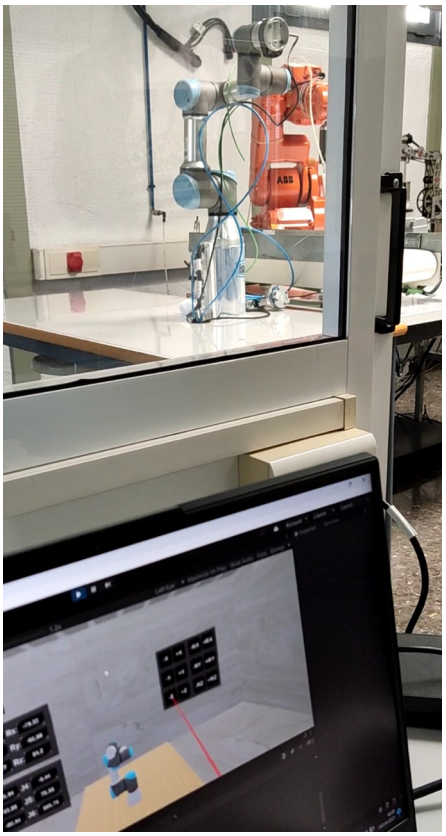

Figura 79: Prueba experimental Gemelo Digital VR (II)

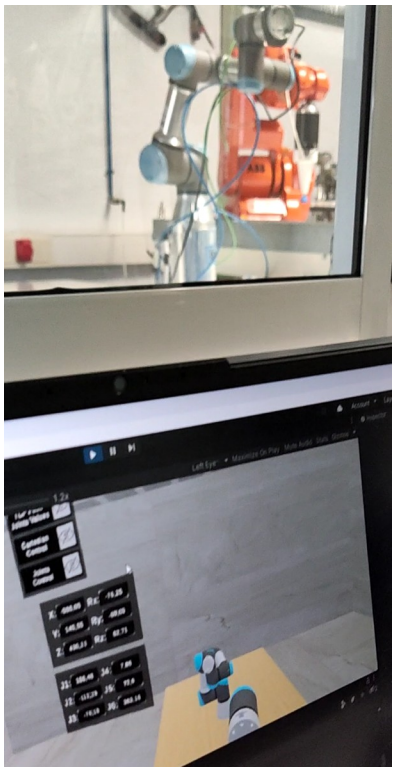

Figura 80: Prueba experimental Gemelo Digital VR (III)

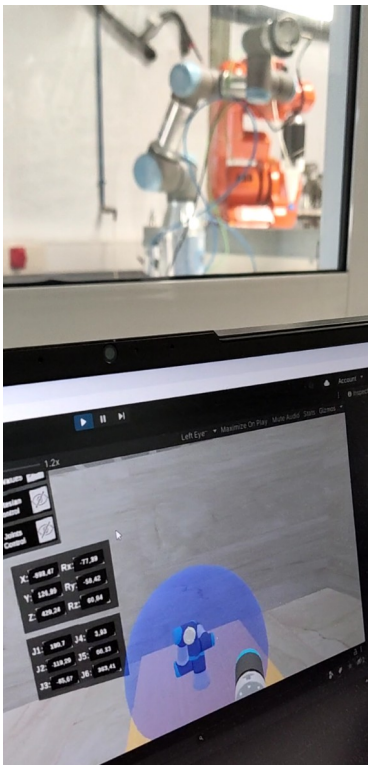

Figura 81: Prueba experimental Gemelo Digital VR (IV)

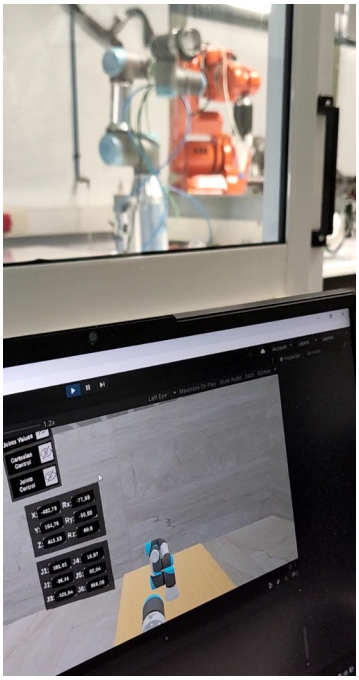

Figura 82: Prueba experimental Gemelo Digital VR (V)

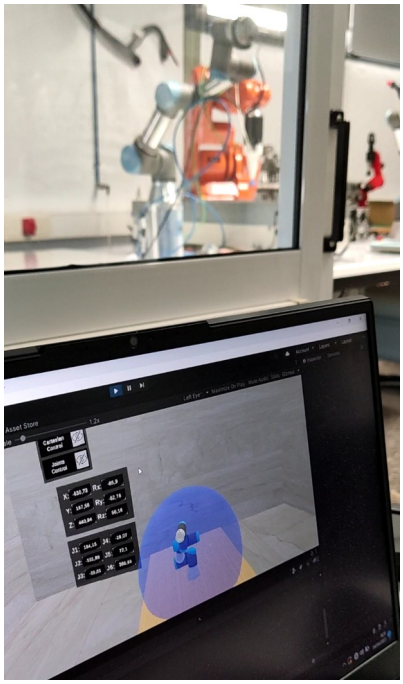

Figura 83: Prueba experimental Gemelo Digital VR (VI)

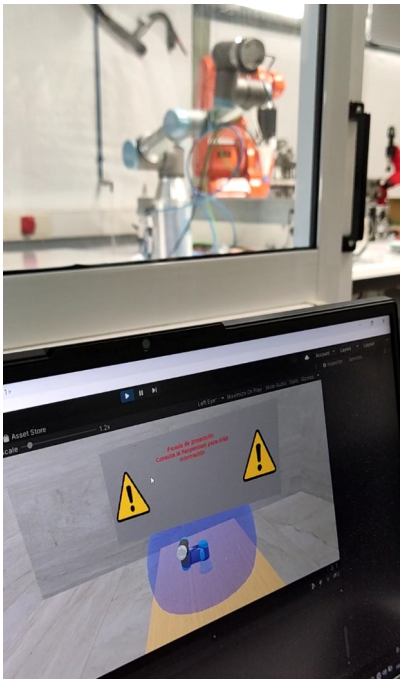

Figura 84: Prueba experimental Gemelo Digital VR (VII)

7. Conclusiones y posibles mejoras

Hay que tener en cuenta que en el control manual del gemelo digital, al usar coordenadas cartesianas es posible tener problemas por acercarse a una singularidad. Este problema surgió haciendo las pruebas de este proyecto pero no es posible solucionarse a no ser que se añada algún tipo de excepción al pasar cerca de una singularidad.

