



## **GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE**

Escuela de Ingeniería de Fuenlabrada

Curso académico 2023-2024

### **Trabajo Fin de Grado**

**TELEOPERACIÓN A DISTANCIA DE BRAZO ROBÓTICO  
MEDIANTE INTERFAZ DE REALIDAD VIRTUAL**

**Autor:** Iván Porras Estébanez  
**Tutor:** Jesús María González Barahona



©2024 Iván Porras Estébanez

Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-SA International License (Creative Commons Attribution-ShareAlike 4.0)<sup>1</sup>. Usted es libre de (a) *compartir*: copiar y redistribuir el material en cualquier medio o formato; y (b) *adaptar*: remezclar, transformar y crear a partir del material. El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia:

- *Atribución.* Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciatante.
- *Compartir igual.* Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

*Documento de Iván Porras Estébanez.*

---

<sup>1</sup>Licencia CC BY-SA:<https://creativecommons.org/licenses/by-sa/4.0/deed.en>

# Agradecimientos

---

A mis padres por haber sido uno de mis grandes apoyos a lo largo de toda la carrera y enseñarme a ser constante, sin vosotros no hubiese sido posible.

A mis abuelos por animarme y siempre estar ahí. Habéis sido un gran pilar y ejemplo para mí.

A mis amigos por toda la felicidad que siempre me habéis dado. En especial a mis cuatro compañeros de piso, gracias por haber escuchado siempre mis quejas y por todos los momentos buenos.

A mi pareja, Jimena, por haber estado siempre a mi lado en esta recta final, dándome apoyo y ayudándome a seguir.

Por último dar las gracias a Jesús, mi tutor de TFG, por su gran dedicación y ayuda en este proyecto.

Madrid, 09 de Julio de 2024

*Iván Porras Estébanez*

# Resumen

---

El presente Trabajo de Fin de Grado describe el diseño, desarrollo, implementación y pruebas de un sistema de teleoperación para un brazo robótico controlado a través de un entorno de Realidad Virtual (VR). Su objetivo principal es permitir al operario visualizar la mayor cantidad de información posible en la escena virtual y dotarlo de un control preciso y natural. Esta forma de control busca conseguir movimientos similares a los humanos para así facilitar la adaptación de los usuarios.

Como brazo robótico se ha utilizado el robot colaborativo UR3 de Universal Robots al que se le ha configurado un sistema de red inalámbrica para poder ser controlado a distancia aprovechando la infraestructura de Internet. Por otro lado, para la creación del entorno VR se ha utilizado el motor de videojuegos Godot junto con el paquete *Open XR Tools*, que permite utilizar casi cualquiera de los visores de realidad virtual actuales. Este entorno contiene un modelo 3D del robot, que se actualiza con la situación del brazo robótico real y una pantalla de proyección sobre la que se visualizan las imágenes capturadas por una cámara IP situada en el área del robot real.

Este sistema ha sido evaluado en términos de rendimiento y experiencia de usuario. Los resultados de estas pruebas garantizan la viabilidad y el cumplimiento de los objetivos previamente mencionados. Entre las observaciones destacan la rápida adaptación a los movimientos y la elevada tasa de refresco obtenida, que ronda los 70Hz.

**Palabras clave:** robótica, realidad virtual, teleoperación, comunicación red, interfaz de usuario, Godot, Universal Robots

# Summary

---

This Final Degree Project describes the design, development, implementation, and testing of a teleoperation system for a robotic arm controlled through a Virtual Reality (VR) environment. Its main objective is to allow the operator to view the greatest amount of information possible in the virtual scene and provide precise and natural control. This form of control seeks to achieve human-like movements to facilitate user adaptation.

The UR3 collaborative robot from Universal Robots has been used as the robotic arm, configured with a wireless network system to enable remote control via the Internet infrastructure. To create the VR environment, the Godot video game engine was used along with the Open XR Tools package, which allows compatibility with most current virtual reality viewers. This environment includes a 3D model of the robot, which is updated in real-time to reflect the position of the actual robotic arm, and a projection screen that displays images captured by an IP camera located near the real robot.

This system has been evaluated in terms of performance and user experience. The results of these tests guarantee the viability and fulfillment of the aforementioned objectives. Notably, the system demonstrated rapid adaptation to movements and a high refresh rate of approximately 70Hz.

**Keywords:** robotics, virtual reality, teleoperation, network communication, user interface, Godot, Universal Robots

# Acrónimos

---

**UR** *Universal Robots*

**VR** *Virtual Reality*

**VNC** *Virtual Network Computing*

**HMI** *Human-Machine Interface*

**HRI** *Human-Robot Interaction*

**LIDAR** *Light Detection and Ranging*

**CPU** *Central Processing Unit*

**FPGA** *Field-Programmable Gate Arrays*

**PCB** *Printed Circuit Board*

**DOF** *Degrees Of Freedom*

**SCARA** *Selective Compliant Assembly Robot Arm*

**API** *Application Programming Interface*

**HMD** *Head-Mounted Display*

**TCP** *Tool Center Point*

**TCP/IP** *Transmission Control Protocol / Internet Protocol*

**HTTP** *HyperText Transfer Protocol*

# Índice general

---

<b>1. Introducción</b>	<b>1</b>
1.1. Contexto . . . . .	1
1.1.1. Robótica . . . . .	1
1.1.2. La Realidad Virtual . . . . .	5
1.1.3. Comunicación red . . . . .	9
1.2. Motivación . . . . .	10
1.3. Objetivos . . . . .	11
1.3.1. Objetivo principal . . . . .	11
1.3.2. Objetivos específicos . . . . .	11
1.3.3. Objetivos relacionados . . . . .	12
1.4. Estructura de la memoria . . . . .	13
1.5. Disponibilidad de código . . . . .	13
<b>2. Tecnologías utilizadas y Trabajos relacionados</b>	<b>15</b>
2.1. Trabajos relacionados . . . . .	15
2.2. Tecnologías software . . . . .	17
2.2.1. Lenguajes de programación . . . . .	17
2.2.2. Plataforma creación de escena VR - Godot . . . . .	18
2.2.3. Software brazo robótico - Universal Robots . . . . .	20
2.3. Tecnologías de comunicación . . . . .	22
2.3.1. Sockets . . . . .	22
2.3.2. TCP . . . . .	23
2.3.3. HTTP . . . . .	23
2.3.4. RTDE . . . . .	24
2.3.5. Dashboard Communication . . . . .	25
2.4. Tecnologías hardware . . . . .	26
2.4.1. Hardware sistema robótico - Universal Robots . . . . .	26
2.4.2. Robotiq Gripper 2F-85 . . . . .	29

2.4.3. Meta Quest 2 . . . . .	30
2.4.4. Router TP-Link Archer AX23 . . . . .	31
2.4.5. Ordenador HP VR Backpack G2 . . . . .	32
2.4.6. Raspberry Pi 4 y RaspiCam . . . . .	33
<b>3. Desarrollo del proyecto . . . . .</b>	<b>35</b>
3.1. Etapa 0 . . . . .	35
3.1.1. Objetivos de diseño . . . . .	35
3.1.2. Diseño e implementación . . . . .	36
3.1.3. Resultados . . . . .	38
3.2. Etapa 1 . . . . .	38
3.2.1. Objetivos de diseño . . . . .	38
3.2.2. Diseño e implementación . . . . .	39
3.2.3. Resultados . . . . .	41
3.3. Etapa 2 . . . . .	42
3.3.1. Objetivos de diseño . . . . .	42
3.3.2. Diseño e implementación . . . . .	42
3.3.3. Resultados . . . . .	43
3.4. Etapa 3 . . . . .	43
3.4.1. Objetivos de diseño . . . . .	43
3.4.2. Diseño e implementación . . . . .	44
3.4.3. Resultados . . . . .	47
3.5. Etapa 4 . . . . .	47
3.5.1. Objetivos de diseño . . . . .	47
3.5.2. Diseño e implementación . . . . .	48
3.5.3. Resultados . . . . .	48
3.6. Etapa 5 . . . . .	49
3.6.1. Objetivos de diseño . . . . .	49
3.6.2. Diseño e implementación . . . . .	49
3.6.3. Resultados . . . . .	52
<b>4. Resultados . . . . .</b>	<b>53</b>
4.1. Descripción funcional . . . . .	53
4.1.1. Inicialización sistema robótico . . . . .	53
4.1.2. Inicialización entorno VR . . . . .	55
4.1.3. Descripción de la escena virtual . . . . .	57
4.1.4. Uso y funcionalidades . . . . .	59

4.2. Descripción técnica . . . . .	61
4.2.1. Entorno Real . . . . .	62
4.2.2. Entorno Simulado . . . . .	67
<b>5. Experimentación y Validación</b>	<b>78</b>
5.1. Experimentación técnica . . . . .	78
5.2. Experimentación con usuarios . . . . .	81
<b>6. Conclusiones</b>	<b>83</b>
6.1. Objetivos cumplidos . . . . .	83
6.1.1. Objetivo principal . . . . .	83
6.1.2. Objetivos específicos y relacionados . . . . .	84
6.2. Aplicación de lo aprendido . . . . .	85
6.2.1. A través de la titulación . . . . .	86
6.2.2. De forma autodidacta . . . . .	86
6.3. Presupuesto y Tiempo Invertido . . . . .	87
6.4. Trabajos Futuros . . . . .	88
<b>Bibliografía</b>	<b>90</b>

# Índice de figuras

---

1.1.	Sensor láser LIDAR SICK TIM781 . . . . .	2
1.2.	Brazo robótico Unimate . . . . .	3
1.3.	Robot SCARA de la compañía Yamaha . . . . .	4
1.4.	Area de trabajo del ABB IRB 140 . . . . .	4
1.5.	Proceso de renderizado en imagen con OpenGL . . . . .	6
1.6.	Visores de VR . . . . .	8
2.1.	Esquema sistema de teleoperación en tiempo real QNX . . . . .	15
2.2.	Sistemas quirúrgico Da Vinci . . . . .	16
2.3.	Robot Pepper de Softbank . . . . .	16
2.4.	Ejemplo de conjunto de nodos en Godot . . . . .	18
2.5.	URScript cargado en Teach Pendant de UR3 . . . . .	21
2.6.	Vista del simulador URsim . . . . .	22
2.7.	Esquema comunicación Sockets . . . . .	23
2.8.	Esquema de comunicación UR . . . . .	26
2.9.	Modelos de brazos robóticos UR CB3-series . . . . .	27
2.10.	Caja controladora Universal Robots . . . . .	28
2.11.	Brazo robótico UR3 . . . . .	29
2.12.	Gripper 2F-85 de la empresa Robotiq . . . . .	30
2.13.	Router TP-Link Archer AX23 . . . . .	32
2.14.	Ordenador HP VR Backpack G2 . . . . .	33
2.15.	Raspberry Pi 4 y Raspicam . . . . .	34
3.1.	Pantalla del curso de creación de programas en URacademy . . . . .	36
3.2.	Mando Logitech F710 . . . . .	38
3.3.	Escena de pruebas de Godot XR tools . . . . .	40
3.4.	Red WIFI inalámbrica del sistema robótico . . . . .	40
3.5.	IPs estáticas dentro de red UR3 . . . . .	40
3.6.	Esquema de funcionamiento del protocolo RTDE . . . . .	41

3.7. Configuración planos de seguridad UR . . . . .	44
3.8. Cono de visualización de elemento terminal del UR3 . . . . .	45
3.9. Imagen cámara IP proyectada sobre entorno VR . . . . .	47
3.10. Mapeo de botones en OpenXR Action Map . . . . .	48
3.11. Comparación área de trabajo entorno VR y entorno real . . . . .	50
3.12. Mesa de operaciones en la aplicación de Realidad Virtual . . . . .	50
3.13. Modelo de brazo robótico UR3 en la aplicación de Realidad Virtual . .	51
4.1. Etapas de lanzamiento sistema robótico desde Teach Pendant . . . . .	55
4.2. Pantalla de configuración de Meta Quest 2 . . . . .	56
4.3. Ejecutable de la aplicación (.exe) . . . . .	56
4.4. Robot en posición de inicio, vista desde entorno virtual . . . . .	57
4.5. Elementos pasivos del entorno VR . . . . .	58
4.6. Elementos activos del entorno VR . . . . .	59
4.7. Comparación entre entorno robot real y escena VR . . . . .	59
4.8. Configuración botones mando Oculus Quest 2 . . . . .	60
4.9. Esquema sistema comunicaciones completo simplificado . . . . .	62
4.10. Formato de rotaciones RPY vs Vector de Rotaciones . . . . .	63
4.11. Esquema de funcionamiento de <i>script</i> de control del sistema robótico .	64
4.12. Conexión Raspberry Pi 4 y Raspicam . . . . .	66
4.13. Tipos de nodos en aplicación Godot . . . . .	68
4.14. Nodos estáticos del árbol de nodos de la aplicación VR . . . . .	69
4.15. Nodos de visualización dinámica del árbol de nodos de la aplicación VR	72
4.16. Nodo de interacción dinámica del árbol de nodos de la aplicación VR .	72
4.17. Nodo de actuación urbot . . . . .	73
4.18. Comparación ejes de coordenadas Godot y UR3 . . . . .	75
4.19. Flujo de control robot nodo urbot . . . . .	76
5.1. Velocidades de funcionamiento de sistema . . . . .	80

# Listado de fragmentos de código

---

2.1. Fichero de configuración protocolo RTDE . . . . .	25
3.1. Formación mensaje salida aplicación VR para movimiento . . . . .	43
3.2. Solicitud HTTP en GdScript para leer la información de una cámara IP	46
4.1. Función recepción imagen por protocolo HTTP y carga en Sprite3D . .	70
4.2. Función de recepción de valores de robot real y asignación modelo 3D .	71

# Índice de cuadros

---

2.1. Especificaciones UR3 . . . . .	29
2.2. Especificaciones Gripper 2F-85 . . . . .	30
2.3. Especificaciones visor Meta Quest 2 . . . . .	30
2.4. Especificaciones router TP-Link Archer AX23 . . . . .	31
4.1. Registros escritura utilizados en comunicación RTDE . . . . .	63
4.2. IPs estáticas configuradas . . . . .	67
6.1. Presupuesto de proyecto . . . . .	87
6.2. Desglose tiempo dedicación proyecto . . . . .	88

---

# **Capítulo 1**

## **Introducción**

---

En este primer capítulo se explicará el contexto de las tecnologías utilizadas para el desarrollo de este proyecto, la motivación detrás de esta idea y los objetivos planteados para llevarla a cabo.

### **1.1. Contexto**

La robótica y la Realidad Virtual son sectores que han experimentado grandes avances y adaptación en los últimos años. Estos han conseguido dejar de ser simples prototipos para abordar nuestro día a día y dar solución a muchos de los problemas a los que comúnmente nos enfrentamos. Habitualmente estos campos están desligados entre sí, desaprovechando el potencial que puede obtenerse de la fusión de dos tecnologías, que, aunque emergentes, continuamente nos demuestran que serán un pilar fundamental en el futuro cercano. Estos avances hacen que la forma de trabajar con ellas cambie constantemente de paradigma, pero manteniendo la base común que las caracteriza. Esta sección trata de poner en conocimiento del lector estas bases.

#### **1.1.1. Robótica**

La robótica es la disciplina encargada del estudio, diseño, programación y aplicación de sistemas robóticos. Para ello integra varias áreas de conocimiento como la electrónica, mecánica, ingeniería de software y de formas más indirecta otras como la biomedicina o la ingeniería de control. Se denomina como robot a un sistema capaz de interaccionar con su entorno de manera inteligente o controlada.

Existen tres elementos principales que definen un robot:

- *Sensores*: dispositivos físicos que aprovechan propiedades de los materiales y conceptos físicos para extraer información del entorno en forma de magnitudes físicas. Existe un gran abanico de opciones en función de la información que se desea conocer, desde sensores láser (LIDAR) (figura 1.1) a simples sensores de presión. Estos elementos son los encargados de dotar al robot de conocimiento del entorno.
- *Actuadores*: dispositivos físicos controlables encargados de dotar al robot de la capacidad de interacción con el entorno. En función del rendimiento y precisión requeridos pueden ser de diferentes tipos: eléctricos, neumáticos, etc.
- *Hardware de Control*: conjunto de componentes electrónicos específicamente diseñados para recibir datos provenientes de los sensores a través de sus entradas para, posteriormente, procesar esta información de manera lógica y enviar las señales de salida correspondientes a los actuadores. Es decir, estos dispositivos dotan al robot de la capacidad de interactuar de forma lógica con el entorno. En la mayoría de casos son programables como procesadores (CPU), microcontroladores o FPGAs, aunque en algunos casos muy simples se utilizan circuitos impresos (PCB).



Figura 1.1: Sensor láser LIDAR SICK TIM781

## Brazos Robóticos

Si nos remontamos a los orígenes de la robótica encontraremos que una de las primeras máquinas que puede considerarse como un robot y que tuvo una aplicación real es el brazo robótico *Unimate* (figura 1.2). Desarrollado por George Devol e instalado en una cadena de montaje de *General Motors*<sup>1</sup>. Desde ese momento este sector no ha

<sup>1</sup>General Motors: <https://www.gm.com/>

dejado de avanzar consiguiendo posicionarse como la aplicación robótica más extendida en el mundo profesional.



Figura 1.2: Brazo robótico Unimate

Los brazos robóticos, además de seguir la composición general de un robot como ya explicamos en la sección 1.1.1 siguen una composición que imita la anatomía humana. Estos están divididos en articulaciones y partes fijas que unen las articulaciones entre sí. Y que comúnmente reciben nombres como hombro, codo o antebrazo. Además, destaca también el elemento terminal, que dota al brazo de la capacidad de realizar la tarea específica para la que se ha preparado. El poder cambiar este elemento permite que un determinado modelo de estos robots tenga la habilidad de realizar operaciones muy diversas. Algunas de las tareas más comunes son soldadura, pintura, *pick and place*...