Una posible mejora del gemelo digital hubiese sido la integración de una herramienta al modelo blender del UR3e para poder controlarla también como el resto del robot pero, al no saber hacer modelos se intentó buscar uno que coincidiese con la herramienta del laboratorio pero no fue posible.

En las pruebas del control de fuerza hubiese sido interesante integrar un control por impedancia o admitancia y compararlos entre ellos.

Como conclusión he de decir que se han logrado cumplir los objetivos planteados en el trabajo y se han logrado aplicar muchos de los conocimientos impartidos en el máster como la visión artificial, comunicaciones TCP/IP, programación de robots... Además, se tocan diversas áreas con una gran proyección de futuro.

8. Bibliografía

[1] F. Sherwani, M. M. Asad and B. S. K. K. Ibrahim, "Collaborative Robots and Industrial Revolution 4.0 (IR 4.0)," 2020 International Conference on Emerging Trends in Smart Technologies (ICETST), Karachi, Pakistan, 2020, pp. 1-5, doi: 10.1109/ICETST49965.2020.9080724.

[2] Gracia Calandin, Luis Ignacio. (2022-2023). Apuntes de la asignatura “Control y programación de robots”. Universidad Politécnica de Valencia

[3] Universal Robots. (s.f.). Robot UR3. Universal Robots. Disponible en: https://www.universal-robots.com/es/productos/robot-ur3/

[4] Universal Robots. (2022). DH Parameters for Calculations of Kinematics and Dynamics. Universal Robots. Disponible en: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/

[5] Robotiq. (2019). Vision System e-Series (Manual en PDF). Disponible en: https://assets.robotiq.com/website-assets/support_documents/document/Vision_System_e-Series_PDF_20190116.pdf

[6] Meta. (s.f.). Tech Specs - Meta Quest 2. Meta. Disponible en: https://www.meta.com/es/quest/products/quest-2/tech-specs/#tech-specs

[7] Universal Robots. (s.f.). Formación en línea de e-Series. Universal Robots Academy. Disponible en: https://academy.universal-robots.com/es/formacion-en-linea-gratuita/formacion-en-linea-de-e-series/

[8] Martín Doñas, Alejandro. (2023). Título del vídeo [Pick & Place colaborativo UR3e con visión]. YouTube. Disponible en: https://www.youtube.com/watch?v=mZuqk9eWnQI

[9] Martín Doñas, Alejandro. (2023). Título del vídeo [Prueba modo fuerza del UR3e]. YouTube. Disponible en: https://www.youtube.com/watch?v=iPaLoac10y8

[10] Martín Doñas, Alejandro. (2023). Título del vídeo [Gemelo Digital UR3e (sin VR)]. YouTube. Disponible en: https://www.youtube.com/shorts/clzXefgaWPI

[11] Martín Doñas, Alejandro. (2023). Título del vídeo [Gemelo Digital UR3e con gafas de realidad virtual (Oculus Quest 2)]. YouTube. Disponible en: https://www.youtube.com/watch?v=_v__dzRaNjU

[12] Universal Robots. (s.f.). Manual de Usuario de los Robots e-Series (Manual en PDF). Disponible en: https://s3-eu-west-1.amazonaws.com/ur-support-site/105224/99436_UR3e_User_Manual_es_Global.pdf

[13] Parak, Roman. (2020-2021). A digital-twins in the field of industrial robotics integrated into the unity3d development platfortm. Github. Disponible en: https://github.com/rparak/Unity3D_Robotics_Overview

[14] Alonso Fernández, Ricardo. (2022). Interfaz inmersiva en realidad virtual para el control de un robot industrial “http://hdl.handle.net/10045/127768”. Universidad de Alicante, España.

[15] AiKodex. (2021). Simple Garage, Unity Asset Store. Disponible en: https://assetstore.unity.com/packages/3d/props/interior/simple-garage-197251

[16] Universal Robots. (s.f.). Realtime Client Interface (Documento en PDF). Disponible en: https://s3-eu-west-1.amazonaws.com/ur-support-site/16496/ClientInterfaces_Realtime.pdf

[17] Universal Robots. (2015). Remote Control via TCP/IP. Universal Robots. Disponible en: https://www.universal-robots.com/articles/ur/interface-communication/remote-control-via-tcpip/

[18] Valiente González, José Miguel. (2022-2023). Apuntes de la asignatura “Visión por computador en la industria”. Universidad Politécnica de Valencia

Anexos

ANEXO A.- PICK & PLACE

A1. Código RAPID ABB. Sensores y actuadores cinta transportadora

```
MODULE Comunicacion
  !***********************************************************
  !
  ! Module:  Unnamed
  !
  ! Description:
  !   <Insert description here>
  !
  ! Author: ALEJANDRO
  !
  ! Version: 1.0
  !
  !***********************************************************



  !***********************************************************
  !
  ! Procedure main
  !
  !   This is the entry point of your program
  !
  !***********************************************************
  PROC main()
```

```
    !Add your code here
    !Se hace un reset a todas las variables
    Reset CONVEYOR_FWD_INVERT;
  Reset CONVEYOR_FWD;
  Reset CONVEYOR_BWD;

    WHILE TRUE DO

!Si la entrada DI10_6 se activa (Lee salida DO1 del UR3)
    !Pone en marcha la cinta en la direcciÃ³n 1
    IF DI10_6=1 THEN
      Set CONVEYOR_BWD;
    ELSE
      Reset CONVEYOR_BWD;
    ENDIF
    !Si la entrada DI10_7 se activa (Lee salida DO2 del UR3)
    !Pone en marcha la cinta en la direcciÃ³n 2
    IF DI10_7=1 THEN
      Set CONVEYOR_FWD;
    ELSE
      Reset CONVEYOR_FWD;
    ENDIF

    !Para el sensor
    !Si detecta el sensor el ABB se activa la entrada DI4 del UR3
    IF CONVEYOR_OBJ_SEN=1 THEN
       Set CONVEYOR_FWD_SENSOR_ENABLE;
    ELSE
       Reset CONVEYOR_FWD_SENSOR_ENABLE;
```

```
      ENDIF


      ENDWHILE


   ENDPROC
ENDMODULE
```