Este tipo de robots se pueden clasificar de muchas formas, pero entre sus características distintivas suelen remarcarse las siguientes:

- *Grados de libertad (DOF)*: los grados de libertad son utilizados en el mundo de la robótica para cuantificar la capacidad de movimiento de un robot en los distintos ejes. Cada grado de libertad corresponde a un movimiento de rotación o traslación que puede realizar el robot en un determinado eje. Existen multitud de brazos robóticos con distintos grados de libertad en función a la tarea para la que estén pensados. Por ejemplo destaca el robot *SCARA* (figura 1.3) muy utilizado en la industria y con normalmente solo 4 DOF. En la actualidad los brazos de 6 DOF están muy presentes debido a la gran libertad que tienen para realizar movimientos y su extensa área de trabajo.



Figura 1.3: Robot SCARA de la compañía Yamaha

- *Área de trabajo:* el área de trabajo marca el espacio de movimiento de un brazo robótico. En concreto, se trata del conjunto de puntos que el elemento terminal puede alcanzar y operar. Este espacio queda definido por el número de grados de libertad y la longitud de los segmentos que unen las articulaciones del robot. En la figura 1.4 podemos ver el área de trabajo de un robot de la compañía ABB.<sup>2</sup>

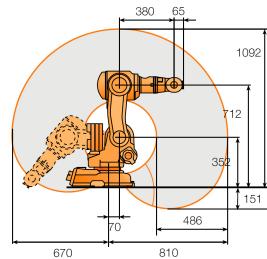


Figura 1.4: Área de trabajo del ABB IRB 140

- *Trabajo colaborativo:* la seguridad es uno de los factores principales en el campo de los robots industriales. Inicialmente, esta clase de máquinas estaban pensadas para trabajar en entornos controlados, manteniendo ciertas distancias con el resto de operarios humanos. Esto debido principalmente a estar pensadas para realizar tareas repetitivas sin necesidad de recopilar y analizar demasiada información del entorno. Sin embargo, en los últimos años han ido cada vez tomando mayor importancia los denominados *Cobots* o robots colaborativos. Este tipo de brazos robóticos están equipados con sensores avanzados que le permiten detectar cualquier contacto o fuerza ejercida sobre él y así poder certificar la seguridad de los trabajadores cercanos. Estos están especialmente pensados para realizar tareas junto a estos operarios, dotándoles de numerosas ventajas y permitiendo

---

<sup>2</sup>ABB: <https://global.abb/group/en>

una implantación más óptima y eficiente en las líneas de producción. Destaca la compañía Universal Robots<sup>3</sup>: como uno de sus principales impulsores.

## Teleoperación

La teleoperación o control remoto se puede definir como la operación controlada a distancia de una determinada máquina o sistema. Se podría considerar que el término aparece a finales de siglo XIX en el ámbito militar. Principalmente de los intentos de muchos inventores para conseguir el control remoto de distintas armas como misiles o torpedos.

La dificultad de crear sistemas robóticos con la capacidad de trabajar de forma autónoma en entornos poco estructurados o tareas con cierto grado de complejidad ha generado que esta disciplina vaya muy ligada a la robótica desde sus inicios. Actualmente, podemos ver estos en una gran parte de los robots comerciales utilizados fuera del ámbito industrial. Como por ejemplo en el control de drones o en robots de uso militar como los encargados de la desactivación de explosivos, por ejemplo el Packbot (iRobot).

### 1.1.2. La Realidad Virtual

La Realidad Virtual es una tecnología que permite la inmersión del usuario en un conjunto de escenas y objetos virtuales generados por ordenador. En algunos casos se busca una inversión total en la que se le puede dotar al usuario de interacción con los objetos del entorno simulado y una escena totalmente tridimensional. Mientras que en otros se muestran escenarios más simples como vídeos 360º. Este campo ha ido ganando popularidad en los últimos años, desarrollándose desde simples aplicaciones de ocio hasta otras profesionales como el entrenamiento militar o la visualización arquitectónica.

## Renderizado gráfico

El renderizado gráfico (generación de imagen virtual) dentro de la Realidad Virtual se puede definir como el proceso de generar las imágenes tridimensionales que se le mostrarán al usuario. Es imprescindible que este renderizado se realice con baja latencia

---

<sup>3</sup>Universal Robots: <https://www.universal-robots.com/es/>

y siguiendo los inputs recibidos por el sistema para conseguir una experiencia realmente inmersiva. Es decir, tiene que tener en cuenta cosas como la posición de la cabeza del operario, la posición dentro del entorno, la luz y otros elementos para conseguir una interacción fluida y realista con el entorno.

Este proceso es muy costoso en términos computacionales y requiere de la máxima optimización posible para conseguir sus objetivos de latencia. Para llevarlo a cabo existen varias librerías entre las que destaca OpenGL<sup>4</sup>.

OpenGL es una API utilizada para renderizar gráficos en 2D y 3D en una amplia variedad de plataformas. Funciona transformando los datos de entrada, como la geometría y los atributos de los objetos, a través de una serie de etapas de procesamiento. Estas etapas incluyen transformaciones geométricas para ajustar la posición y orientación de los objetos en el espacio, cálculos de iluminación para simular la interacción de la luz con los objetos, y procesamiento de texturas para aplicar imágenes a las superficies de los objetos.

Una vez que se han procesado todos los datos de entrada, OpenGL rasteriza la escena, convirtiendo los objetos definidos en la etapa de geometría en píxeles. Luego, aplica técnicas de mapeo de texturas, sombreado y otros efectos visuales para producir la imagen final que se muestra al usuario.

En la figura 1.5 se puede ver el proceso que sigue OpenGL para renderizar una imagen.

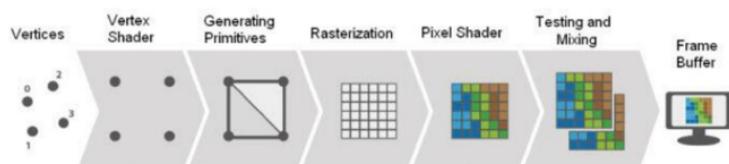


Figura 1.5: Proceso de renderizado en imagen con OpenGL

## Visores VR

Para poder simular estos entornos inmersivos necesitaremos una serie de elementos:

---

<sup>4</sup>OpenGL: <https://www.opengl.org/>

- Procesador: encargado del cómputo de los diferentes elementos que componen la escena y su transformación en píxeles (renderizado).
- Elemento de visualización: normalmente, una o varias pantallas con dos lentes que permiten una vista estereoscópica. Muestran los píxeles extraídos por el procesador que forman el punto de vista del usuario en la escena.
- Elementos de control: elementos hardware o software que permiten el control o interacción del usuario con la escena. Desde mandos a algoritmos de seguimiento o *tracking*.

Normalmente, todos estos elementos son encapsulados en los denominados Visores de Realidad Virtual. Existen visores sumamente diferentes entre sí y con distintos grados de complejidad en función a su finalidad y la época en la que fueron desarrollados. A continuación recorremos algunos ejemplos siguiendo la historia de estos dispositivos:

- *Virtual Boy*: consola de videojuegos lanzada al mercado por Nintendo en 1995, consiste en un visor con una pantalla monocromática roja y negra, en la que los juegos utilizan un efecto de paralaje para crear una ilusión de profundidad, permitiendo así mostrar gráficos estereoscópicos «3D» (figura 1.6 (a)).
- *Oculus Rift (primera generación)*: fue desarrollado en 2010 y aunque este cuenta con varias limitaciones, se puede considerar como el precursor de la Realidad Virtual moderna. Permite un campo de visión inmersivo y seguimiento de cabeza. Necesita estar conectado a un ordenador externo para funcionar (figura 1.6 (b)).
- *Google Cardboard*: carcasa de cartón con dos lentes en resina que permiten colocar un smartphone de cualquier marca en su interior y utilizar un software genérico para reproducir contenido VR de forma sencilla y a muy bajo coste (figura 1.6 (c)).
- *Meta Quest 3*: visor de Realidad Virtual que se distingue por su integración de procesador y sistema Android, lo que le permite operar de forma autónoma sin necesidad de estar conectado a una fuente externa. Equipado con dos controladores que simulan las manos del usuario, ofrece una amplia capacidad

para el desarrollo de aplicaciones, especialmente en el ámbito de entretenimiento. Su combinación de características avanzadas y precio accesible lo consolida como uno de los visores líderes en el mercado, con un sólido historial de desarrollo y una extensa base de usuarios (figura 1.6 (d)).

- *Apple Vision Pro*: visor de realidad mixta que utiliza sus 12 cámaras para tomar en tiempo real imágenes del entorno y poder superponer elementos virtuales en ella. Además, cuenta con otros 6 sensores que combinados con las cámaras permiten la aplicación de robustos algoritmos de *tracking* tanto para los ojos del usuario como para sus manos. Permitiendo así una experiencia mucho más realista y evitando la necesidad de usar controles externos, como los mandos de las Meta Quest 3, para interaccionar con los elementos virtuales (figura 1.6 (e)).



Figura 1.6: Visores de VR

Además de los visores se han desarrollado otros elementos que tratan de mejorar la sensación de inmersión consiguiendo aumentar el realismo como los guantes o chalecos hápticos. Estos dispositivos no solo permiten el capturar la posición de las manos del jugador, sino que también cuentan con una serie de actuadores que permiten dotar al usuario de una retroalimentación táctil. Es decir, permiten al usuario sentir los objetos del entorno virtual durante su manipulación. Se han realizado muchos estudios sobre estos en los últimos años, como el chaleco háptico presentado por la Universidad Politécnica de Madrid para la «Spanish Robotics Conference»<sup>5</sup> de 2017 [1].

<sup>5</sup>Spanish Robotics Conference: <http://jnr2017.ai2.upv.es/en/home.html>

### 1.1.3. Comunicación red

La comunicación vía red es una constante actual en nuestro día a día. Desde su invención son innumerables las aplicaciones desarrolladas sobre este paradigma y la evolución que esto ha generado en nuestra sociedad. Este tipo de comunicación podría definirse como el conjunto de medios, protocolos y resto de herramientas que permiten el intercambio de información entre los distintos usuarios de la red. Dentro de las telecomunicaciones, existen diferentes tipos de redes como las redes de computadoras, redes telefónicas o Internet.

Esta última podría considerarse como la red que ha generado un mayor cambio en nuestra forma de comunicarnos. Tanto en el ámbito doméstico, como en el ámbito profesional o industrial, permitiendo por ejemplo, la comunicación entre distintos dispositivos autónomos. Entre sus muchas virtudes que le han hecho posicionarse como uno de los pilares de la comunicación humana podemos destacar:

- Interoperabilidad: internet está pensado para poder transmitir datos de diferente naturaleza a cualquier otra máquina que se encuentre conectada a la red sin necesidad de una adaptación especial para cada uno de los casos. Además, cuando hablamos de máquina estamos haciendo referencia a que los dispositivos que intercambian esta información no necesitan ser de la misma clase. Por ejemplo un ordenador podría comunicarse con un microcontrolador a miles de kilómetros de distancia.

Las características de los diferentes paquetes enviados se definen a través de los protocolos. Un protocolo de comunicación es el conjunto de reglas y convenciones que determinan cómo se lleva a cabo la comunicación entre dos o más dispositivos de una red. Estas reglas marcan, por ejemplo: formato, secuencia, tamaño, codificación, etc. Gran parte de la interoperabilidad que encontramos en Internet se debe a que permite el uso de diferentes modelos de comunicación y protocolos.

- Escalabilidad: la infraestructura de Internet está diseñada para ser altamente escalable, lo que significa que puede crecer y adaptarse fácilmente para satisfacer la demanda creciente de comunicación en línea sin comprometer el rendimiento.
- Alcance global: internet ofrece un alcance global, lo que significa que las comunicaciones pueden realizarse fácilmente entre personas, organizaciones y

dispositivos en todo el mundo. Todo esto utilizando una infraestructura común, por lo tanto, simplifica enormemente el despliegue de aplicaciones.

Además, continuamente se desarrollan nuevas mejoras en estas redes que buscan reducir las limitaciones que podemos encontrar en esta forma de comunicación. Por ejemplo una de las limitaciones más graves la podemos encontrar en la latencia, especialmente cuando los paquetes tienen que pasar por muchas máquinas intermedias antes de llegar a su destino. La latencia se refiere al tiempo que tarda un paquete de datos en viajar desde su origen hasta su destino. Esto es crucial para alguno de los casos de uso de esta tecnología, como en el envío de órdenes en tiempo real para sistemas críticos. En la actualidad se utilizan tecnologías de comunicación satelital como el 5G para solventar este problema.

## 1.2. Motivación

Este trabajo surge de la motivación de utilizar una tecnología emergente como la Realidad Virtual para poder mejorar los sistemas de teleoperación robótica. Consiguiendo con esto una sensación más realista y humana para el usuario, solucionando así los problemas derivados del uso de controles rudimentarios que requieren de formaciones muy complejas y resultados menos precisos. A su vez, este sistema busca mejorar la forma de visualización del entorno aprovechando la libertad que otorga el uso de espacios tridimensionales. Por otro lado, se apoya en los avances dentro de los sistemas de comunicación de alta velocidad para investigar sobre una teleoperación en tiempo real a la máxima distancia posible.

Acercándonos al ámbito más humano, busca cumplir uno de los objetivos principales de la robótica, como es el evitar que los humanos tengan que realizar actividades que pongan en riesgo su integridad física.

En relación a la motivación personal, desde la primera vez que tuve la oportunidad de desarrollar sobre un sistema VR, gracias a los cursos impartidos por Jesús, el tutor de este proyecto, me quede cautivado por ese mundo. Cuando planteé el tema de mi TFG no dude de que quería que mi última huella en la universidad fusionase este campo con mi campo de estudio, la robótica.

## 1.3. Objetivos

A continuación se detallan los objetivos perseguidos durante el desarrollo de este proyecto:

### 1.3.1. Objetivo principal

El objetivo principal de este proyecto es la creación de un sistema de teleoperación robótico a través de un entorno de Realidad Virtual. Con creación nos referimos a diseño, montaje de cada uno de los elementos, gestión de comunicaciones e implementación. Además, se busca que el sistema sea lo más natural, sencillo y preciso posible, basando toda la comunicación en la infraestructura de Internet para tratar de conseguir ser lo más generalista posible. Por otro lado, busca permitir una visualización realista y completa de la escena real para el usuario.

### 1.3.2. Objetivos específicos

Este objetivo principal se divide en una serie de subobjetivos, centrados tanto en las labores de investigación y formación, como en desarrollo:

- El software del robot y el del entorno de Realidad Virtual deben permitir una comunicación bidireccional entre ellos para poder en todo momento tener control sobre ambos sistemas.
- El software del robot debe ser capaz de interpretar la posiciones y rotaciones recibidas del entorno de Realidad Virtual y generar las trayectorias oportunas para alcanzarlas.
- El robot debe ser consciente de la posición de su elemento terminal, un gripper o garra en español, y contener el software para su control.
- El robot debe estar conectado a la red y poder admitir tanto conexión local como a distancia.
- El software de Realidad Virtual debe ser capaz de conocer la posición y rotación de las manos del usuario.

- El software de Realidad Virtual debe ser capaz mapear diferentes botones de los mandos para producir ciertas órdenes con ellos.
- El entorno de Realidad Virtual debe permitir la visualización de una proyección del mundo real durante la operación.
- El sistema debe permitir al usuario tener control sobre el gripper del robot.
- El sistema debe permitir al usuario realizar todas las labores necesarias para operar desde los propios mandos de Realidad Virtual.
- Utilizar Godot como software *Open Source* para la creación del escenario y control de la escena VR.

### 1.3.3. Objetivos relacionados

De la experimentación y la búsqueda de la continua mejora del sistema durante el trabajo han ido apareciendo una serie de objetivos relacionados que merece la pena comentar:

- El software del robot debe tener mecanismos de seguridad que eviten la posibilidad de movimientos peligrosos para el mismo o su entorno.
- El software del robot debe trabajar en varios hilos (*threads*) para poder leer nuevos mensajes entrantes mientras realiza movimientos y conseguir una mayor fluidez.
- La controladora del robot debe permitir una conexión remota VNC para permitir control completo a distancia.
- El entorno de Realidad Virtual debe contener un modelo 3D que indique la posición y movimiento del robot con una tasa de refresco adecuada.
- El sistema debe permitir al usuario tener un botón de retorno para devolver el robot a su posición inicial (*home*).
- El sistema debe permitir al usuario tener un botón para reiniciar el sistema del robot a distancia y desbloquear los frenos de este.

## 1.4. Estructura de la memoria

Este documento esta dividido en seis capítulos en los que se van recorriendo cada uno de los puntos principales que conforman este proyecto:

- **Capítulo 1: Introducción:** en el capítulo 1 se realiza una introducción al proyecto, explicando el contexto en el que este se desarrolla y los objetivos planteados para la resolución de la idea planteada.
- **Capítulo 2: Tecnologías utilizadas y Trabajos relacionados:** en el capítulo 2 recorreremos las tecnologías aplicadas y algunos trabajos relacionados que resultan de interés.
- **Capítulo 3: Desarrollo del proyecto:** en el capítulo 3 se irán explicando cada una de las etapas de desarrollo planteadas para la resolución, explicando los distintos problemas encontrados y lo pasos seguidos hasta la obtención del prototipo final.
- **Capítulo 4: Resultados:** en el capítulo 4 se explica el diseño final del sistema. Hablando por un lado de su descripción funcional y por otro lado su descripción técnica.
- **Capítulo 5: Experimentación y Validación:** en el capítulo 5 se comentan las distintas pruebas realizadas para validar el sistema. Estas se han dividido en experimentación técnica y en experimentación con usuarios.
- **Capítulo 6: Conclusiones:** en el capítulo 6 se hace un repaso del cumplimiento de los objetivos planteados, indicando los problemas principales encontrados durante el desarrollo y las posibles futuras vías de desarrollo. También se comentan los conocimientos adquiridos durante el desarrollo, así como la aplicación de lo aprendido durante la carrera.

## 1.5. Disponibilidad de código

Este proyecto sigue la filosofía *Open Source*, por lo cual todo el código desarrollado puede encontrarse en los siguientes repositorios:

- TelBiB<sup>6</sup>: repositorio principal del proyecto. Este contiene el *core* de la aplicación de Realidad Virtual, el programa que corre en el robot y el *plugin* que permite la configuración remota (VNC). Además incluye la *release* en la que se puede descargar directamente el ejecutable de la aplicación.
- RTDE PYTHON CLIENT LIB EXPANDED<sup>7</sup>: este repositorio contiene la librería de alto nivel para establecer comunicaciones RTDE, con los ejemplos usados para realizar pruebas durante el desarrollo del proyecto.

Además se ha creado una pagina web<sup>8</sup>: desde la que se puede acceder a estos y más recursos de interés, como por ejemplo videos de algunas de las pruebas realizadas.

---

<sup>6</sup>TelBiB Github: <https://github.com/porrasp8/TelBiB>

<sup>7</sup>RTDE Expanded Github: [https://github.com/porrasp8/RTDE\\_PYTHON\\_CLIENT\\_LIB\\_EXPANDED](https://github.com/porrasp8/RTDE_PYTHON_CLIENT_LIB_EXPANDED)

<sup>8</sup>TelBiB WebPage: <https://porrasp8.github.io/TelBiBWeb>

---

## Capítulo 2

# Tecnologías utilizadas y Trabajos relacionados

---

En este capítulo recorreremos las distintas tecnologías utilizadas en la elaboración de este proyecto, así como algunos trabajos relacionados que me han resultado de interés.

### 2.1. Trabajos relacionados

A pesar de tratarse de un área bastante poco explorada, podemos encontrar algunos trabajos de interés. Por ejemplo, en [2] se presenta una plataforma experimental para un sistema de teleoperación robótico en tiempo real. Este tiene como objetivo evitar tareas peligrosas para el operador humano y utiliza el sistema operativo de tiempo real QNX<sup>1</sup> y comunicación TCP. La búsqueda de una comunicación con restricciones de tiempo real resulta de gran interés para este trabajo y para el tratar de realizar trayectorias fluidas. El sistema propuesto se puede ver en forma de esquema en la figura 2.1.

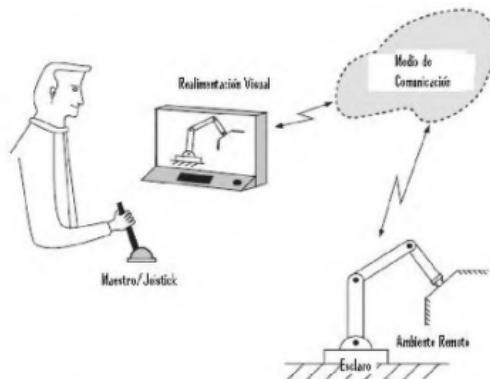


Figura 2.1: Esquema sistema de teleoperación en tiempo real QNX

<sup>1</sup>QNX: <https://blackberry.qnx.com/en>

Por otro lado, podemos centrarnos en sistemas más comerciales como el robot quirúrgico Da Vinci<sup>2</sup> (figura 2.2) [3]. Un robot que permite la realización de un gran abanico de operaciones con una precisión milimétrica controlado por un cirujano a través de la denominada «torre de visión». Esta combinación humano-máquina permite juntar la inteligencia y capacidad de resolución humana con la precisión y control robótico para conseguir el mejor resultado posible para el paciente. Además, este sistema permite la operación a distancia.



Figura 2.2: Sistemas quirúrgico Da Vinci

Acercándonos más al conjunto de tecnologías utilizadas en este proyecto, podemos encontrar [4]. En este TFG se plantea el control de un robot Pepper (figura 2.3)<sup>3</sup> mediante realidad virtual, aprovechando la propia sensorización de este robot y utilizando Unity. En este caso, debido al gran flujo de datos que debe manejar, toda la comunicación se realiza en una red local.



Figura 2.3: Robot Pepper de Softbank

---

<sup>2</sup>Da Vinci: <https://www.intuitive.com/en-us/products-and-services/da-vinci>

<sup>3</sup>Pepper: <https://us.softbankrobotics.com/pepper>

## 2.2. Tecnologías software

En esta sección se explicarán todas las tecnologías y herramientas software utilizadas para la realización de este trabajo.

### 2.2.1. Lenguajes de programación

La programación es un factor fundamental en el mundo digital, ya que gracias a esta somos capaces de interconectar distintos sistemas, procesar datos, simular entornos, etc. Las instrucciones que se utilizan para programar estos comportamientos son definidas por los distintos lenguajes de programación. A continuación se comentan los lenguajes más utilizados durante este proyecto:

#### Python

Python<sup>4</sup>[5][6] es un lenguaje de programación de alto nivel muy extendido en la actualidad. Destaca principalmente por su sencillez, gran comunidad y su repertorio de librerías que extienden su funcionalidad. Además, se trata de un lenguaje interpretado, es decir, no se compila antes de la ejecución. Permite una programación multiparadigma y puede ser ejecutado en múltiples plataformas.

Estas características hacen de Python un lenguaje idóneo para crear *scripts* que evalúen diferentes comportamientos, lo cual resultará crucial durante el desarrollo de este proyecto.

#### C#

C#<sup>5</sup> [7][8] es un lenguaje de programación multiparadigma diseñado por Microsoft para la plataforma .NET. Combina elementos de lenguajes orientados a objetos y lenguajes estructurados, lo que permite la creación de software escalable y robusto. Con una sintaxis similar a la de C++ y Java, C# ofrece características avanzadas como la administración automática de memoria a través del recolector de basura y un sistema de gestión de excepciones sólido. Además, proporciona acceso directo a las características de la plataforma .NET, lo que facilita el desarrollo de aplicaciones de escritorio, web y móviles con alto rendimiento y seguridad integrada.

---

<sup>4</sup>Python: <https://www.python.org/>

<sup>5</sup>C#: <https://dotnet.microsoft.com/en-us/languages/csharp>

Este lenguaje además se puede utilizar con el motor de videojuegos Godot y es el lenguaje principal del entorno VR del proyecto.

### 2.2.2. Plataforma creación de escena VR - Godot

Godot es un motor de desarrollo de juegos de código abierto y multiplataforma, diseñado para la creación eficiente de juegos 2D y 3D. Utiliza un lenguaje de scripting propio llamado GDScript. Además de este, Godot es compatible con otros lenguajes de programación como C#[9] y C++, lo que brinda a los desarrolladores flexibilidad según sus necesidades y preferencias. Con un enfoque en la facilidad de uso, Godot proporciona un conjunto completo de herramientas integradas para el diseño de juegos, incluidos editores visuales para la creación de escenas, animaciones y efectos especiales. Además, este también cuenta con una amplia comunidad que desarrolla continuamente nuevas librerías que aumentan su funcionalidad.

Su diseño sigue un enfoque modular en el que cada uno de los elementos que conforman la escena se definen como nodos, siguiendo una estructura en forma de árbol (figura 2.4). Cada uno de estos nodos tiene la capacidad de comunicarse con el resto, y su lógica de comportamiento está marcada por un script asociado al mismo. Existen muchos tipos de nodos, cada uno diseñado para tareas específicas, tales como: renderización de gráficos, interacción física, peticiones de red, interfaz de usuario, etc.

Sobre este motor de videojuegos se construye la escena de Realidad Virtual del proyecto.

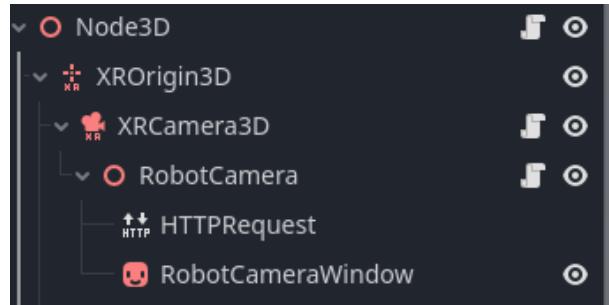


Figura 2.4: Ejemplo de conjunto de nodos en Godot

### GDScript

GDScript[10] es un lenguaje de scripting creado específicamente para el motor de desarrollo de videojuegos Godot. Se caracteriza por su sintaxis simple y fácil

de aprender, inspirada en Python. Además, ofrece una integración perfecta con el motor, lo que permite codificar *scripts* de manera eficiente y potente para controlar el comportamiento de los nodos y objetos en el juego. Está diseñado para seguir un esquema de programación secuencial en el que predominan dos funciones:

- *func \_ready()*: se ejecuta una sola vez y al inicio del programa. Está pensada para inicializar las variables necesarias y las conexiones con otros nodos o redes.
- *func \_process(delta)*: una vez se ha ejecutado la función *\_ready()* esta función sera continuamente llamada por cada cuadro de renderizado por lo que el tiempo entre una y otra estará determinado por los límites del equipo y el nivel de cómputo del programa. El parámetro *delta* representa el tiempo transcurrido entre el cuadro actual y el anterior. Esta función está pensada para ir actualizando el estado del nodo de forma continua.

## Godot XR tools

Como ya mencionamos al hablar sobre Godot, en la sección 2.2.2, una de sus características principales es la gran cantidad de herramientas y paquetes disponibles para facilitar el desarrollo. Existen paquetes que buscan dotar a este motor de soporte para poder utilizarse con distintas plataformas.

Godot XR tools es un paquete creado para permitir el desarrollo de aplicaciones VR utilizando este motor. Contiene tanto las herramientas para poder ejecutar entornos 3D, como los mecanismos básicos utilizados por los desarrolladores de Realidad Virtual para la creación de aplicaciones. Por ejemplo, permite marcar límites al área de juego, realizar el seguimiento de la posición de las manos del jugador, dotar al usuario de capacidad de desplazamiento por el entorno, etc.

Este paquete sigue los estándares OpenXR y WebXR por lo que es compatible con una gran cantidad de visores de Realidad Virtual como los que mencionamos en la subsección 1.1.2.

### 2.2.3. Software brazo robótico - Universal Robots

Para la realización de este proyecto se ha decidido utilizar un brazo robótico de la empresa Universal Robots, cuyas características hardware mencionaremos más adelante en la sección 2.4.1. En esta sección comentaremos las bases para el desarrollo software en estos sistemas utilizando las herramientas proporcionadas para ello.

#### UrScript

Para permitir a los desarrolladores diseñar aplicaciones eficientes que permitan utilizar todo el potencial de estos sistemas, la empresa Universal Robots diseñó un lenguaje de *scripting* propio denominado como URscript[11]. Este lenguaje permite a los usuarios configurar diversas acciones, como controlar entradas y salidas, establecer conexiones con otros dispositivos y definir trayectorias de movimiento para el robot.

El control de entradas y salidas permite interactuar con otros dispositivos y sistemas en el entorno de trabajo. Los usuarios pueden programar acciones específicas que respondan a señales de entrada, como sensores o interruptores, y generar salidas que controlen dispositivos externos, como válvulas o sistemas de transporte.

Tanto a través del Teach Pendant (subsección 2.4.1) (figura 2.5) como cargando el programa de forma externa. URScript ofrece herramientas para definir y ejecutar trayectorias de movimiento para el robot. Esto incluye la capacidad de especificar puntos de inicio y finalización, así como trayectorias intermedias, velocidades y aceleraciones. Los usuarios pueden programar movimientos para realizar distintas tareas, como ensamblaje, manipulación de piezas o soldadura.

Este lenguaje contiene un extenso abanico de funciones que pueden ser llamadas desde los programas. Además, cuenta con estructuras típicas de la programación tradicional como: bucles, sentencias condicionales, control de hilos de ejecución, etc.

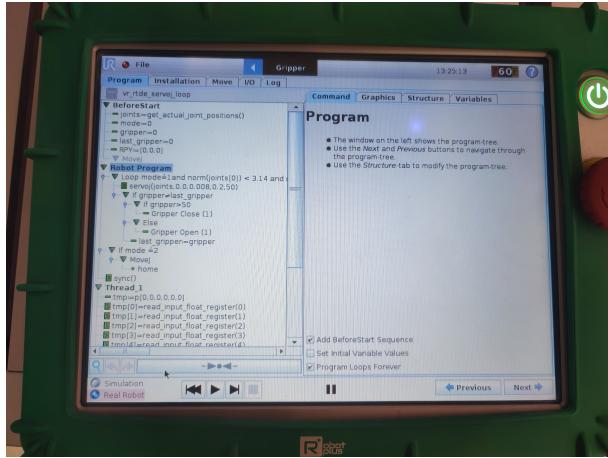


Figura 2.5: Programa URScript cargado en pantalla de control (Teach Pendant) de UR3

## URCaps

Un URCap es un *plugin* basado en Java diseñado para integrarse en PolyScope, la interfaz gráfica de programación de Universal Robots. Estos *plugins* amplían la funcionalidad de PolyScope al proporcionar nuevas pantallas de programación amigables y sentencias personalizadas. Permiten al usuario configurar *hardware* externo, crear nuevos modelos de programación y realizar operaciones complejas sin requerir una programación extensa. En esencia, un URCap facilita la personalización y la ampliación de las capacidades de programación de los robots de Universal Robots, permitiendo una mayor flexibilidad y adaptabilidad en una variedad de aplicaciones robóticas.

Esto permite que la comunidad de desarrolladores existente alrededor de estos sistemas pueda crear estos *plugins* y compartirlos con el resto de usuarios, ayudando a agregar multitud de funcionalidades a estos robots. Además, permite que las empresas de *hardware* robótico puedan acompañar sus dispositivos con *software* de control que simplifique su uso.

En nuestro caso, utilizaremos uno de estos URCap para controlar el elemento terminal configurado en el robot.

## URsim

URsim es una de las herramientas desarrolladas por Universal Robots para ayudar en la programación de sus sistemas. Esta herramienta permite simular el sistema operativo interno de uno de estos robots en una máquina virtual y poder acceder a toda la funcionalidad que tendríamos en el sistema real. Este presenta algunas limitaciones como el control de fuerza o la reducida visualización, ya que únicamente podremos ver una pequeña representación 2D de la posición del robot. En la figura 2.6 podemos ver la vista de una de las pantallas del simulador.

Esta herramienta resulta muy interesante para hacer pruebas de funcionamiento sin miedo a posibles movimientos incontrolados o golpes, evitando daños innecesarios al robot.

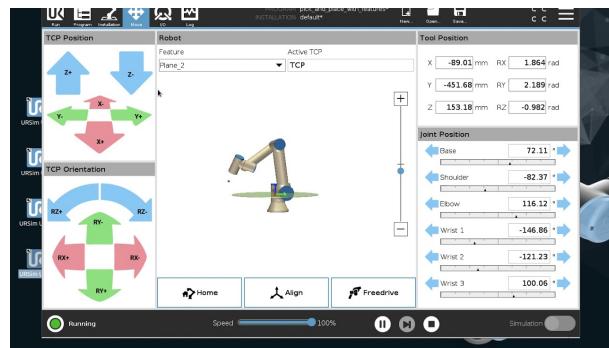


Figura 2.6: Vista de simulador URsim en pantalla movimiento

## 2.3. Tecnologías de comunicación

### 2.3.1. Sockets

Aunque tiene muchas definiciones, de manera general podemos definir un *socket*[12] como un método de comunicación entre un programa cliente y un servidor a través de una red. Normalmente, cada programa se encuentra en una máquina distinta.

En programación, los *sockets* se utilizan como una abstracción que permite a distintos procesos establecer una comunicación y transferir datos entre sí. Para esto, el *socket* necesita conocer las direcciones IP de la máquina que actúa como servidor y la que actúa como cliente y los puertos dentro de la máquina por los que están transfiriendo los datos estos procesos (figura 2.7). La mayoría de lenguajes de

programación actuales cuentan con librerías que implementan esta abstracción, como por ejemplo Python<sup>6</sup> o C#<sup>7</sup>.

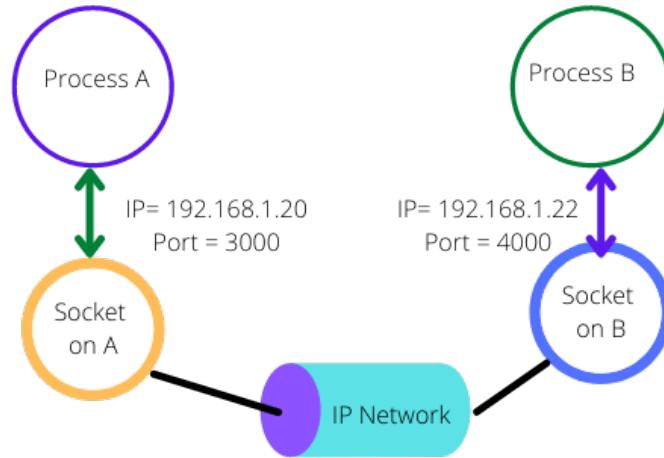


Figura 2.7: Comunicación entre dos procesos de distintas máquinas usando sockets

### 2.3.2. TCP

TCP es uno de los protocolos fundamentales de Internet. Su propósito principal es garantizar la entrega confiable de datos entre dispositivos en una red de computadores. Destaca principalmente por sus mecanismos de seguridad que aseguran que todos los paquetes transmitidos lleguen sin errores a su destino y en el mismo orden en que fueron enviados.