A2. Estructura programación polyscope UR3e (I)

```
Program
  Robot Program
    MoveJ
      P1
    Camera Locate
      For object(s) found
        MoveJ
          Waypoint_1
          Waypoint_2
          Set DO[5]=On
          Wait: 1.0
        MoveJ
          P1
          Waypoint_5
        MoveJ
          Waypoint_3
          Waypoint_4
          Set DO[5]=Off
          Wait: 1.0
          P1
```

```
Thread_1
  Wait DI[6]=HI
  Wait: 2.0
  Set DO[2]=On
  Wait DI[5]=HI
  Set DO[2]=Off
```

A3. Estructura programación polyscope UR3e (II)

```
Program
  Robot Program
    MoveL
      Waypoint_1
      Wait DI[5]=HI
    MoveJ
      Waypoint_2
    MoveJ
      Waypoint_3
      Set DO[5]=On
      Wait: 1.0
    MoveL
      Waypoint_2
    MoveJ
      Waypoint_4
    MoveL
      Waypoint_5
      Set DO[5]=Off
    MoveL
      Waypoint_1
```

ANEXO B.- Gemelo Digital. Unity C#

B1. Movimiento de las articulaciones UR3e

B1.1. Articulación 1

```
// System
using System;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur3_link1 : MonoBehaviour
{
    void FixedUpdate()
    {
        try
        {
            transform.localEulerAngles = new Vector3(0.0f, (-1.0f) * (float)
(ur_data_processing.UR_Stream_Data.J_Orientation[0] * (180.0 / Math.PI)), 0.0f);
        }
        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
    }
    void OnApplicationQuit()
    {
        Destroy(this);
    }
}
```

B1.2. Articulación 2

```
// System
using System;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur3_link2 : MonoBehaviour
{
    void FixedUpdate()
    {
        try
        {
            transform.localEulerAngles = new Vector3(0.0f, 0.0f, 90.0f + (float)
(ur_data_processing.UR_Stream_Data.J_Orientation[1] * (180.0 / Math.PI)));
        }
        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
    }
    void OnApplicationQuit()
```

```
    {
        Destroy(this);
    }
}
```

B1.3. Articulación 3

```
// System
using System;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur3_link3 : MonoBehaviour
{
    void FixedUpdate()
    {
        try
        {
            transform.localEulerAngles = new Vector3(0.0f, 0.0f, (float)
(ur_data_processing.UR_Stream_Data.J_Orientation[2] * (180.0 / Math.PI)));

        }
        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
    }
    void OnApplicationQuit()
    {
        Destroy(this);
    }
}
```

B1.4. Articulación 4

```
// System
using System;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur3_link4 : MonoBehaviour
{
    void FixedUpdate()
    {
        try
        {
            transform.localEulerAngles = new Vector3(0.0f, 0.0f, 90.0f + (float)
(ur_data_processing.UR_Stream_Data.J_Orientation[3] * (180.0 / Math.PI)));
        }
        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
```

```
    }
    void OnApplicationQuit()
    {
        Destroy(this);
    }
}
```

B1.5. Articulación 5

```
// System
using System;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur3_link5 : MonoBehaviour
{
    void FixedUpdate()
    {
        try
        {
            transform.localEulerAngles = new Vector3(0.0f, (-1.0f) * (float)
(ur_data_processing.UR_Stream_Data.J_Orientation[4] * (180.0 / Math.PI)), 0f);
        }
        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
    }
    void OnApplicationQuit()
    {
        Destroy(this);
    }
}
```

B1.6. Articulación 6

```
// System
using System;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur3_link6 : MonoBehaviour
{
    void FixedUpdate()
    {
        try
        {
            transform.localEulerAngles = new Vector3(0.0f, 0.0f, (float)
(ur_data_processing.UR_Stream_Data.J_Orientation[5] * (180.0 / Math.PI)));
        }
        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
```

```
    }
    void OnApplicationQuit()
    {
        Destroy(this);
    }
}
```