Además, TCP implementa un mecanismo de puertos que permite identificar diferentes aplicaciones en una misma máquina, facilitando la gestión y el control del tráfico de datos en la red.

### 2.3.3. HTTP

HTTP (*Hypertext Transfer Protocol*) es un protocolo de comunicación utilizado para la transferencia de datos en la *World Wide Web*. Este protocolo define la forma en que

---

<sup>6</sup>Python Sockets: <https://docs.python.org/3/library/socket.html>

<sup>7</sup>C# : <https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket>

los clientes (como navegadores) y los servidores web interactúan entre sí para solicitar y enviar información. Este protocolo opera sobre el modelo de solicitud-respuesta, donde el cliente envía una solicitud al servidor para obtener recursos, como páginas web o archivos, y el servidor responde proporcionando los datos solicitados. Utilizando métodos como GET, POST, PUT y DELETE, HTTP facilita la transmisión de datos de manera eficiente y confiable a través de la web, lo que lo convierte en un componente fundamental de la infraestructura de Internet.

Este protocolo es utilizado en nuestro caso para tomar las imágenes de la cámara IP del servidor web en el que son publicadas.

#### 2.3.4. RTDE

RTDE (*Real Time Data Exchange*) es un protocolo que permite la sincronización e intercambio de datos entre la controladora de los robots UR y aplicaciones externas. Este intercambio se realiza a través de una conexión TCP/IP estándar bidireccional. Este protocolo permite la comunicación entre ambos manteniendo las propiedades de tiempo real propias de estos brazos robóticos.

El funcionamiento de este protocolo se divide en dos etapas:

- *Conexión a la interfaz RTDE:* En esta fase el cliente debe iniciar la conexión con el servidor e indicar las variables de los paquetes de entrada y salida que se van a transmitir durante la comunicación. Es decir, tiene que elegir entre una lista de registros del robot cuáles va a utilizar para enviarle información y la lista de variables del robot que este debe transmitirle, como por ejemplo la posición del TCP o el ángulo de los *joints*. Esta configuración se almacena en un fichero en formato *.xml* y se transmite a través de la conexión TCP.
- *Bucle de comunicación:* En esta fase el cliente domina la comunicación, enviando datos y recibiéndolos cuando lo solicite. Normalmente, para poder recibir los paquetes en los registros del robot y asignar las correspondientes variables, se crea un nuevo hilo que lee la información de los registros continuamente.

En el *snippet 2.1* se puede ver un ejemplo de fichero de configuración de paquete RTDE. Los campos dentro de *state* se refieren a la información que el programa externo

desea recibir del robot y los campos dentro de *setp* los registros en los que desea escribir el programa externo dentro del robot.

---

```
<?xml version="1.0"?>
<rtde_config>
<recipe key="state">
<field name="target_q" type="VECTOR6D"/>
<field name="target_qd" type="VECTOR6D"/>
<field name="output_inrtdet_register_0" type="INT32"/>
</recipe>

<recipe key="setp">
<field name="input_double_register_0" type="DOUBLE"/>
<field name="input_double_register_1" type="DOUBLE"/>
<field name="input_double_register_2" type="DOUBLE"/>
<field name="input_double_register_3" type="DOUBLE"/>
<field name="input_double_register_4" type="DOUBLE"/>
<field name="input_double_register_5" type="DOUBLE"/>
</recipe>
</rtde_config>
```

---

Fragmento de código 2.1: Fichero de configuración de paquete de entrada y salida para una comunicación utilizando el protocolo RTDE

En [13] se puede acceder a la guía de variables de configuración.

Existe una librería<sup>8</sup> desarrollada en Python (subsección 2.2.1) por Universal Robots que encapsula la creación de estos canales y la comunicación en funciones de alto nivel.

Dentro de este proyecto, este protocolo, junto con los *sockets*, será utilizado para habilitar la comunicación bidireccional entre el robot y la aplicación de Realidad Virtual.

### 2.3.5. Dashboard Communication

Además de la comunicación RTDE (subsección 2.3.4) que se realiza por el puerto 30004, las controladoras UR habilitan un servidor de comunicación TCP/IP a través del puerto 29999. Esta comunicación recibe el nombre de *Dashboard Communication* y permite el envío de comandos de forma remota, estos normalmente se lanzarían a través del Teach Pendant. Permite, por ejemplo, el desbloqueo de frenos, armado del

<sup>8</sup>RTDE Python Lib: [https://github.com/UniversalRobots/RTDE\\_Python\\_Client\\_Library](https://github.com/UniversalRobots/RTDE_Python_Client_Library)

robot, carga de programas, cambio de modos, etc.

En la figura 2.8 podemos ver el esquema general de comunicación de estos robots.

En nuestra aplicación esta comunicación se utilizará para desarrollar un comportamiento de reinicio en caso de existir algún problema durante la operación.

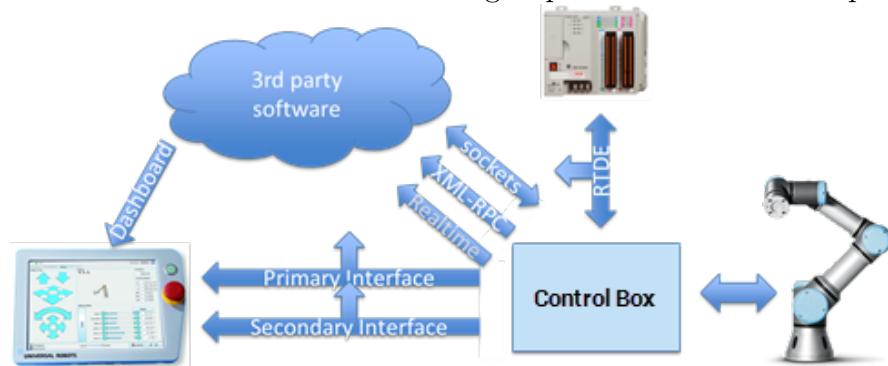


Figura 2.8: Esquema de comunicación UR

## 2.4. Tecnologías hardware

En esta sección se explicarán las diferentes tecnologías hardware utilizadas durante la realización de este trabajo.

### 2.4.1. Hardware sistema robótico - Universal Robots

Universal Robots es una empresa con origen en Dinamarca que ha conseguido posicionarse como el referente en el mercado de la robótica colaborativa (subsección 1.1.1). Esta empresa cuenta con varios modelos de brazos robóticos en función al tamaño y el software que soportan. Entre sus productos se encuentran las series *CB3* y *E*, que destacan por sus distintas características de conectividad y funcionalidad. Estas series han sido diseñadas para adaptarse a diversas necesidades y entornos de trabajo, ofreciendo soluciones versátiles y eficientes en el ámbito de la automatización industrial.

Esta empresa cuenta con extensos programas de formación para el uso de sus equipos, en parte debido a que una de sus principales filosofías es acercar estas tecnologías a la pequeña y mediana empresa. Muchos de estos programas de formación son gratuitos y se pueden hacer en línea.

En la figura 2.9 podemos ver los 3 brazos pertenecientes a la serie CB3 y la diferencia de tamaño entre estos modelos.

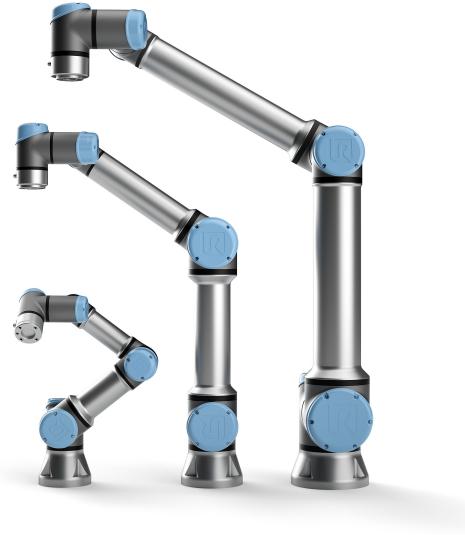


Figura 2.9: Modelos de brazos robóticos CB3-series (UR3, UR5 y UR10)

### Controladora y Teach Pendant

Los brazos robóticos de Universal Robots están controlados por un sistema operativo basado en Linux dispuesto en su caja controladora (figura 2.10). Esta controladora además dispone de varias conexiones que permiten el adaptar diferentes elementos al sistema del robot, como por ejemplo conexión Ethernet y diferentes entradas y salidas digitales programables.

Este sistema también cuenta con una pantalla táctil denominada *Teach Pendant* que permite la interacción del operario con el robot. Entre sus funcionalidades destacan:

- Inicialización del sistema (encendido de motores, desbloqueo de los frenos, carga de programa, etc).
- Visualización del estado y posición de cada uno de los *joints* del robot.
- Movimiento de cada uno de los ejes en diferentes modos de desplazamiento y referencia.

- Desbloqueo de movimiento manual del brazo.
- Creación de programas en lenguaje UrScript (subsección 2.2.3).
- Configuración de sistema y seguridad del robot.
- Configuración *plugins* (subsección 2.2.3).

Estos elementos dotan al sistema de altas capacidades de desarrollo y además, permiten que sea fácilmente utilizado por operarios sin necesidad de un alto nivel de especialización.

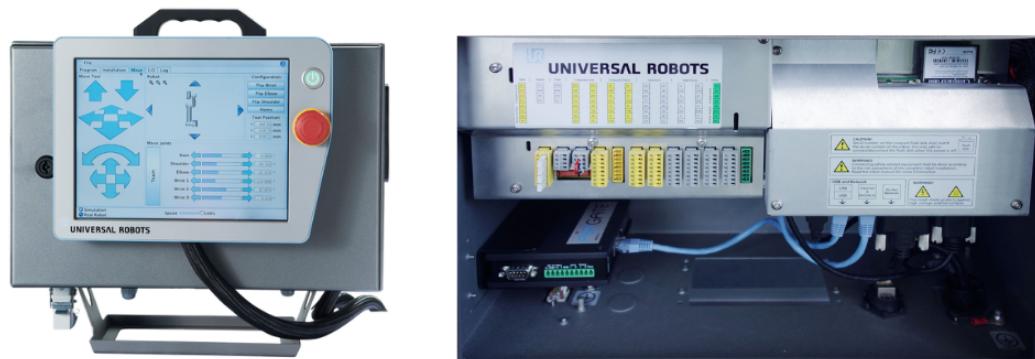


Figura 2.10: Caja controladora Universal Robots

## UR3

Dentro de los diferentes modelos de brazos robóticos de Universal Robots (subsección 2.4.1) encontramos al UR3[14]. Este destaca por ser la versión más pequeña desarrollada por esta empresa y, por lo tanto, la más utilizada como demostrador tecnológico y en investigación debido a su precio reducido y fácil portabilidad.

En la tabla 2.1 aparecen las características principales del robot UR3. Mientras que en la figura 2.11 podemos ver una imagen del robot real.

El modelo UR3 de la serie CB3 es el que ha sido utilizado durante el desarrollo de este proyecto.

Parámetros	Valores
Numero de ejes	Brazo de 6 ejes
Carta de pago	3Kg (6.6lbs)
Alcance	500mm (19.7in)
Peso	11kg (24.3lbs)
Velocidad	Efecto final:1m/s
Número pines digitales	16 entrada/16 salida
Número pines analógicos	2 entrada/2 salida

Cuadro 2.1: Especificaciones UR3



Figura 2.11: Brazo robótico UR3

#### 2.4.2. Robotiq Gripper 2F-85

El 2F-85 es una herramienta terminal en forma de garra diseñada por la empresa Robotiq para diferentes modelos de brazos robóticos, entre ellos los modelos de Universal Robots. Además, este *gripper* cuenta con su propia URCap (subsección 2.2.3) que permite controlar la velocidad, fuerza y posición de este de forma sencilla.

En la tabla 2.2 podemos ver las especificaciones del *gripper* y en la figura 2.12 una imagen de este.

Este *gripper* ha sido elegido como la herramienta terminal del robot para este proyecto.

Parámetros	Valores
Apertura	85mm (3.3in)
Fuerza de agarre	20 to 235N
Payload	500mm (19.7in)
Peso	0.9kg (2lbs)
Velocidad cierre	De 20 a 150mm/s

Cuadro 2.2: Especificaciones Gripper 2F-85



Figura 2.12: Gripper 2F-85 de la empresa Robotiq

### 2.4.3. Meta Quest 2

El visor Meta Quest 2 es uno de los más extendidos al día de hoy gracias a sus altas capacidades y reducido precio. Este cuenta además con dos mandos que unidos a sus cámaras le permiten realizar un robusto seguimiento de las manos del jugador. La gran comunidad que tienen detrás y las facilidades para desarrollar contenido en ellas las hacen ser una gran alternativa para la experimentación.

En la tabla 2.3 podemos ver las especificaciones de este visor.

Parámetros	Valores
Peso	503 gramos
Tipo de panel	LCD
Resolución	1.920 x 1.832 por ojo
Frecuencia de refresco	72 Hz

Cuadro 2.3: Especificaciones visor Meta Quest 2

#### 2.4.4. Router TP-Link Archer AX23

El router TP-Link Archer AX23 es un dispositivo pensando para el mercado doméstico pero con un gran rendimiento. Está equipado con Wi-Fi 6 y brinda velocidades de hasta 1800 Mbps en la banda de 5 GHz y 574 Mbps en la banda de 2.4 GHz, lo que permite una transmisión fluida de datos sin interrupciones. Su capacidad para manejar múltiples dispositivos simultáneamente garantiza una experiencia de red sin demoras, incluso en entornos con una gran cantidad de dispositivos conectados.

En la tabla 2.4 podemos ver las especificaciones de este router. Y en la figura 2.13 una imagen de este.

Este ha sido el utilizado en este trabajo para habilitar la comunicación entre sus distintos elementos.

Parámetros	Valores
Dimensiones	260,2 / 135,0 / 38,6 mm
Puertos	1 puerto WAN y 4 puertos LAN
Procesador	CPU de doble núcleo
Velocidades	5 GHz: 1201 Mbps / 2.4 GHz: 574 Mbps

Cuadro 2.4: Especificaciones router TP-Link Archer AX23



Figura 2.13: Router TP-Link Archer AX23

#### 2.4.5. Ordenador HP VR Backpack G2

El HP VR Backpack G2 (figura 2.14) es un ordenador diseñado esencialmente para el procesamiento de entornos de Realidad Virtual. Se trata de un ordenador extraíble con una batería interna que le permite no necesitar estar conectado a la corriente. Además, incluye una mochila que permite poder portarlo para tener una buena libertad de movimiento durante la inmersión en escenas VR. Está equipado con un Intel Core i7 y una tarjeta gráfica NVIDIA RTX 2080 que le permiten unas altas prestaciones computacionales.

La necesidad de disminuir al máximo los tiempos de procesamiento y conseguir una buena tasa de actuación es fundamental para el buen desempeño del sistema que se va a implementar. Por todo esto se seleccionó este ordenador como el encargado de llevar a cabo todo el procesamiento de escena y la comunicación. Evitado los posibles problemas de rendimiento derivados del uso del procesador de las Meta Quest 2.



Figura 2.14: Ordenador HP VR Backpack G2

#### 2.4.6. Raspberry Pi 4 y RaspiCam

La Raspberry Pi 4 (figura 2.15 (a)) es un ordenador monoplaca de bajo coste muy extendido en la actualidad. Este cuenta con un tamaño muy reducido y una gran cantidad de puertos, tanto de uso general, como de control de señales. Además, cuenta con múltiples dispositivos de expansión que pueden conectarse a sus diferentes puertos para extender sus funcionalidades. Entre ellos se encuentra la RaspiCam (figura 2.15 (b)). Una serie de pequeñas cámaras que se integran perfectamente en la placa con tecnología *plug & play*.

La combinación de ambos elementos convierte esta placa en la cámara IP ideal para el sistema de este proyecto. Ya que permite la creación de un propio servidor web[15] en el dispositivo y dota de un control total sobre el tratamiento de las imágenes, protocolos de comunicación, etc. Además, deja una puerta abierta para poder añadir más elementos al sistema en el futuro.



(a) Raspberry Pi 4 Model B

(b) RaspiCam V2.1

Figura 2.15: Raspberry Pi 4 y Raspicam

---

## **Capítulo 3**

# **Desarrollo del proyecto**

---

En este capítulo se irán detallando cada una de las fases de desarrollo del proyecto. Estas fases se han llevado a cabo inspirándose en las metodologías *Agile*. Estas metodologías son ampliamente utilizadas para la gestión de proyectos de software y se centran en la división del trabajo final en iteraciones más pequeñas que permiten ir obteniendo prototipos que cumplan con una serie de objetivos marcados para cada una de las fases. Esta secuencia de subobjetivos va en sintonía con los objetivos principales del proyecto y la combinación de ellos forma el resultado final esperado de este. Comúnmente esta metodología es utilizada para gestionar grupos de varios desarrolladores, pero en este caso ha sido adaptada para el desarrollo individual. Se ha optado por esta adaptación, ya que esta forma de trabajo facilita la documentación y ayuda a ir resolviendo los problemas de forma consistente.

Cada una de estas etapas se puede dividir en: objetivos planteados, desarrollo y resultado o prototipo. En concreto este proyecto está dividido en 6 diferentes etapas que detallaremos a continuación:

### **3.1. Etapa 0**

Esta primera fase se centra en la familiarización con las principales tecnologías utilizadas y la toma de algunas decisiones de diseño sobre las plataformas utilizadas.

#### **3.1.1. Objetivos de diseño**

En concreto los objetivos de esta etapa se centran en la familiarización con las distintas herramientas de desarrollo, funciones de movimiento y tecnologías de comunicación del brazo robótico UR3. Además, se busca conseguir un prototipo muy simple con comunicación local que permita realizar movimientos con el robot desde un

programa externo.