B2. Comunicación con el robot

```
// System
using System;
using System.Diagnostics;
using System.Net.Sockets;
using System.Threading;
// Unity
using UnityEngine;
using Debug = UnityEngine.Debug;
public class ur_data_processing : MonoBehaviour
{
    public enum versions
    {
        V3_15,
        V3_14
    };
    public static class GlobalVariables_Main_Control
    {
        public static bool connect, disconnect;
        public static versions polyscopeVersion;

    }

    public string IP ="192.168.8.128";
    public static class UR_Stream_Data
    {
        // IP Port Number and IP Address
        public static string ip_address;
        //  Real-time (Read Only)
        public const ushort port_number = 30013;//30013;
        // Comunication Speed (ms)
        public static int time_step;
        // Joint Space:
        //  Orientation {J1 .. J6} (rad)
        public static double[] J_Orientation = new double[6];
        // Cartesian Space:
        //  Position {X, Y, Z} (mm)
        public static double[] C_Position = new double[3];
        //  Orientation {Euler Angles} (rad):
        public static double[] C_Orientation = new double[3];
        public static double SafetyStatus;
        // Class thread information (is alive or not)
        public static bool is_alive = false;
    }
    public static class UR_Control_Data
    {
        // IP Port Number and IP Address
        public static string ip_address;
        //  Real-time (Read/Write)
        public const ushort port_number = 30003;//30003;
```

```
        // Comunication Speed (ms)
        public static int time_step;
        // Control Parameters UR3/UR3e:
        public static string aux_command_str;
        public static byte[] command;
        public static bool[] button_pressed = new bool[12];
        public static bool joystick_button_pressed;
        // Class thread information (is alive or not)
        public static bool is_alive = false;

    }
    // Class Stream / Control {Universal Robots TCP/IP}
    private UR_Stream ur_stream_robot;
    private UR_Control ur_ctrl_robot;
    // Other variables
    private int main_ur3_state = 0;
    private int aux_counter_pressed_btn = 0;
    // Start is called before the first frame update
    void Start()
    {
        // Initialization {TCP/IP Universal Robots}
        //  Read Data:
        //UR_Stream_Data.ip_address = "192.168.2.129"; //"127.0.0.1";
        //UR_Stream_Data.ip_address = "192.168.8.128";
        UR_Stream_Data.ip_address = IP;
        //  Communication speed: CB-Series 125 Hz (8 ms), E-Series 500 Hz (2 ms)
        UR_Stream_Data.time_step = 2;
        //  Write Data:
       // UR_Control_Data.ip_address = "192.168.2.129"; //"127.0.0.1";
        UR_Control_Data.ip_address = IP;
        //  Communication speed: CB-Series 125 Hz (8 ms), E-Series 500 Hz (2 ms)
        UR_Control_Data.time_step = 2;
        // Initialization Stream {Universal Robots TCP/IP}
        ur_stream_robot = new UR_Stream();
        // Start Control {Universal Robots TCP/IP}
        ur_ctrl_robot = new UR_Control();
        switch (GlobalVariables_Main_Control.polyscopeVersion)
        {
            case versions.V3_15:
                ur_stream_robot.total_msg_length = 3288596480;
                break;
            case versions.V3_14:
                ur_stream_robot.total_msg_length = 1946419200;
                break;

        }

    }
    // Update is called once per frame
    void FixedUpdate()
    {
        switch (GlobalVariables_Main_Control.polyscopeVersion)
        {
            case versions.V3_15:
                ur_stream_robot.total_msg_length = 3288596480;
                break;
            case versions.V3_14:
                ur_stream_robot.total_msg_length = 1946419200;
                break;
```

```
        }
        switch (main_ur3_state)
        {
            case 0:
                {
                    // ------------------------ Wait State {Disconnect State}
-----------------------//
                    if (GlobalVariables_Main_Control.connect == true)
                    {
                        // Start Stream {Universal Robots TCP/IP}
                        ur_stream_robot.Start();
                        // Start Control {Universal Robots TCP/IP}
                        ur_ctrl_robot.Start();
                        // go to connect state
                        main_ur3_state = 1;
                    }
                }
                break;
            case 1:
                {
                    // ------------------------ Data Processing State {Connect State}
-----------------------//
                    for (int i = 0; i < UR_Control_Data.button_pressed.Length; i++)
                    {
                        // check the pressed button in joystick control mode
                        if (UR_Control_Data.button_pressed[i] == true)
                        {
                            aux_counter_pressed_btn++;
                        }
                    }
                    // at least one button pressed
                    if (aux_counter_pressed_btn > 0)
                    {
                        // start move -> speed control
                        UR_Control_Data.joystick_button_pressed = true;
                    }
                    else
                    {
                        // stop move -> speed control
                        UR_Control_Data.joystick_button_pressed = false;
                    }
                    // null auxiliary variable
                    aux_counter_pressed_btn = 0;
                    if (GlobalVariables_Main_Control.disconnect == true)
                    {
                        // Stop threading block {TCP/Ip -> read data}
                        if (UR_Stream_Data.is_alive == true)
                        {
                            ur_stream_robot.Stop();
                        }
                        // Stop threading block {TCP/Ip  -> write data}
                        if (UR_Control_Data.is_alive == true)
                        {
                            ur_ctrl_robot.Stop();
                        }
                        if (UR_Stream_Data.is_alive == false && UR_Control_Data.is_alive ==
false)
                        {
```

```
                            // go to initialization state {wait state -> disconnect state}
                            main_ur3_state = 0;
                        }
                    }
                }
                break;
        }
    }
    void OnApplicationQuit()
    {
        try
        {
            // Destroy Stream {Universal Robots TCP/IP}
            ur_stream_robot.Destroy();
            // Destroy Control {Universal Robots TCP/IP}
            ur_ctrl_robot.Destroy();
            Destroy(this);
        }
        catch (Exception e)
        {
            Debug.Log("Application Quit Exception:" + e);
        }
    }
    class UR_Stream
    {
        // Initialization of Class variables
        //  Thread
        private Thread robot_thread = null;
        private bool exit_thread = false;
        //  TCP/IP Communication
        private TcpClient tcp_client = new TcpClient();
        private NetworkStream network_stream = null;
        //  Packet Buffer (Read)
        private byte[] packet = new byte[1116];
        // Offset:
        //  Size of first packet in bytes (Integer)
        private const byte first_packet_size = 4;
        //  Size of other packets in bytes (Double)
        private const byte offset = 8;
        // Total message length in bytes
        // polyscope version 3.15
        public UInt32 total_msg_length = 3288596480;
        //polyscope version 3.14
        //public const UInt32 total_msg_length = 1946419200;

        public void UR_Stream_Thread()
        {
            try
            {
                if (tcp_client.Connected == false)
                {
                    // Connect to controller -> if the controller is disconnected
                    tcp_client.Connect(UR_Stream_Data.ip_address,
UR_Stream_Data.port_number);
                }

                // Initialization TCP/IP Communication (Stream)
                network_stream = tcp_client.GetStream();
```

```
                // Initialization timer
                var t = new Stopwatch();
                while (exit_thread == false)
                {

                    // Get the data from the robot
                    if (network_stream.Read(packet, 0, packet.Length) != 0)
                    {

                        if (BitConverter.ToUInt32(packet, first_packet_size - 4) ==
total_msg_length)
                        {
                            // t_{0}: Timer start.
                            t.Start();
                            // Reverses the order of elements in a one-dimensional array or
part of an array.
                            Array.Reverse(packet);
                            // Note:
                            //  For more information on values 32... 37, etc., see the UR
Client Interface document.
                            // Read Joint Values in radians

                            UR_Stream_Data.J_Orientation[0] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (32 * offset));
                            UR_Stream_Data.J_Orientation[1] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (33 * offset));
                            UR_Stream_Data.J_Orientation[2] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (34 * offset));
                            UR_Stream_Data.J_Orientation[3] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (35 * offset));
                            UR_Stream_Data.J_Orientation[4] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (36 * offset));
                            UR_Stream_Data.J_Orientation[5] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (37 * offset));
                            // Read Cartesian (Positon) Values in metres
                            UR_Stream_Data.C_Position[0] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (56 * offset));
                            UR_Stream_Data.C_Position[1] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (57 * offset));
                            UR_Stream_Data.C_Position[2] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (58 * offset));
                            // Read Cartesian (Orientation) Values in metres
                            UR_Stream_Data.C_Orientation[0] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (59 * offset));
                            UR_Stream_Data.C_Orientation[1] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (60 * offset));
                            UR_Stream_Data.C_Orientation[2] = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (61 * offset));
                            UR_Stream_Data.SafetyStatus = BitConverter.ToDouble(packet,
packet.Length - first_packet_size - (139 * offset));

                            t.Stop();
                            // Recalculate the time: t = t_{1} - t_{0} -> Elapsed Time in
milliseconds
                            if (t.ElapsedMilliseconds < UR_Stream_Data.time_step)
                            {
                                Thread.Sleep(UR_Stream_Data.time_step -
(int)t.ElapsedMilliseconds);
                            }
```

```
                        // Reset (Restart) timer.
                        t.Restart();
                    }
                }
            }
        }
        catch (SocketException e)
        {
            Console.WriteLine("SocketException: {0}", e);
        }
    }
    public void Start()
    {
        // Start thread
        exit_thread = false;
        // Start a thread and listen to incoming messages
        robot_thread = new Thread(new ThreadStart(UR_Stream_Thread));
        robot_thread.IsBackground = true;
        robot_thread.Start();
        // Thread is active
        UR_Stream_Data.is_alive = true;
    }
    public void Stop()
    {
        // Stop and exit thread
        exit_thread = true;
        if (robot_thread.IsAlive == true)
        {
            Thread.Sleep(100);
            UR_Stream_Data.is_alive = false;
        }
    }
    public void Destroy()
    {
        if (tcp_client.Connected == true)
        {
            // Disconnect communication
            network_stream.Dispose();
            tcp_client.Close();
        }
        Thread.Sleep(100);
    }
}
class UR_Control
{
    // Initialization of Class variables
    //  Thread
    private Thread robot_thread = null;
    private bool exit_thread = false;
    //  TCP/IP Communication
    private TcpClient tcp_client = new TcpClient();
    private NetworkStream network_stream = null;
    public void UR_Control_Thread()
    {
        try
        {
            if (tcp_client.Connected != true)
            {
                // Connect to controller -> if the controller is disconnected
```

```
                    tcp_client.Connect(UR_Control_Data.ip_address,
UR_Control_Data.port_number);
                }
                // Initialization TCP/IP Communication (Stream)
                network_stream = tcp_client.GetStream();
                // Initialization timer
                var t = new Stopwatch();
                while (exit_thread == false)
                {
                    // t_{0}: Timer start.
                    t.Start();
                    // Note:
                    //  For more information about commands, see the URScript Programming
Language document
                    if (UR_Control_Data.joystick_button_pressed == true)
                    {
                        // Send command (byte) -> speed control of the robot (X,Y,Z and
EA{RX, RY, RZ})
                        network_stream.Write(UR_Control_Data.command, 0,
UR_Control_Data.command.Length);
                    }
                    // t_{1}: Timer stop.
                    t.Stop();
                    // Recalculate the time: t = t_{1} - t_{0} -> Elapsed Time in
milliseconds
                    if (t.ElapsedMilliseconds < UR_Stream_Data.time_step)
                    {
                        Thread.Sleep(UR_Stream_Data.time_step -
(int)t.ElapsedMilliseconds);
                    }
                    // Reset (Restart) timer.
                    t.Restart();
                }
            }
            catch (SocketException e)
            {
                Debug.Log("Socket Exception:" + e);
            }
        }
        public void Start()
        {
            // Start thread
            exit_thread = false;
            // Start a thread and listen to incoming messages
            robot_thread = new Thread(new ThreadStart(UR_Control_Thread));
            robot_thread.IsBackground = true;
            robot_thread.Start();
            // Thread is active
            UR_Control_Data.is_alive = true;
        }
        public void Stop()
        {
            // Stop and exit thread
            exit_thread = true;
            if (robot_thread.IsAlive == true)
            {
                // Disconnect communication
                Thread.Sleep(100);
                UR_Control_Data.is_alive = false;
```

```
            }
        }
        public void Destroy()
        {
            if (tcp_client.Connected == true)
            {
                // Disconnect communication
                network_stream.Dispose();
                tcp_client.Close();
            }
            Thread.Sleep(100);
        }
    }
}
```