### 3.1.2. Diseño e implementación

En primer lugar, es necesario conocer las distintas capacidades del UR3 y entender su funcionamiento para poder familiarizarse con él. Durante la carrera utilicé el «hermano mayor» de este, el UR5, por lo cual ya estaba ligeramente familiarizado con el Teach Pendant y las funciones básicas de movimiento de este dispositivo. Para poder recuperar y afianzar estos conocimientos me ayudé de la página oficial de formación de Universal Robots, la denominada Universal Robots Academy<sup>1</sup>. Esta página contiene numerosos tutoriales interactivos que te permiten recorrer cada una de las características y configuraciones del robot y guías introductorias a su lenguaje de scripting 2.2.3. En la figura 3.1 (a y b) podemos ver dos de las ventanas del curso inicial de creación de programas.



Figura 3.1: Pantalla del curso de creación de programas en URacademy

Una vez adquirido este conocimiento realicé dos programas de prueba en el robot tratando de entender la diferencia entre las distintas funciones de movimiento como MoveJ o MoveL. Esto me llevo también a encontrar el manual de *scripting* completo del robot en el que vienen estas y el resto de funciones explicadas al detalle.

Una vez ya sabia crear y ejecutar programas de forma local en el robot, era momento de entender como podía comunicarle estos movimientos desde otros dispositivos. Mi investigación me llevo a encontrar el protocolo de comunicación RTDE (ya mencionado en la subsección 2.3.4) y su implementación de alto nivel como librería escrita en

<sup>1</sup>UR Academy: <https://academy.universal-robots.com/es>

lenguaje Python.

Para realizar mis primeras pruebas de comunicación conecté mi ordenador directamente a la controladora del robot a través de un cable Ethernet y configuré manualmente una dirección local en cada uno de los dispositivos. A continuación probé la conectividad entre ambos probando algunos de los programas de ejemplo de la propia librería y comencé a tratar de entender el funcionamiento de esta comunicación.

Entre toda la investigación de esta primera fase, cabe destacar la elección de utilizar la función de movimiento ServoJ en lugar de las mencionadas anteriormente, que comúnmente son más utilizadas. Esto se debe a que, a diferencia del resto de funciones de movimiento, ServoJ es no bloqueante, lo que permite modificar la trayectoria en todo momento, consiguiendo movimientos fluidos al pasarle varios puntos que sigan una determinada dirección. Además, esta función es capaz de funcionar a 125 Hz, un tiempo superior al que comúnmente operan los equipos de Realidad Virtual. La única dificultad que plantea esta función es que solo es capaz de recibir las posiciones de cada uno de los servomotores del brazo, por lo que el cálculo de trayectorias se debe realizar externamente a ella.

Finalmente, para probar todos los conocimientos aprendidos y conseguir comenzar a controlar el robot desde un programa externo, se diseñó un *script* de Python capaz de recabar la información de un mando Logitech F710 (figura 3.2) y mapear a esos valores a posiciones de dos de los seis ejes del brazo, que aun resultando un rango de movimiento muy limitado permitía visualizar el funcionamiento. Para esta labor también se desarrolló el correspondiente *script* dentro del sistema robótico encargado de leer estos valores a través de la conexión local y realizar un movimiento MoveJ (por simplicidad) hasta la posición calculada.



Figura 3.2: Mando Logitech F710

### 3.1.3. Resultados

En resumen, en esta primera etapa aprendí sobre el funcionamiento, forma de desarrollo y comunicación de los sistemas de Universal Robots e investigué sobre las diferentes funciones de movimiento del robot. En lo que respecta al desarrollo, fui capaz de crear una aplicación externa capaz de comunicarse con el robot y enviarle información de movimiento mapeada desde un mando de control remoto. Estos resultados generaron una buena base para el proyecto, así como una gran confianza en su viabilidad.

Todo el código desarrollado durante esta etapa se encuentra en el siguiente repositorio publico de Github: *RTDE\_PYTHON\_CLIENT\_LIB\_EXPANDED*<sup>2</sup>.

## 3.2. Etapa 1

En esta etapa se desarrollará en paralelo el aprendizaje en las tecnologías de Realidad Virtual y el desarrollo y configuración de las comunicaciones del sistema.

### 3.2.1. Objetivos de diseño

Los objetivos de esta etapa se pueden dividir en 3 tareas principales:

- Aprender a desarrollar con Godot y las Oculus Quest 2.
- Configurar una red propia para la comunicación con el robot.

---

<sup>2</sup>RTDE\_PYTHON\_CLIENT\_LIB\_EXPANDED Github: [https://github.com/porrasp8/RTDE\\_PYTHON\\_CLIENT\\_LIB\\_EXPANDED](https://github.com/porrasp8/RTDE_PYTHON_CLIENT_LIB_EXPANDED)

- Conseguir utilizar el protocolo RTDE desde Godot.

### 3.2.2. Diseño e implementación

Durante la primera parte de esta etapa traté de familiarizarme con el funcionamiento de Godot y con el paquete de Realidad Virtual de este motor de videojuegos (subsección 2.2.2). Además de su enfoque *Open Source* y su integración con los sistemas de VR, una de las cosas que me llevó a usar este entorno de desarrollo fue el haber trabajado previamente con él en el instituto. Estos conocimientos previos me permitieron agilizar en gran medida esta tarea.

Para recuperar esos conocimientos y obtener los necesarios para trabajar con entornos de Realidad Virtual aproveché los tutoriales que pueden encontrarse en su propia página web como el *Step by Step*<sup>3</sup> y el *VR starter*<sup>4</sup>.

Además, aproveché las plantillas de desarrollo de los tutoriales de Realidad Virtual para poder empezar a crear mi propio entorno para la aplicación final. En la figura 3.3 podemos ver la pantalla de edición del entorno con una de estas plantillas.

En esta primera tarea también aprendí a conectar las Meta Quest 2 al ordenador, donde se iba a llevar a cabo el procesamiento, a través de Quest Link<sup>5</sup>. Con esto pude ejecutar los primeros escenarios de pruebas comprobando el correcto desempeño de los dispositivos.

---

<sup>3</sup>[https://docs.godotengine.org/en/stable/getting\\_started/step\\_by\\_step/index.html](https://docs.godotengine.org/en/stable/getting_started/step_by_step/index.html)

<sup>4</sup>[https://docs.godotengine.org/en/3.1/tutorials/VR/vr\\_starter\\_tutorial.html](https://docs.godotengine.org/en/3.1/tutorials/VR/vr_starter_tutorial.html)

<sup>5</sup><https://www.meta.com/en-gb/help/quest/articles/headsets-and-accessories/oculus-link/set-up-link/>

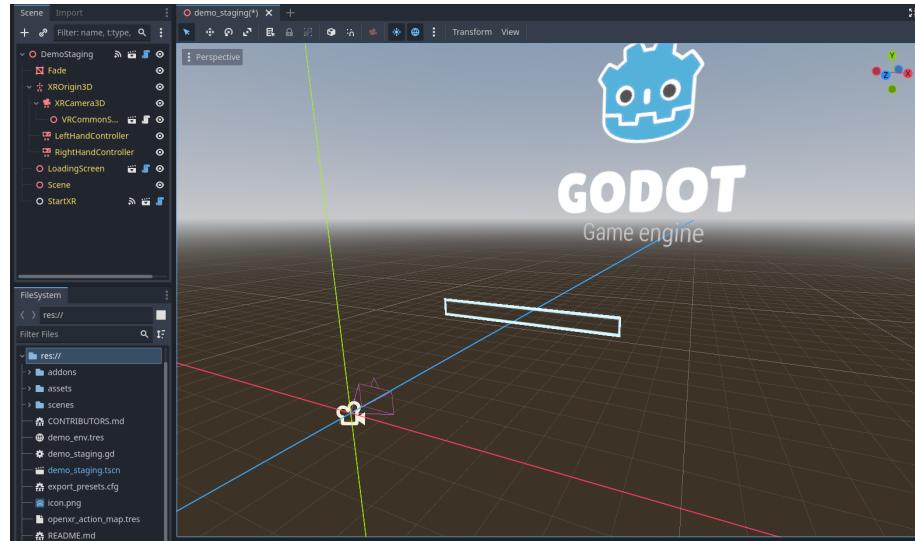


Figura 3.3: Escena de pruebas de Godot XR tools

La segunda tarea planteada para esta etapa era conseguir crear una red inalámbrica de comunicaciones para el sistema robótico. La idea era conseguir una red propia que sirviera para conectar de manera local al robot y el resto de sistemas que se integrasen con él. Y que en el futuro se le pudiera dar salida a una red más amplia.

Para esta comunicación se utilizó un router doméstico *TP-Link Archer AX23* (subsección 2.4.5) conectado al puerto Ethernet del robot, previamente usado en la etapa 0. A su vez se aprovechó la red WIFI (figura 3.4) de este para poder conectar otros dispositivos como el ordenador en el que posteriormente se llevarían a cabo el procesamiento de las gafas VR. Para simplificar la conexión de estos dispositivos se dotó de direcciones IP estáticas (figura 3.5) a los dispositivos que irían conectados al sistema.



Figura 3.4: Red WIFI inalámbrica del sistema robótico

INNOVACION12	A0-51-0B-2E-5C-07	192.168.0.100	Permanent
ivan-Msi	98-3B-8F-D8-44-38	192.168.0.101	Permanent

Figura 3.5: IPs de ordenador de procesamiento VR y ordenador personal configuradas en la red UR3

Como última tarea de esta etapa, se propuso investigar y desarrollar una forma de comunicación entre un nodo de Godot y la controladora del UR3. Basándome en el funcionamiento de la librería para crear clientes con comunicación RTDE utilizada en la etapa 0, traté de idear un proceso de ingeniería inversa.

Para realizar este proceso, primero debemos recordar que la comunicación a través de la interfaz RTDE se lleva a cabo mediante una conexión TCP/IP entre ambas máquinas. Luego, se asignan valores a paquetes predefinidos con los tipos de mensajes que deben transmitirse. Posteriormente se realizan las transmisiones marcadas por el cliente hasta que se cierra la conexión.

Dado que Godot permite el uso del lenguaje C#, que soporta la creación de *sockets* para comunicación TCP/IP, podemos replicar este comportamiento. Para esto recorri las funciones necesarias de conexión, codificación y transferencia de la librería cliente, y las adapté para integrarlas en un nuevo nodo de nuestro entorno en Godot, que será el encargado de gestionar las comunicaciones. En la figura 3.6 podemos ver el esquema de funcionamiento del protocolo.

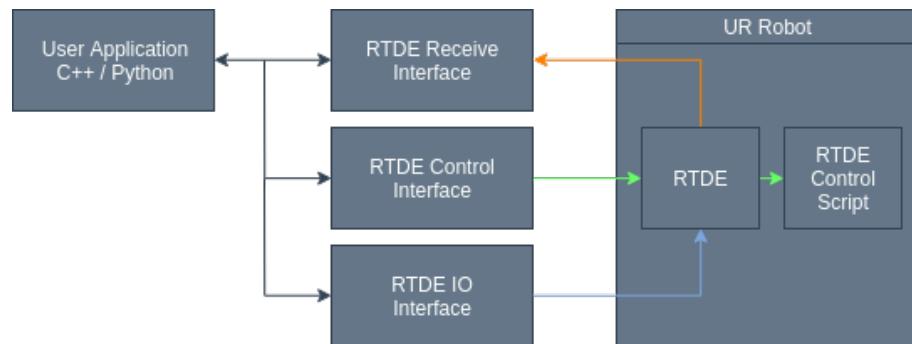


Figura 3.6: Esquema de funcionamiento del protocolo RTDE

### 3.2.3. Resultados

Se podría considerar que el resultado de esta etapa sentó las bases definitivas del proyecto. Por un lado, estableciendo un sistema de comunicaciones que nos permitiese realizar pruebas entre los dos elementos fundamentales de nuestro proyecto. Mientras por otro, adquiriendo los conocimientos necesarios para desarrollar nodos en Godot, lo cual resulta crucial para las siguientes etapas.

## 3.3. Etapa 2

En esta etapa se empezarán a combinar los conocimientos y pruebas realizadas durante las etapas anteriores. Tratando de conseguir realizar la primera prueba de operación

### 3.3.1. Objetivos de diseño

Los objetivos de esta etapa se pueden dividir en 3 tareas principales:

- Probar nodo de comunicación RTDE desde Godot.
- Transformación de movimientos lineales y rotacionales entre Godot y entorno UR.
- Realizar primera operación (limitada y a baja velocidad).

### 3.3.2. Diseño e implementación

Antes de llevar a cabo la primera prueba de operación es necesario poder establecer la comunicación entre Godot y la controladora del robot. Para esto aprovechamos el script de comunicación utilizado para el prototipo de la etapa 0 y el código de comunicación desarrollado en la etapa 1. Combinando ambas y lanzando el escenario Godot y dando valores aleatorios de posición, podemos ver como la comunicación se realiza satisfactoriamente.

Ahora necesitaremos poder extraer la posición y rotación de la mano del usuario dentro de la aplicación. Una vez con esta información necesitaremos conocer los ejes de coordenadas del entorno de VR y del robot para poder hacer las transformaciones oportunas entre ambos. Con las rotaciones tendremos que realizar este mismo procedimiento, pero teniendo también en cuenta el formato en el que esta información es devuelta. Todo este procedimiento quedará detallado más adelante en la sección 4.2.

Dentro de estas transformaciones también necesitaremos tener en cuenta los posibles desplazamientos (*offsets*) entre el extremo del mando de Realidad Virtual y la posición deseada para el elemento terminal del brazo robótico. En el *snippet* 3.2 se puede ver como se completa en el nodo de Godot el paquete de datos que se enviará al robot con las posiciones ya transformadas y sus respectivos *offsets*.

---

```
// Add offsets and assign to registers
UrInputs.input_double_register_0 = robot_posx;
UrInputs.input_double_register_1 = robot_posy;
UrInputs.input_double_register_2 = robot_posz + Constants.POSZ_OFFSET;
UrInputs.input_double_register_3 = robot_rotx + Constants.ROTX_OFFSET;
UrInputs.input_double_register_4 = robot_roty + Constants.ROTY_OFFSET;
UrInputs.input_double_register_5 = robot_rotz;
UrInputs.input_int_register_25 = 1; // ServoJ mode
```

---

Fragmento de código 3.1: Formación mensaje salida aplicación VR para movimiento. Asignación de posiciones y rotaciones al robot.

A continuación podemos llevar a cabo la última tarea de la etapa. Añadiendo el modo de movimiento ServoJ en el *script* inicial y conectando todos los elementos a la red, iniciamos la trasmisión de datos del programa. En la subsección 3.3.3 comentaremos el desempeño de este primer prototipo.

### 3.3.3. Resultados

El resultado final de esta etapa marca el primer prototipo «funcional» del proyecto. Durante estas primeras pruebas se consiguió que el robot reaccionase a los estímulos enviados desde el HMI pero con ciertas limitaciones. Analizando estas limitaciones se pueden destacar las siguientes: parones continuos del robot debido a singularidades o choques, dificultades para controlar los giros y una clara falta de desconocimiento de la situación desde el sistema de Realidad Virtual. Estos resultados llevan a plantear la necesidad de poder realizar pruebas de funcionamiento sin poner en riesgo la integridad del robot que estudiaremos durante la siguiente etapa.

## 3.4. Etapa 3

Esta etapa se centrará en mejorar el funcionamiento del prototipo anterior, tratando de mejorar la comprensión de la escena y asegurando un desempeño más seguro del sistema.

### 3.4.1. Objetivos de diseño

Los objetivos de esta etapa pueden ser divididos en 4 tareas principales:

- Configuración y uso de simulador UrSim.
- Introducir límites de seguridad en el robot.
- Comunicación bidireccional Robot-Godot para mejorar el entorno virtual.
- Añadir cámara IP al sistema.

### 3.4.2. Diseño e implementación

Después de las pruebas realizadas durante la etapa anterior, aparece la necesidad de poder llevar a cabo ensayos sin poner en riesgo el sistema robótico. Una vez investigadas las posibles soluciones, encontré la herramienta UrSim (subsección 2.2.3). Esta herramienta permite simular el sistema operativo de la controladora del robot en una máquina virtual Linux. Una vez configurada esta máquina virtual y la conexión de esta con el ordenador *host* a través de una conexión *bridged adapter* podemos empezar a realizar nuestras pruebas en el simulador como si del robot real se tratase. Durante el resto del proyecto sera una constante el probar nuevo código primero en el simulador y luego llevarlo al sistema real. El único cambio que hay que realizar para utilizar un sistema u otro será seleccionar la IP adecuada para cada caso.

Otra de las necesidades que quedó latente tras las pruebas de la etapa 2 fue la necesidad de introducir medidas de seguridad al brazo robótico que eviten la posibilidad de choques con otros elementos del entorno. La solución indicada por Universal Robots para estos casos es la configuración de planos de seguridad. Esto permite marcar al robot una serie de planos prohibidos alrededor de su entorno que no podrá superar. En caso de traspasar mínimamente una de estas limitaciones el sistema se detendrá entrando en el estado de *Parada de emergencia*. En la figura 3.7 (a y b) podemos ver el menú de configuración de planos de seguridad y la forma de este en el espacio del robot real.



Figura 3.7: Configuración planos de seguridad UR

Una vez ya hemos dotado al robot de un cierto grado de seguridad, tenemos que buscar soluciones para mejorar otro de los principales problemas del primer prototipo. Este problema se refiere a la falta de conocimiento del operario sobre lo que está sucediendo en el entorno real. Este problema es resuelto desde dos enfoques diferentes:

- Comunicación bidireccional entre Godot y el sistema robótico: la idea principal de esta tarea es la de configurar nuestra conexión RTDE para que la aplicación de Realidad Virtual reciba información de interés del sistema robótico. Esta información se refiere a datos de posicionamiento de cada uno de los servomotores del robot, rotación del elemento terminal en un determinado instante o los datos que se consideren oportunos para cada situación. La idea es que esta información permita retroalimentar al operario conociendo como afectan sus movimientos a los movimientos del sistema real y la trayectoria que está siguiendo el robot en cada instante.

Esta información se puede utilizar para mostrar diferentes elementos en la escena VR. Inicialmente, con objetivo de mejorar el conocimiento de las rotaciones por parte del usuario, se introduce una figura con forma cónica situada en la posición en la que se encuentra el elemento terminal del robot. En posteriores etapas se planteará como mejorar esta visualización. En la figura 3.8 se puede ver la posición de este elemento durante una de las pruebas.

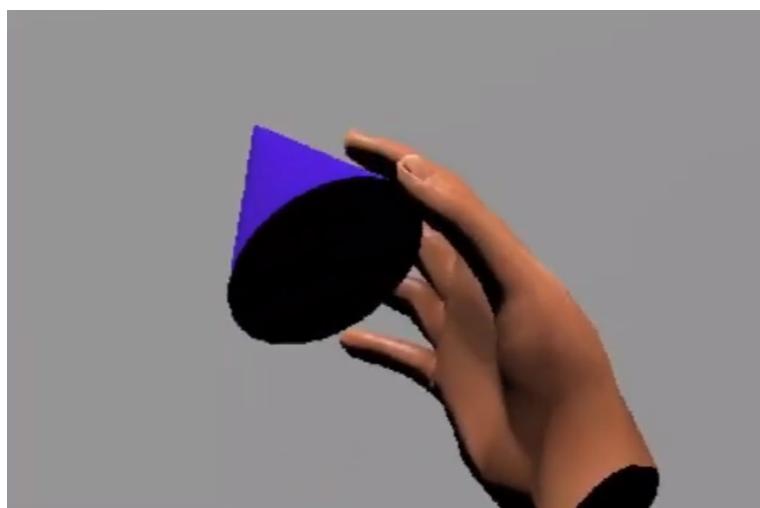


Figura 3.8: Cono de visualización de elemento terminal del UR3

- Instalación de cámara IP en el entorno del sistema robótico: esta tarea nace de la necesidad por parte del usuario de también conocer la situación del entorno que rodea al sistema. Apoyándonos en la red de comunicación ya creada, la forma más sencilla de realizar la trasmisión de imágenes resulta a través de una cámara IP conectada a esta misma red. Para poder leer las imágenes que publica esta cámara realizamos peticiones HTTP (subsección 2.3.3) a la URL donde estas se están publicando, en nuestro caso a `http://<ip_camara>/shot.jpg`. Es decir, solicitamos el recurso `shot.jpg` del dispositivo que actúa como cámara. En 3.2 se encuentra el *snippet* correspondiente a la solicitud HTTP realizada desde el nodo correspondiente de Godot. Esta imagen la cargaremos como textura en un nodo de tipo *Sprite 3D* que actuará como pantalla de visualización dentro del entorno VR. Además, a este nodo le añadiremos un sistema de seguimiento de vista del usuario y otro para evitar en lo posible que se hunda debajo del escenario. En la figura 3.9 podemos ver la imagen capturada por la cámara durante una prueba de operación.

---

```

#-- Init callback and connect
http_request.request_completed.connect(self._http_request_completed)

#-- Make a new http request and check error
var http_error = http_request.request(CAMARA_URL)
if http_error != OK:
    print("An error occurred in the HTTP request.")

#-- Callback and texture change
func _http_request_completed(result, response_code, headers, body):

    if response_code == 200:
        #-- Load new image
        var image = Image.new()
        var image_error = image.load_jpg_from_buffer(body)
        if image_error != OK:
            print("An error occurred while trying to display the image.")

        #-- Transform image to texture and assign to the 3D
        $RobotCameraWindow.texture = ImageTexture.create_from_image(image)
        print("HTTP: texture changed")

```

---

Fragmento de código 3.2: Solicitud HTTP en GdScript para leer la información de una cámara IP

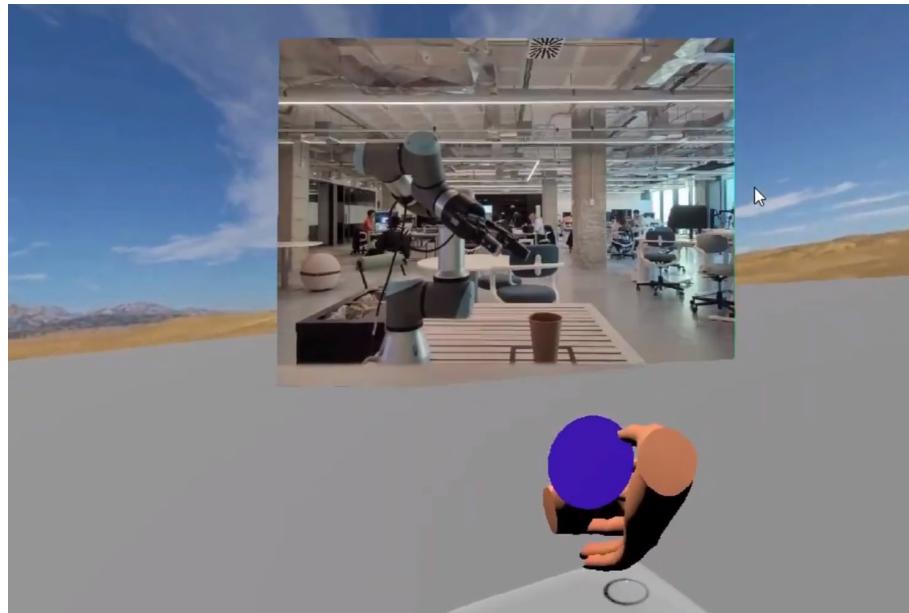


Figura 3.9: Imagen capturada por cámara IP proyectada sobre la escena de Realidad Virtual

### 3.4.3. Resultados

Los resultados obtenidos en esta etapa marcan un prototipo con unas grandes mejoras en lo que respecta a funcionalidad, seguridad y comprensión del entorno para el operario. Además, estos avances permiten una mejor calidad a la hora de realizar diferentes ajustes en los cálculos de trayectorias del sistema.

## 3.5. Etapa 4

### 3.5.1. Objetivos de diseño

Los objetivos de esta etapa pueden ser divididos en 5 tareas principales:

- Instalación de *gripper* y *software* de control.
- Mapeo de botones del mando y lectura desde aplicación de Godot.
- Comportamiento botón movimiento.
- Comportamiento botón retorno.
- Comportamiento botón *gripper*.

### 3.5.2. Diseño e implementación

Para poder realizar labores de operación realistas, como por ejemplo clasificación de elementos, labores de *pick and place*, etc., instalaremos un *gripper* como elemento terminal del brazo robótico. Elegí el modelo 2F-85 de la empresa Robotiq (subsección 2.4.2) debido a su fácil integración con los robots de Universal Robots. Su instalación sigue un formato *plug & play* por lo que únicamente tendremos que conectar este a la controladora del robot y después instalar a través de un *pendrive* el *plugging UrCap* (subsección 2.2.3) desarrollado por Robotiq para su control.

Como segunda parte de esta etapa se busca dotar al operario de un mayor control del sistema a través del mando del HMI. Para esto tendremos que conseguir extraer la información de los distintos botones del mando desde nuestro programa de Godot. Para ello utilizamos el *OpenXR Action Map*. Este hace referencia a un menú desplegable que se incluye dentro del paquete de Realidad Virtual de Godot (sección 2.2.2). Dentro de este menú podemos configurar el nombre de las acciones de cada uno de los botones y el tipo de información capturada por cada uno de ellos. En la figura 3.10 podemos ver el mapa de botones usado en esta etapa. Después podremos utilizar este nombre dentro de nuestro código para capturar los valores asociados a cada botón del mando en un determinado instante.

Una vez extraída esta información, implementé el respectivo comportamiento para cada una de las acciones, tanto en el código de la propia aplicación de Godot como en el *script* que domina los comportamientos en el lado del robot.

ax_button	A/X button	Bool	▼
by_button	B/Y button	Bool	▼
grip	Grip	Float	▼

Figura 3.10: Mapeo de botones en el menu OpenXR Action Map de Godot

### 3.5.3. Resultados

El prototipo resultante de esta etapa marca un sistema con altas capacidades de operación y amigable para el operario final. Además, gracias al mapeo de botones conseguimos dotar al usuario de un control mucho mayor, limitando al máximo

los posibles errores durante la ejecución. Las pruebas realizadas con este prototipo marcan como principales problemas a tratar la mejora de los elementos del entorno de visualización VR y la búsqueda de formas de reiniciar estados de parada del robot desde los mandos del usuario.

## 3.6. Etapa 5

En esta última etapa se busca terminar de pulir los detalles expuestos en los resultados en la etapa anterior. La idea será lograr un prototipo final fácil de manejar por un usuario inexperto y con la máxima libertad de operación posible. Además, se busca proporcionar una experiencia visual completa que facilite una mejor comprensión de la escena.

### 3.6.1. Objetivos de diseño

Los objetivos de esta etapa pueden ser divididos en 3 tareas principales:

- Añadir elementos de soporte al escenario VR.
- Añadir un modelo 3D del brazo robótico en el entorno.
- Buscar forma de reiniciar las paradas de emergencia del brazo robótico desde la aplicación de VR.

### 3.6.2. Diseño e implementación

Para que el usuario se sienta verdaderamente inmerso en el entorno del robot durante la operación, en esta primera tarea se propone añadir varios elementos que simulen ese entorno. En concreto, se plantean los siguientes:

- Área de trabajo del brazo robótico: En la subsección 1.1.1 explicamos la importancia de conocer este espacio durante la operación de brazos robóticos. En la escena de Realidad Virtual añadimos dos circunferencias. La circunferencia interior marca la zona en la que aparecen singularidades en las trayectorias debido a la cercanía de los elementos del robot y la exterior, los límites válidos que puede alcanzar. En la figura 3.11 (a) se pueden ver los elementos que marcan el área de

trabajo en la aplicación y en la figura 3.11 (b) el área de trabajo en la realidad.

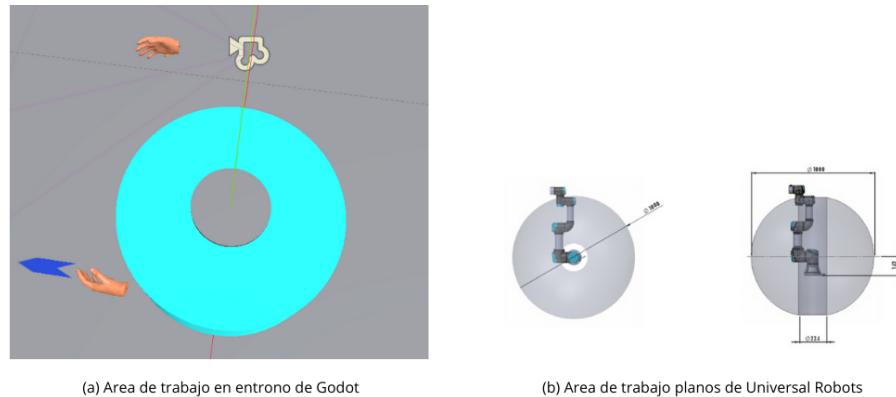


Figura 3.11: Comparación área de trabajo entorno VR y entorno real

- Mesa de operaciones: Simula la mesa en la que el robot se encuentra situado y configurado en la realidad. Su vista en el entorno VR se puede ver en la figura 3.12.

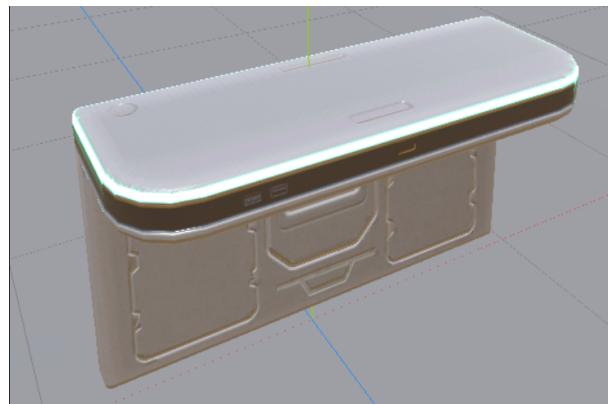


Figura 3.12: Mesa de operaciones en la aplicación de Realidad Virtual

- Límites de seguridad del robot: Los límites de seguridad establecidos durante la etapa (sección 3.4) se mostrarán en el entorno como planos infinitos semitransparentes.

Una vez modificado el entorno de esta forma, para conseguir de verdad un entorno de operación realista y funcional se plantea la integración de un modelo 3D del brazo robótico. Este modelo debe estar formado por las distintas articulaciones del robot

para así permitir su movimiento independiente. Además, utilizamos la información de posición que estamos recibiendo constantemente del módulo robótico para mover cada una de las articulaciones a su posición real. Con esto conseguimos un modelo 3D que muestra exactamente la situación del robot. El modelo utilizado puede descargarse en la página oficial de Universal Robots<sup>6</sup>.

También añadí el modelo del *gripper* instalado en la etapa anterior, el cual puede descargarse de la página oficial de Robotiq<sup>7</sup>.

Todo el modelo 3D ha sido modificado con una textura semitransparente para intentar que interrumpa lo menos posible la vista de la imagen de la cámara IP. En la figura 3.13 podemos ver este modelo con la transparencia aplicada en el entorno VR.

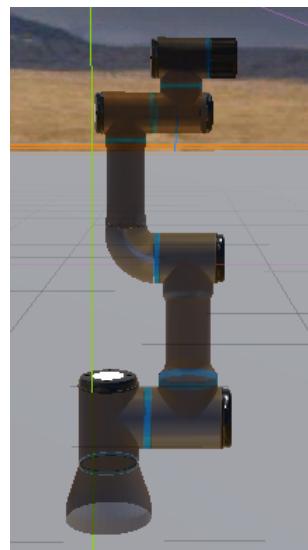


Figura 3.13: Modelo de brazo robótico UR3 en la aplicación de Realidad Virtual

Como ultima tarea de esta fase buscamos corregir otro de los problemas principales que limitan la autonomía del usuario, las paradas de emergencia. Estas paradas se dan en dos principales situaciones: cuando el usuario excede los límites de seguridad configurados o cuando el movimiento del robot le hace entrar en una posición de singularidad. Investigando sobre las diferentes formas de enviar señales de actuación al robot, encontré la *Dashboard communication*, explicada en la subsección 2.3.5. Para dotar al usuario de la capacidad de reiniciar el sistema desde el propio controlador configuramos uno de los gatillos del mando para que cree una comunicación de este

<sup>6</sup>UR Downloads: <https://www.universal-robots.com/download/>

<sup>7</sup>Robotiq Gripper: <https://robotiq.com/products/2f85-140-adaptive-robot-gripper>

tipo y envíe una secuencia de comandos encargados del reinicio.

### 3.6.3. Resultados

Esta última etapa marca el prototipo final del proyecto. Este proyecto podría seguir desarrollándose y consiguiendo aún más funcionalidad. Todo el código puede encontrarse en el repositorio de GitHub del proyecto indicado en la sección 1.5.

---

# **Capítulo 4**

# **Resultados**

---

En este capítulo explicaremos el diseño final del sistema. Por un lado, explicaremos su funcionamiento, uso y operación a nivel funcional (sección 4.1) y, por otro lado, a nivel técnico o de desarrollo (sección 4.2).

## **4.1. Descripción funcional**

Esta sección explica todo lo necesario para que una persona que decida operar el sistema pueda hacerlo. Recorre la configuración e inicialización de los dispositivos, lanzamiento de aplicaciones, comprensión del entorno y funcionalidad o uso.

### **4.1.1. Inicialización sistema robótico**

Para la inicialización del sistema robótico primero tenemos que tener en cuenta si queremos lanzar el sistema real o el simulado. En caso de querer lanzar el sistema de forma simulada, deberemos iniciar la máquina virtual que contiene la imagen de UrSim (subsección 2.2.3) y comprobar la IP de esta para poder indicárselo posteriormente a la aplicación de VR. En caso de estar utilizando el robot real este paso se realizará de forma automática.

Una vez realizado el paso anterior, ahora ambas configuraciones convergen en la misma secuencia de acciones. Cabe mencionar que esta secuencia de acciones solo será necesario realizarla en arranque manual:

Cada letra de la enumeración hace referencia a una de las pantallas mostradas en la figura 4.1.

- a Encendido de la controladora y Teach Pendant: se debe pulsar el botón que se observa en la imagen.
- b Desbloqueo de frenos e inicio de motores del brazo: se debe entrar en la pantalla de inicialización observada en la imagen y pulsar de forma secuencial en el cuadro de acción superior hasta que aparezca *Normal* como estado actual del robot.
- c Activación de *Gripper*: tocamos en el menú desplegable que aparece en la parte superior de la pantalla y pulsamos el botón *Activate*. El *gripper* se abrirá y cerrará para indicar el correcto funcionamiento.
- d Modo programa: desde la pantalla principal (a) pulsamos sobre la opción *Program Robot*.
- e Selector de programas: desde la pantalla anterior (d), pulsamos en *Load Program* y pasamos a la pantalla de selección. En esta seleccionamos el programa. En nuestro caso *VR\_rtde\_servoj\_loop.urp*.
- f Lanzamiento de programa: pulsamos el botón marcado con el círculo rojo para ejecutar el *script*.
- g Movimiento a posición de inicio: para que el programa comience el robot deberá desplazarse a la posición de inicio asignada. Al realizar el paso anterior automáticamente serás redirigido a esta pantalla. Para colocar el robot simplemente deberás mantener pulsado el botón *Auto* y el robot realizará el movimiento de forma autónoma.