B3. Interfaces

B3.1. Main Control (UI)

```
// System
using System;
using System.Text;
// Unity
using UnityEngine;
using UnityEngine.UI;
// TM
using TMPro;
using UnityEngine.SceneManagement;
public class main_ui_control : MonoBehaviour
{
    // -------------------- GameObject -------------------- //

    // -------------------- Image -------------------- //
   // public Image connection_panel_img, diagnostic_panel_img, joystick_panel_img;
    //public Image connection_info_img;
    // -------------------- TMP_InputField -------------------- //
    public TextMeshProUGUI ip_address_txt;
    // -------------------- Float -------------------- //
   // private float ex_param = 100f;
    // -------------------- TextMeshProUGUI -------------------- //
    public TextMeshProUGUI position_x_txt, position_y_txt, position_z_txt;
    public TextMeshProUGUI position_rx_txt, position_ry_txt, position_rz_txt;
    public TextMeshProUGUI position_j1_txt, position_j2_txt, position_j3_txt;
    public TextMeshProUGUI position_j4_txt, position_j5_txt, position_j6_txt;
    public Text j1, j2, j3, j4, j5, j6;
    public TextMeshProUGUI warningTxt;
    public static GameObject warningPopUp;
    public static GameObject controlModeCanvas;
    public static GameObject connectionModeCanvas;
    public Toggle joystickToggle;
    public Toggle jointsControlToggle;

    //public TextMeshProUGUI connectionInfo_txt;
    // -------------------- UTF8Encoding -------------------- //
    private UTF8Encoding utf8 = new UTF8Encoding();

    //
```

```
// ------------------------------------------------------------------------------------------
// ------------------------------ //
    // ------------------------------------------------ INITIALIZATION {START}
// ------------------------------------------------ //
    //
// ------------------------------------------------------------------------------------------
// ------------------------------ //
    void Start()
    {
        warningPopUp = GameObject.Find("WarningPopUp");
        controlModeCanvas = GameObject.Find("ControlMode");
        connectionModeCanvas = GameObject.Find("ConnectionMode");
        // Position {Cartesian} -> X..Z
        position_x_txt.text = "0.00";
        position_y_txt.text = "0.00";
        position_z_txt.text = "0.00";
        // Position {Rotation} -> EulerAngles(RX..RZ)
        position_rx_txt.text = "0.00";
        position_ry_txt.text = "0.00";
        position_rz_txt.text = "0.00";
        // Position Joint -> 1 - 6
        position_j1_txt.text = "0.00";
        position_j2_txt.text = "0.00";
        position_j3_txt.text = "0.00";
        position_j4_txt.text = "0.00";
        position_j5_txt.text = "0.00";
        position_j6_txt.text = "0.00";
        // Robot IP Address
        ip_address_txt.text = ur_data_processing.UR_Control_Data.ip_address;
        // Auxiliary first command -> Write initialization position/rotation with
acceleration/time to the robot controller
        // command (string value)
        ur_data_processing.UR_Control_Data.aux_command_str =
"speedl([0.0,0.0,0.0,0.0,0.0,0.0], a = 0.15, t = 0.03)" + "\n";
        // get bytes from string command
        ur_data_processing.UR_Control_Data.command =
utf8.GetBytes(ur_data_processing.UR_Control_Data.aux_command_str);
        QualitySettings.vSyncCount = 0;
        Application.targetFrameRate = 72;
        joystickToggle.isOn = false;
        jointsControlToggle.isOn = false;
        controlModeCanvas.SetActive(false);
        warningPopUp.SetActive(false);
        ur_data_processing.GlobalVariables_Main_Control.connect    = false;
        ur_data_processing.GlobalVariables_Main_Control.disconnect = true;
    }
    //
// ------------------------------------------------------------------------------------------
// ------------------------------ //
    // ------------------------------------------------ MAIN FUNCTION {Cyclic}
// ------------------------------------------------ //
    //
// ------------------------------------------------------------------------------------------
// ------------------------------ //
    void FixedUpdate()
    {
         ip_address_txt.text = ur_data_processing.UR_Control_Data.ip_address;
        // ------------------------ Connection Information ------------------------//
        // If the button (connect/disconnect) is pressed, change the color and text
```

```
        if (ur_data_processing.UR_Stream_Data.SafetyStatus == 1 &&
ur_data_processing.GlobalVariables_Main_Control.connect )
        {
            // green color
            ip_address_txt.color = new Color32(0, 255, 0, 255);
            warningPopUp.SetActive(false);
            controlModeCanvas.SetActive(true);
        }
        else if(ur_data_processing.UR_Stream_Data.SafetyStatus != 1 &&
ur_data_processing.GlobalVariables_Main_Control.connect)
        {
            // red color
            ip_address_txt.color = new Color32(255, 0, 48, 255);
            warningPopUp.SetActive(true);
            controlModeCanvas.SetActive(false);
            warningTxt.text = "Parada de protección \n Consulta la flexpendant para más
información";
            ur_data_processing.UR_Control_Data.button_pressed[0] = false;
        }
        // ------------------------ Cyclic read parameters {diagnostic panel}
------------------------ //
        // Position {Cartesian} -> X..Z
        position_x_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.C_Position[0] * (1000f),
2)).ToString();
        position_y_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.C_Position[1] * (1000f),
2)).ToString();
        position_z_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.C_Position[2] * (1000f),
2)).ToString();
        // Position {Rotation} -> EulerAngles(RX..RZ)
        position_rx_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.C_Orientation[0] * (180 / Math.PI),
2)).ToString();
        position_ry_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.C_Orientation[1] * (180 / Math.PI),
2)).ToString();
        position_rz_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.C_Orientation[2] * (180 / Math.PI),
2)).ToString();
        // Position Joint -> 1 - 6
        position_j1_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.J_Orientation[0] * (180 / Math.PI),
2)).ToString();
        position_j2_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.J_Orientation[1] * (180 / Math.PI),
2)).ToString();
        position_j3_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.J_Orientation[2] * (180 / Math.PI),
2)).ToString();
        position_j4_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.J_Orientation[3] * (180 / Math.PI),
2)).ToString();
        position_j5_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.J_Orientation[4] * (180 / Math.PI),
2)).ToString();
        position_j6_txt.text =
((float)Math.Round(ur_data_processing.UR_Stream_Data.J_Orientation[5] * (180 / Math.PI),
```

```
2)).ToString();
        j1.text = position_j1_txt.text;
        j2.text = position_j2_txt.text;
        j3.text = position_j3_txt.text;
        j4.text = position_j4_txt.text;
        j5.text = position_j5_txt.text;
        j6.text = position_j6_txt.text;
    }
    //
-----------------------------------------------------------------------------------------------
-----------------------------//
    // ------------------------------------------------------- FUNCTIONS
-----------------------------------------------------//
    //
-----------------------------------------------------------------------------------------------
-----------------------------//
    // -------------------- Destroy Blocks -------------------- //
    void OnApplicationQuit()
    {
        // Destroy all
        Destroy(this);
    }
}
```

B3.2. Orientación interfaz

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class LookAtPlayer : MonoBehaviour
{
    public Transform target;
    void Update()
    {
        Vector3 targetPosition = new Vector3(-target.transform.position.x,
transform.position.y, -target.transform.position.z);
        transform.LookAt(targetPosition);
    }
}
```

B3.3. Botones

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
public class NumberButtons : MonoBehaviour, IPointerClickHandler
{
    public string value = "0";
    public Text ipText;
    public  Dropdown polyscopeVersionSelected;
    public static string actual_IP;
    public Toggle ConnectionPanelToggle;
    public void OnPointerClick(PointerEventData eventData)
    {
```

```
        if (value == "c")
        {
            actual_IP = "";
            ipText.text = actual_IP;
        }
        else if (value == "<")
        {
            actual_IP = actual_IP.Remove(actual_IP.Length-1);
        }
        else if (value == ".")
        {
            actual_IP = actual_IP + value;
        }
        else if (value == "connect")
        {
                ConnectionPanelToggle.GetComponent<Toggle>().isOn = false;
                ur_data_processing.UR_Stream_Data.ip_address = actual_IP;
                ur_data_processing.UR_Control_Data.ip_address = actual_IP;
                ur_data_processing.GlobalVariables_Main_Control.connect    = true;
                ur_data_processing.GlobalVariables_Main_Control.disconnect = false;
                switch(polyscopeVersionSelected.value)
                {
                    case 0:
                        ur_data_processing.GlobalVariables_Main_Control.polyscopeVersion =
ur_data_processing.versions.V3_14;
                        break;
                    case 1:
                        ur_data_processing.GlobalVariables_Main_Control.polyscopeVersion =
ur_data_processing.versions.V3_15;
                        break;
                }
                main_ui_control.controlModeCanvas.SetActive(true);
                main_ui_control.connectionModeCanvas.SetActive(false);
                ur_data_processing.UR_Stream_Data.SafetyStatus = 1;

        }
        else if (value == "real")
        {
            actual_IP = "192.168.1.49";
            polyscopeVersionSelected.value = 0;
        }
        else if (value == "simu")
        {
            actual_IP = "192.168.2.132";
            polyscopeVersionSelected.value = 1;
        }
        else
        {
            actual_IP = actual_IP + value;
        }
        ipText.text = actual_IP;

    }


}
```

B3.4. Control articular individual

```
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.EventSystems;
using System.Text;
public class JointIndividualControl : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    public enum articulaciones{
        j1,j2,j3,j4,j5,j6
    };
    public enum Signo{
        plus, minus
    };
    public articulaciones join = articulaciones.j1;
    public Signo signo = Signo.plus;
    public Text valueLabel;
    public Slider factor;
    public double[] joints ={0,0,0,0,0,0};
    // Start is called before the first frame update
    private UTF8Encoding utf8 = new UTF8Encoding();
    public void OnPointerDown(PointerEventData eventData)
    {
        for(int i=0; i < joints.Length; i++)
        {
            joints[i] = 0;
        }

        // Evitar paro de seguridad por salirse de los limites articulares
        if(signo == Signo.plus)
        {
            joints[(int)join] += factor.value;
        }
        else
        {
            joints[(int)join] -= factor.value;
        }

        ur_data_processing.UR_Control_Data.aux_command_str = "speedj([" +
joints[0].ToString(CultureInfo.InvariantCulture) +"," +
joints[1].ToString(CultureInfo.InvariantCulture) + "," +
joints[2].ToString(CultureInfo.InvariantCulture) + "," +
joints[3].ToString(CultureInfo.InvariantCulture) + "," +
joints[4].ToString(CultureInfo.InvariantCulture) + "," +
joints[5].ToString(CultureInfo.InvariantCulture)
                                                        + "], a = 0.5, t = " +
Time.deltaTime.ToString(CultureInfo.InvariantCulture) + ")" + "\n";

        ur_data_processing.UR_Control_Data.command =
utf8.GetBytes(ur_data_processing.UR_Control_Data.aux_command_str);
        // confirmation variable -> is pressed
        ur_data_processing.UR_Control_Data.button_pressed[0] = true;

    }
    public void OnPointerUp(PointerEventData eventData)
```

```
    {
        // confirmation variable -> is un-pressed
        ur_data_processing.UR_Control_Data.button_pressed[0] = false;
    }
}
```

B3.5. Chequear botones

```
// -------------------- System ------------------- //
using System.Text;
// -------------------- Unity ------------------- //
using UnityEngine.EventSystems;
using UnityEngine;
using UnityEngine.UI;
using System;
public class button_check: MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    // -------------------- String ------------------- //
    public string acceleration = "1.0";
    public string time = "0.05";
    public string[] speed_param      = new string[6] {"0.0", "0.0", "0.0",
"0.0","0.0","0.0"};
    private string[] speed_param_null = new string[6] { "0.0", "0.0", "0.0", "0.0", "0.0",
"0.0" };
    // -------------------- Int ------------------- //
    public int index;
    // -------------------- UTF8Encoding ------------------- //
    private UTF8Encoding utf8 = new UTF8Encoding();
    // -------------------- Button -> Pressed ------------------- //
    public void OnPointerDown(PointerEventData eventData)
    {
        // create auxiliary command string for speed control UR robot
        ur_data_processing.UR_Control_Data.aux_command_str = "speedl([" + speed_param[0]
+","+  speed_param[1] + "," + speed_param[2]
                                                                + "," + speed_param[3] +
"," + speed_param[4] + "," + speed_param[5] + "], a =" + acceleration + ", t =" + time +
")" + "\n";
        // get bytes from command string
        ur_data_processing.UR_Control_Data.command =
utf8.GetBytes(ur_data_processing.UR_Control_Data.aux_command_str);
        // confirmation variable -> is pressed
        ur_data_processing.UR_Control_Data.button_pressed[index] = true;
    }
    // -------------------- Button -> Un-Pressed ------------------- //
    public void OnPointerUp(PointerEventData eventData)
    {
        // confirmation variable -> is un-pressed
        ur_data_processing.UR_Control_Data.button_pressed[index] = false;
    }
}
```

B3.6. Toggle control

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class ToogleControl : MonoBehaviour
{
    public Sprite spriteActive;
    public Sprite spriteNotActive;
    public GameObject imageBackground;
    public Toggle joystickToggle, jointsToggle;
    public void ImageChange()
    {
      if(this.GetComponent<Toggle>().isOn)
      {
          imageBackground.GetComponent<Image>().sprite = spriteActive;
      }
      else
      {
          imageBackground.GetComponent<Image>().sprite = spriteNotActive;
      }
    }
    public void JointsValueToggleChanged()
    {
        if(jointsToggle.isOn)
        {

            joystickToggle.isOn = false;
        }
    }

    public void JoystickValueToggleChanged()
    {
        if(joystickToggle.isOn)
        {
            jointsToggle.isOn = false;
        }
    }
}
```

B4. Movimiento jugador

B4.1. Joystick (I)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using OVR;
using UnityEngine.XR;
public class JoystickLocomotion : MonoBehaviour
{
    public GameObject player;
    public float speed;

   // Update is called once per frame
```

```
    void Update()
    {
        var JoystickAxis = OVRInput.Get(OVRInput.RawAxis2D.LThumbstick,
OVRInput.Controller.LTouch);
        //float pos = player.transform.position.y;
        float fixedY = player.transform.position.y;
        //player.position += (transform.right * JoystickAxis.x + transform.forward *
JoystickAxis.y)*Time.deltaTime*speed;
        player.transform.position += (transform.right * -JoystickAxis.x + transform.forward
* -JoystickAxis.y)*Time.deltaTime*speed;
        //player.transform.Translate()
        player.transform.position = new Vector3(player.transform.position.x, fixedY,
player.transform.position.z);
    }
}
```