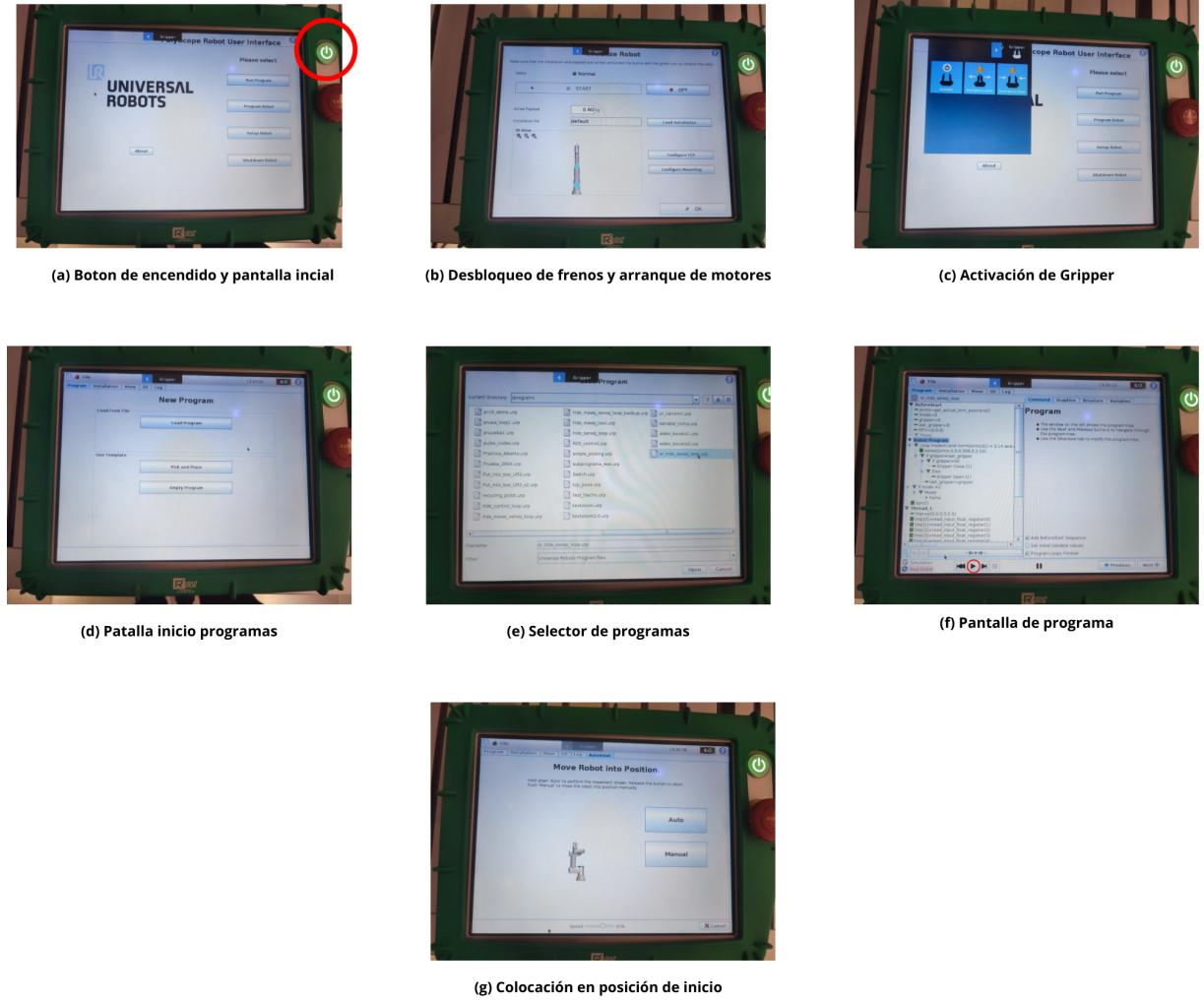


Figura 4.1: Etapas de lanzamiento de sistema robótico manualmente desde Teach Pendant

#### 4.1.2. Inicialización entorno VR

Para iniciar el sistema de Realidad Virtual lo primero que deberemos hacer será conectar nuestro visor VR al ordenador que se encargará del cómputo de la escena. En caso de usar las Oculus Quest 2 la conexión entre ambas se realiza a través del programa *Oculus Developer*<sup>1</sup>. Una vez el ordenador ha reconocido las gafas deberemos buscar dentro de ellas el modo *Quest Link* (figura 4.2) y ejecutarlo. Con esto ya estarán conectadas correctamente y podemos lanzar la aplicación.

<sup>1</sup>Oculus Developer: <https://developer.oculus.com/downloads/>



Figura 4.2: Pantalla de configuración de Meta Quest 2

La aplicación de Realidad Virtual se encuentra compilada en un archivo .exe (figura 4.3), es decir, un archivo ejecutable de Windows. Lanzamos este ejecutable y automáticamente entraremos en el entorno de Realidad Virtual y se establecerá una comunicación con el sistema robótico. Si queremos estar seguros de que la conexión ha sido efectiva es recomendable mover el robot real manualmente y ver si su posición se actualiza en el entorno real. En la figura 4.4 podemos ver como debería verse el robot en la posición de *home*.

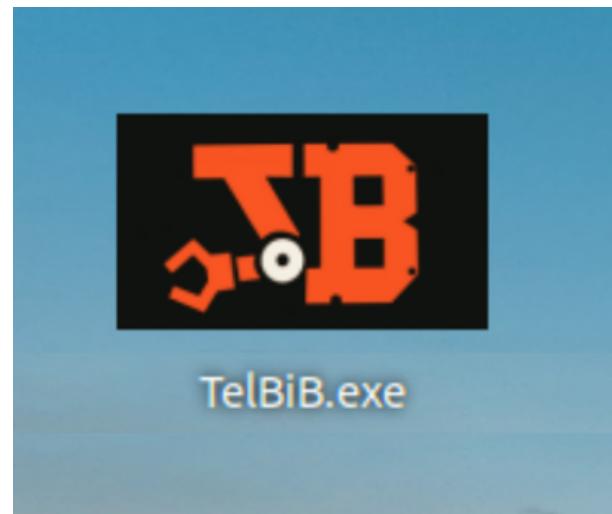


Figura 4.3: Ejecutable de la aplicación (.exe)



Figura 4.4: Robot en posición de inicio, vista desde entorno virtual

#### 4.1.3. Descripción de la escena virtual

La escena virtual incluye numerosos elementos pensados para mejorar la interacción del usuario con ella y la comprensión de la situación del sistema robótico en el entorno real. Estos elementos se puede dividir en dos clases:

##### Elementos Pasivos

Con elementos pasivos nos referimos a elementos que no varían su estado durante la interacción:

- Área de trabajo (figura 4.5 (a)): marca la zona en la que el brazo robótico puede operar sin detenerse por encontrar singularidades en su trayectoria.
- Mesa de operaciones (figura 4.5 (b)): se corresponde con la mesa en la que está montado el real. Esta tiene el objetivo de que la escena virtual se asemeje más a la real y de hacer entender al usuario donde está el límite inferior de movimiento.
- Planos de seguridad (figura 4.5 (c)): marcan las zonas que no puede exceder la punta del robot por seguridad. En caso de exceder una de estas el robot se detendrá.

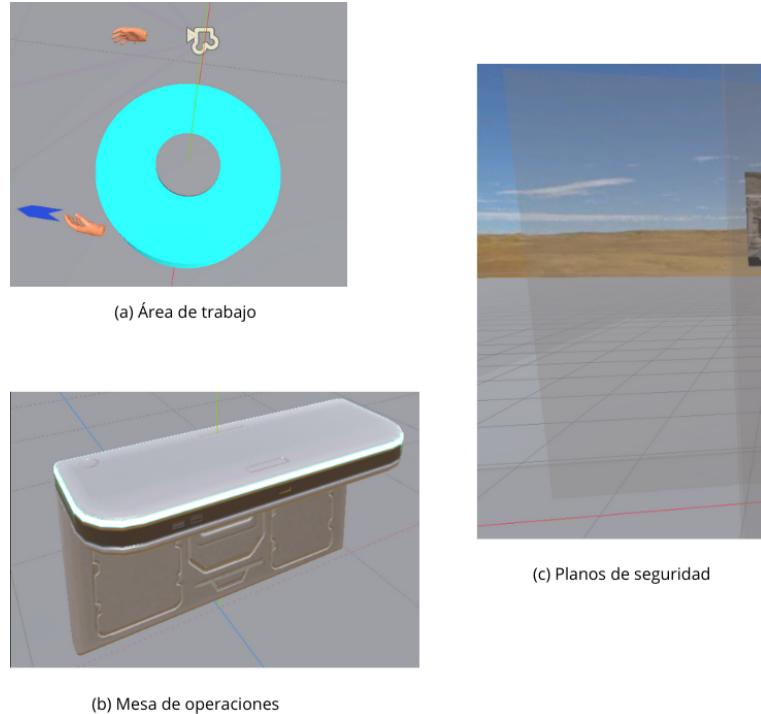


Figura 4.5: Elementos pasivos del entorno VR

### Elementos Activos

Con elementos activos nos referimos a aquellos que varían su estado durante la interacción. Este cambio puede ocurrir tanto por modificaciones en el entorno real como en el entorno virtual.:

- Cono TCP (figura 4.6 (a)): elemento con forma cónica que toma información de la posición y rotación del elemento terminal del brazo robótico y la muestra en el entorno VR. Utilizado principalmente para depuración y ajustes. Puede ser desactivado.
- Proyección cámara IP (figura 4.6 (b)): este elemento muestra la información recogida de una cámara colocada en el entorno real. Gracias a esta podemos conocer el resto de elementos reales durante las labores de operación.
- Modelo animado del robot (figura 4.6 (c)): este se podría considerar como el elemento activo más importante. Es un modelo 3D del robot real con 6 articulaciones que pueden ser rotadas de la misma forma que el robot real. Este lee la información de cada uno de los ejes del robot real y actualiza las suyas en el entorno virtual. Esto le permite al usuario conocer en todo momento la situación

del robot y de las trayectorias que está siguiendo en un formato tridimensional.

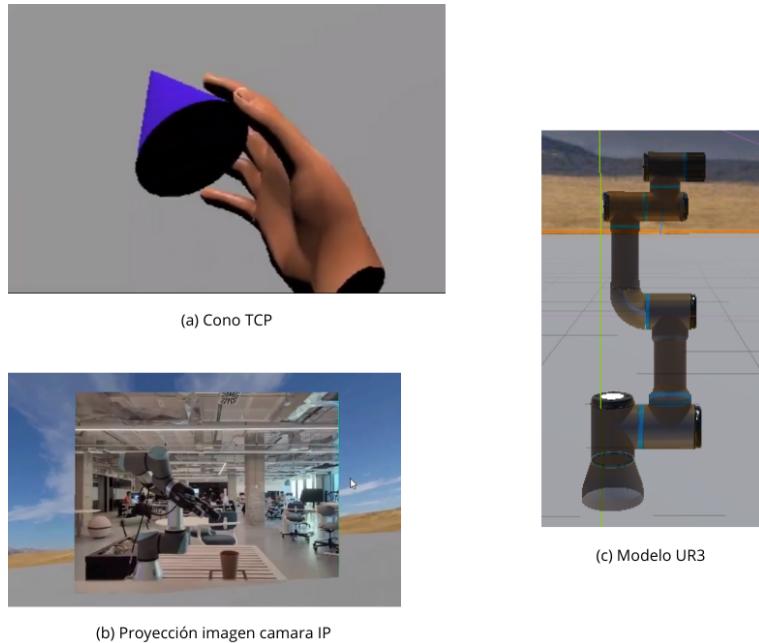


Figura 4.6: Elementos activos del entorno VR

A continuación en la figura 4.7 podemos ver la comparación entre el entorno virtual comentado en esta sección y el entorno real en el que se encuentra el robot.

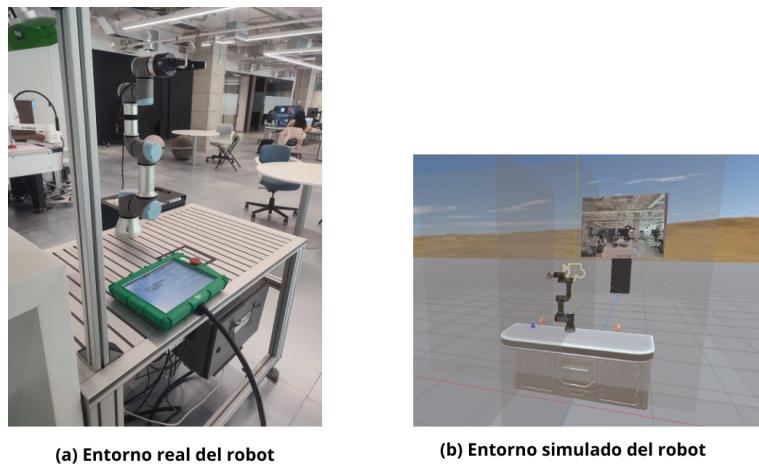


Figura 4.7: Comparación entre entorno robot real y escena VR

#### 4.1.4. Uso y funcionalidades

Para permitir al usuario completar operaciones complejas con la mínima posibilidad de fallo, el sistema dota a este de varias funcionalidades que detallaremos a

continuación. Todas estas funcionalidades se activan una vez se ha iniciado la aplicación de Realidad Virtual y se controlan a través del controlador derecho del Visor VR.

Como comentamos en la subsección 3.5.2. El mando de las Meta Quest 2 está formado por varios botones que pueden ser mapeados en Godot para utilizarlos en nuestra aplicación. En concreto, este modelo se compone de tres botones (a, b y home), dos gatillos (trigger y grip) y un *joystick*.

En la figura 4.8 podemos ver cada uno de los botones que componen el mando que acabamos de mencionar. En esta figura se asigna una letra a cada uno de estos. Esta letra la utilizaremos durante esta subsección para referirnos a cada uno de ellos.



Figura 4.8: Configuración botones mando Oculus Quest 2

### Movimiento por la escena

Para poder desplazarse por la escena se llevarán a cabo movimientos de teletransporte. Para realizar estos movimientos usaremos el joystick (c) con el que apuntaremos hacia la posición a la que deseemos desplazarnos.

### Movimiento del robot

Para poder mover el robot existen dos tipos de movimientos:

- Movimiento común (botón (a)): cuando coloquemos nuestra mano en una determinada posición y rotación donde deseamos que se mueva el TCP del robot, deberemos presionar el botón (a). Cuando pulsemos este, el sistema comunicará

esa información al robot y en caso de ser un punto dentro del área de trabajo del robot este se moverá a él.

- Retorno a posición de inicio o *home* (botón (b)): si el robot se ha detenido, ha llegado a una zona complicada o deseamos comenzar de nuevo el movimiento, se recomienda el uso del botón (b). Este botón habilita la funcionalidad de retorno, la cual indica al sistema robótico que tiene que realizar un movimiento en el que retome su posición de partida.

### Control de garra (*gripper*)

Para manejar el elemento terminal del robot se utiliza el gatillo (e). Cuando este se encuentre presionado a más del 50% este procederá a cerrarse y en caso contrario procederá a abrirse.

### Reinicio de sistema robótico

En caso de que el robot se detenga por alguna violación de sus límites de seguridad o singularidades, se recomienda mantener pulsado el gatillo (d). Este gatillo enviará una señal al sistema indicándole que debe reiniciarse para continuar con la operación. Es altamente recomendando utilizar el botón (b) de regreso a posición de inicio después de cada uso.

## 4.2. Descripción técnica

Esta sección describe la estructura y desarrollo de cada uno de los elementos que componen el sistema desde un punto de vista técnico. Este permite explicar el funcionamiento detallado de cada uno de los componentes para usuarios experimentados en estas tecnologías y desarrolladores que deseen utilizar este sistema.

La figura 4.9 muestra un esquema del modelo completo. Por simplicidad este modelo muestra una comunicación local (WIFI) con el entorno virtual y obvia algunos detalles específicos que comentaremos más adelante. A partir de este modelo, a continuación iremos desglosando cada uno de los módulos que lo componen y explicando su implementación al detalle.

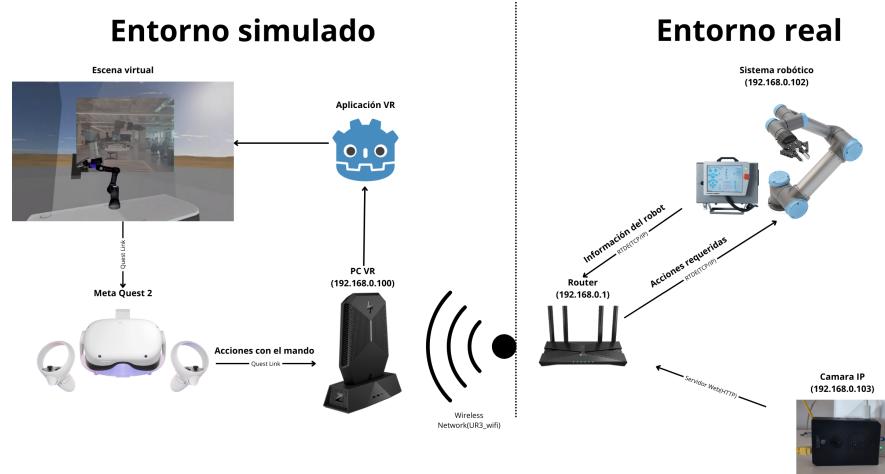


Figura 4.9: Esquema sistema comunicaciones completo simplificado

#### 4.2.1. Entorno Real

Consideramos como entorno real a la parte del sistema que se encuentra en el área de operación. Este está formado por tres componentes principales: el sistema robótico UR3 (brazo y controladora), la cámara IP y el router que hace de túnel entre este entorno y el simulado.