B4.2. Rotación joystick

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using OVR;
using UnityEngine.XR;
public class JoystickRotation : MonoBehaviour
{
    public GameObject player;
    public float speed;
    // Update is called once per frame
    void Update()
    {
        var JoystickAxis = OVRInput.Get(OVRInput.RawAxis2D.RThumbstick,
OVRInput.Controller.RTouch);
        player.transform.Rotate(Vector3.up * JoystickAxis.x * Time.deltaTime * speed,
Space.Self );

    }
}
```

B5. Control manual del robot

```
using System.Collections;
using System.Globalization;
using System.Collections.Generic;
using UnityEngine;
using System.Text;
using System;
public class ManualModeControl : MonoBehaviour
{
    private GameObject TCPinHand;
    public GameObject TCPinHandModel;
    private GameObject robotTCP;
    public GameObject rightController;
    private UTF8Encoding utf8 = new UTF8Encoding();
    // Start is called before the first frame update
    void Start()
    {
```

```
        // Find TCP of the robot
        robotTCP = GameObject.Find("UR3_Robot/Link1/Link2/Link3/Link4/Link5/Link6");
        // Create a copy of the robot TCP in the right hand
        TCPinHand = GameObject.Instantiate(TCPinHandModel);
        // Set the model as child of the right controller at (0, 0, 0) position
        TCPinHand.transform.position = new Vector3(0.0f, 0.0f, 0.0f);
        TCPinHand.transform.parent = transform;
        TCPinHand.transform.localPosition = new Vector3(0.0f, 0.0f, 0.0f);
        TCPinHand.SetActive(false);
    }
    // Update is called once per frame
    void Update()
    {

        var trigPress = OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger,
OVRInput.Controller.RTouch);
        var trigRelease = OVRInput.GetUp(OVRInput.Button.PrimaryIndexTrigger,
OVRInput.Controller.RTouch);

        if (trigPress)
        {
            TCPinHand.SetActive(true);
            TCPinHand.transform.rotation = robotTCP.transform.rotation;
            rightController.SetActive(false);
        }
        if (trigRelease)
        {

            TCPinHand.SetActive(false);
            rightController.SetActive(true);
            ur_data_processing.UR_Control_Data.button_pressed[0] = false;

        }
        var trigActive = OVRInput.Get(OVRInput.Button.PrimaryIndexTrigger,
OVRInput.Controller.RTouch);

        if(trigActive)
        {

            // Get controller velocity, reference controller axis
            Vector3 controllerVelocity =
OVRInput.GetLocalControllerVelocity(OVRInput.Controller.RTouch);
            // rotate velocity vector around Y World Axis (the player can move around the
robot)
            controllerVelocity =
Quaternion.AngleAxis(rightController.transform.rotation.eulerAngles[1], Vector3.up) *
controllerVelocity;
            /// robot base axis (x, y, z), unity world axis (x, z, y)
            Vector3 aux = controllerVelocity;
            controllerVelocity[2] = aux[1];
            controllerVelocity[1] = aux[2];
            // Get controller angular velocity, reference controller axis
            Vector3 controllerAngularVelocity =
OVRInput.GetLocalControllerAngularVelocity(OVRInput.Controller.RTouch);
            // rotate angular velocity vector around Y World Axis (the player can move
around the robot)
            controllerAngularVelocity =
Quaternion.AngleAxis(rightController.transform.rotation.eulerAngles[1], Vector3.up) *
controllerAngularVelocity;
```

```
            /// robot base axis (x, y, z), unity world axis (x, z, y)
            aux = controllerAngularVelocity;
            controllerAngularVelocity[1] = aux[2];
            controllerAngularVelocity[2] = aux[1];
            // Send speedl command
            ur_data_processing.UR_Control_Data.aux_command_str = "speedl(["
+controllerVelocity[0].ToString(CultureInfo.InvariantCulture) +","
                                                                  +
controllerVelocity[1].ToString(CultureInfo.InvariantCulture)
                                                                  + ","
+controllerVelocity[2].ToString(CultureInfo.InvariantCulture)
                                                                  + "," +
controllerAngularVelocity[0].ToString(CultureInfo.InvariantCulture)
                                                                  + "," +
controllerAngularVelocity[1].ToString(CultureInfo.InvariantCulture)
                                                                  + "," +
controllerAngularVelocity[2].ToString(CultureInfo.InvariantCulture)
                                                                  + "], a =" + "1.0" + ",
t =" + Time.deltaTime.ToString(CultureInfo.InvariantCulture) + ")" + "\n";
            ur_data_processing.UR_Control_Data.command =
utf8.GetBytes(ur_data_processing.UR_Control_Data.aux_command_str);
            // confirmation variable -> is pressed
            ur_data_processing.UR_Control_Data.button_pressed[0] = true;
        }
    }
}
```

B6. Visibilidad espacio trabajo robot

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class WorkSpaceVisbility : MonoBehaviour
{
    // Start is called before the first frame update
    public GameObject WorkSpaceModel;
    public GameObject EndEffector;
    public Material NoVisible;
    public Material Visible;
    void OnTriggerExit(Collider other) {
        if (other.gameObject.CompareTag(EndEffector.tag))
        {
            WorkSpaceModel.GetComponent<MeshRenderer>().material = Visible;
        }
    }
    void OnTriggerStay(Collider other) {

        if (other.gameObject.CompareTag(EndEffector.tag))
        {
            WorkSpaceModel.GetComponent<MeshRenderer>().material = NoVisible;
        }
    }
}
```