A continuación explicaremos en detalle la estructura de cada elemento, su funcionamiento y su implementación:

## Sistema robótico

Como ya hemos comentado en secciones anteriores como en 2.2 y 2.4 este sistema contiene muchas herramientas que nos permiten trabajar fácilmente con él.

Una de las herramientas más potentes que utilizaremos es el protocolo de comunicación RTDE. Este permite a aplicaciones externas escribir en una serie de registros del sistema. Los registros de lectura utilizados en esta aplicación se detallan en la tabla 4.1.

Registro	Descripción
input_double_register_0	Posición X
input_double_register_1	Posición Y
input_double_register_2	Posición Z
input_double_register_3	Rotación X
input_double_register_4	Rotación Y
input_double_register_5	Rotación Z
input_integer_register_25	Modo de operación
input_integer_register_26	Estado Gripper

Cuadro 4.1: Registros escritura utilizados en comunicación RTDE

En resumen, podríamos decir que el robot recibe por estos registros la posición y rotación deseada en cada instante, el modo de operación y el estado deseado para el *gripper*.

El formato en el que estos datos son recogidos tiene ciertas particularidades para cada caso y algunos necesitarán transformarse para poder ser comprendidos por las funciones de movimiento. En concreto la posición X, Y y Z deseada es previamente transformada en la aplicación de Realidad Virtual por lo que no necesita ninguna modificación. En cambio, las rotaciones son recogidas y enviadas en formato *RPY* (*roll*, *pitch*, *yaw*) mientras el robot utiliza el formato conocido como vector de rotación. Esto nos genera la necesidad de realizar un cambio de formato para el que aprovechamos la función *rpy2rotvec()* de URscript. La diferencia entre ambos formatos se puede ver en la figura 4.10.

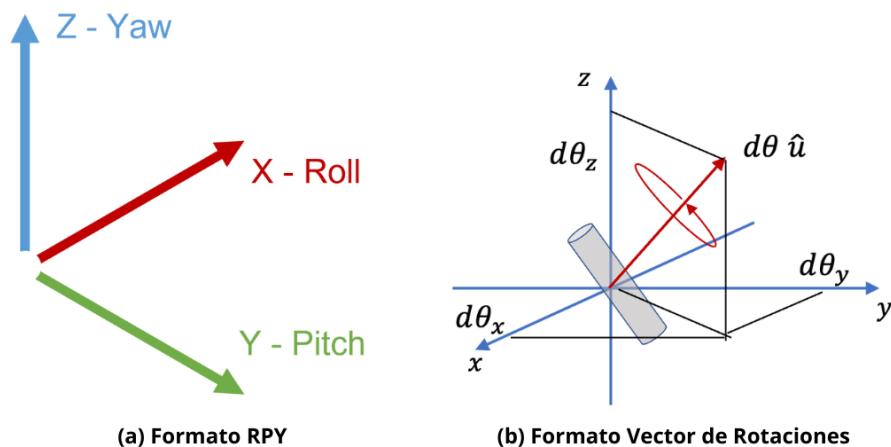


Figura 4.10: Formato de rotaciones RPY vs Vector de Rotaciones.

En RPY se representa la rotación como tres ángulos secuenciales alrededor de los ejes fijos del sistema de coordenadas y en el vector de rotación se utiliza un vector para representar el eje de rotación y un ángulo para la magnitud de esta.

Por otro lado, el modo de operación se envía como un número entero y marca en cuál de los tres posibles estados de movimiento se encuentra el robot. Si  $mode=1$  el robot se encuentra en el estado normal de operación, es decir, siguiendo el movimiento del operario. En cambio, si  $mode=2$ , el robot se encontrará en el modo de retorno a la posición inicial. Por último si  $mode$  almacena cualquier otro valor, el robot se encontrará en el estado detenido. En este proyecto se utiliza el valor de cero como modo *Stop* y cualquier otro valor fuera de estos como *Error*.

El modo que marca el estado del gripper del robot también es enviado como un número entero. Este valor está comprendido entre cero y cien. Este rango está pensado para poder controlar la apertura exacta del *gripper* suponiendo qué cero es totalmente abierto y cien totalmente cerrado. En nuestro caso, el *software* controlador del *gripper* no permite este tipo de control así que se establece que si este valor supera el 50 se considera como cerrado y si es inferior como abierto.

Para entender como se transforman estos valores y modos a movimiento del robot en la figura 4.11 podemos ver un esquema del flujo de ejecución del *script* que controla el robot.

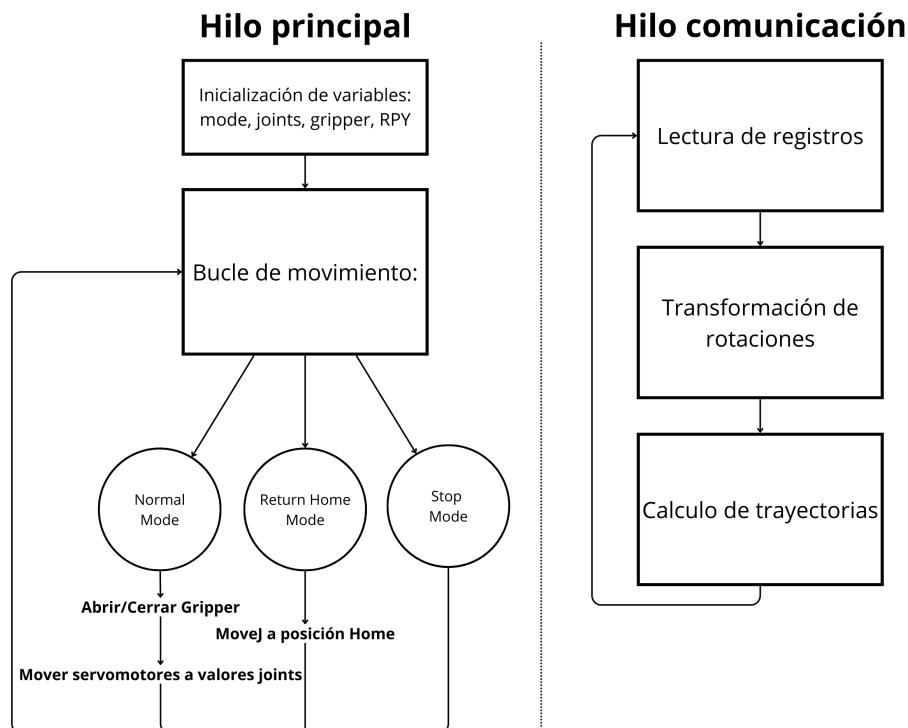


Figura 4.11: Esquema de funcionamiento de *script* de control del sistema robótico

Como podemos ver en la figura anterior, el esquema está dividido en dos hilos (*threads*) de ejecución:

- Hilo de comunicación: este hilo desempeña su labor en tres etapas. En la primera etapa este será el responsable de leer los valores actualizados por el canal de comunicación que se encuentran en los registros del robot. Una vez con estos valores se encargará de cambiar el formato de las rotaciones recibidas como comentamos anteriormente. Por último este comprueba la viabilidad de alcanzar la nueva posición y rotación requeridas y calcula la posición en la que se deberán colocar cada uno de los servomotores para alcanzarla.
- Hilo principal: este hilo será el encargado de dominar la actuación del robot y del *gripper*. Su funcionamiento se puede dividir en dos etapas. La primera etapa solo se ejecutará una vez al iniciar el programa y se encarga de inicializar las variables necesarias y de que el robot empiece el movimiento desde la posición de inicio. La segunda etapa se puede considerar como un bucle de control en el que el robot irá realizando los comportamientos correspondientes en función a los valores que se vayan actualizando en el bucle de control. Cabe destacar que como se mencionó en la subsección 3.1.2 la función de movimiento utilizada para el modo de operación *Normal* es *servoJ*. Ya que esta función permite un movimiento no bloqueante con el que ajustando sus valores podemos conseguir un movimiento fluido y a una velocidad de 125Hz.

## Cámara IP

Como mencionamos previamente en subsección 2.4.6 la solución elegida para que la cámara IP publicase imágenes del entorno real fue utilizar una Raspberry Pi con un módulo RaspiCam y programarlo según nuestras necesidades. Con esto conseguimos un mayor control sobre la transmisión y la posibilidad de en el futuro añadir otros módulos para expandir sus funcionalidades.

En concreto, se ha utilizado una Raspberry Pi 4 modelo B y una Raspicam V2.1. En la figura 4.12 (a) se puede ver la conexión entre ambas. Además, se ha instalado una carcasa pensada para proteger ambos dispositivos (figura 4.12 (b)).



(a) Montaje para cámara IP

(b) Cámara IP con caracasa

Figura 4.12: Conexión Raspberry Pi 4 y Raspicam

La idea de esta cámara IP es que sea capaz de publicar las imágenes capturadas en un servidor web para que la aplicación de Realidad Virtual pueda realizar peticiones HTTP y leerlas. Para ello se plantea un *script* con dos hilos de ejecución:

- Hilo captura de imágenes: el primer hilo se encargará de utilizar la librería *libcamera* para comunicarse con el dispositivo y poder capturar imágenes de forma continua. Después de capturar esta imagen la almacena para que pueda ser accesible por el segundo hilo del *script*.
- Hilo servidor WEB: este hilo se encarga de controlar las peticiones al servidor WEB. De tal forma que si recibe un GET a la dirección asignada, este se encargará de leer la última imagen captura y devolvérsela al solicitante a través del protocolo HTTP.

Además, este *script* ha sido configurado como un servicio en el propio sistema Linux que se encuentra corriendo en la placa. Esto permite que cada vez que se inicie el dispositivo, el sistema operativo lance el *script* de forma automática, evitando tener que interactuar manualmente con él.

## Router

Como ya explicamos en la subsección 2.4.5 el router TP-Link Archer AX23 es el que se usará en nuestro sistema como puente entre el Entorno Real y el Entorno Simulado. Este dispositivo se encuentra en el lado del Entorno Real y tanto el sistema robótico,

como la Raspberry que actúa como cámara IP, irán conectados a él a través de un cable Ethernet. Este además genera una señal WIFI (*UR3\_wifi*) que permite la conexión de otros dispositivos. En nuestro caso se ha utilizado esta señal para conectar los elementos del Entorno Simulado de forma inalámbrica, pero manteniendo una baja latencia que nos permitiese realizar pruebas de forma más sencilla.

La dirección local IPv4 de este router es 192.168.0.1. Para evitar tener que comprobar las direcciones asociadas a cada uno de los elementos del sistema en cada uno de los arranques, se han configurado una serie de direcciones estáticas que se pueden ver en la tabla 4.2. Aun así, este sistema podría funcionar con cualquiera de sus componentes en otras direcciones.

Elemento	IP estática asociada
Ordenador Entorno Simulado (HP)	192.168.0.100
Sistema robótico	192.168.0.102
Cámara IP (Raspberry Pi)	192.168.0.193

Cuadro 4.2: IPs estáticas configuradas

#### 4.2.2. Entorno Simulado

Consideramos como entorno simulado, a la parte del sistema que se encuentra en el lado del operario. Es decir distanciado en cierta medida del área de operación. Este está formado por dos elementos *hardware*: el ordenador de procesamiento y el visor de Realidad Virtual con sus respectivos mandos. Además, podemos considerar como elementos software de importancia a la aplicación de Realidad Virtual y a la escena virtual proyectada en el visor.

A continuación explicaremos en detalle el funcionamiento de cada uno de estos elementos y el desarrollo realizado sobre ellos:

##### Ordenador Principal

Como ya explicamos en la sección 2.4 debido a las limitaciones del procesador de las Meta Quest 2, se utilizará un ordenador externo para el procesamiento y la comunicación necesaria dentro del entorno simulado. Por lo tanto, la aplicación de Realidad Virtual ejecutará dentro de este y se comunicará con el visor a través del

protocolo *Quest link* que comentaremos cuando hablemos del visor más adelante.

La aplicación de Realidad Virtual, desarrollada con el motor de videojuegos Godot encapsula todo el comportamiento del entorno simulado. Para esto, como ya mencionamos en (subsección 2.2.2) este motor de juegos sigue un enfoque de programación en árbol de nodos. Cada uno de estos nodos tiene unas características especiales y va asociado a un *script* que define su comportamiento. Para simplificar lo máximo posible la explicación de cada uno de los nodos que forman la escena se han dividido estos en 3 grupos:



Figura 4.13: Tipos de nodos en aplicación Godot

### Nodos estáticos

Consideramos como nodos estáticos a los que no tienen un *script* asociado a ellos que controle su comportamiento. Estos serán principalmente elementos que forman parte de la escena y ayudan a la inmersión como el fondo o la mesa de operaciones.

En la figura 4.14 podemos ver cada grupo de nodos estáticos que componen la aplicación. En la siguiente enumeración iremos explicando con más detalle el tipo y lo que representa cada uno de ellos. Las explicaciones a nivel funcional de algunos de estos pueden encontrarse en 4.1. Cada letra de la enumeración hace referencia a cada uno de los grupos de la imagen:

a Planos de seguridad: Cuatro nodos de tipo MeshInstance3D. Dentro de esta

clase se definen con una geometría de plano y se les dota de una textura semi-transparente para no molestar la visión del usuario.

- b Área de trabajo: Tres nodos de tipo MeshInstance3D. Uno de ellos se define con geometría cuadrada y marca el área de *spawn* del jugador virtual que representa al usuario. Los otros dos se definen con una geometría cilíndrica y marcan el área de trabajo del brazo.
- c Mesa de operaciones: Un nodo de tipo Node3D. Este nodo contiene un objeto 3D en formato *.obj*.
- d Suelo: Un nodo StaticBody3D como nodo principal y un nodo MeshInstance3D y CollisionShape3D como nodos hijos de este. El primero de ellos representa el apartado visual del suelo y el otro marca la colisión con el que permiten que el jugador no lo atraviese.
- e Luces y Cámara: Varios nodos que definen la dirección y posición de la luz y el punto de inicio y *offset* de la cámara del usuario.

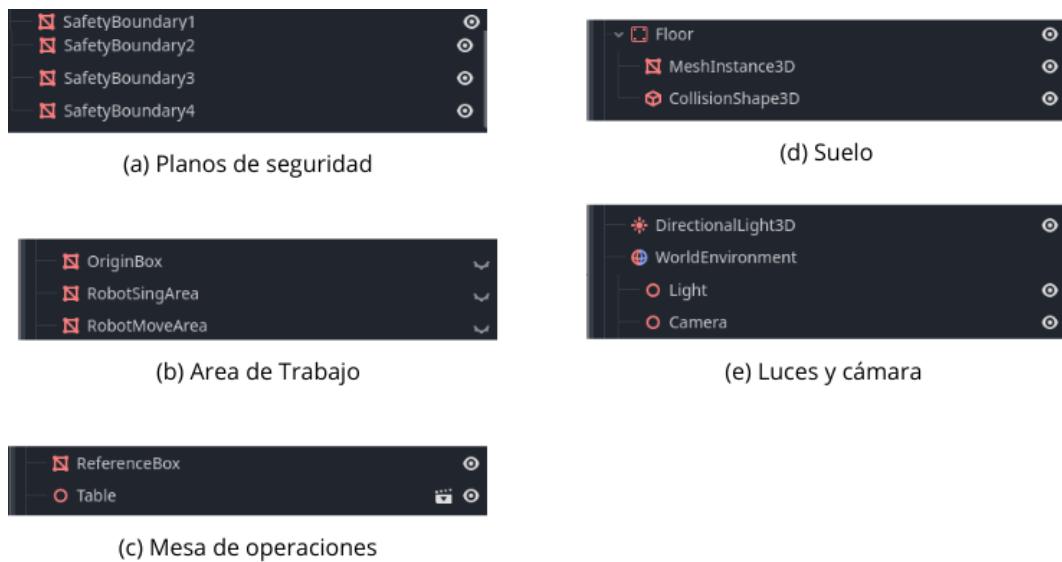


Figura 4.14: Nodos estáticos del árbol de nodos de la aplicación VR

### Nodos dinámicos

Consideramos como nodos dinámicos a los que tienen un *script* asociado a ellos que permite ir modificando su comportamiento a lo largo de la ejecución. Dentro de estos se pueden diferenciar dos grupos. Por un lado, tendríamos los de visualización,

que están pensados con la idea de ayudar al usuario a entender la escena real durante la operación. Y, por otro lado, los de interacción virtual, estos permiten al usuario desplazarse por la escena, interactuando con los diferentes elementos, además de captar datos que posteriormente resulten de intereses para los nodos de actuación.

Dentro de los nodos dinámicos de visualización en esta aplicación existen un total de dos grupos que se pueden ver en la figura 4.15. En la siguiente enumeración iremos explicando la implementación y composición de cada uno de ellos. Lo que representan estos para el usuario fue explicado en la sección 4.1. Cada letra de la enumeración hace referencia a cada uno de los grupos de la imagen:

- a Proyección de cámara IP: este grupo está formado por 3 nodos. Como nodo padre tenemos un Node3D el cual contiene el *script* que marca el funcionamiento de este en ejecución. Este nodo padre a su vez tiene 2 nodos hijo. Por un lado, tiene un nodo HTTPRequest que es el que habilita al nodo padre a obtener las imágenes de la cámara a través de este protocolo y, por otro lado, tiene un nodo Sprite3D que representa el plano donde se proyectarán las imágenes obtenidas. En el *snippet* 4.1, se puede ver como se asigna la imagen recibida al nodo Sprite3D. Cabe mencionar también que como se observa en la figura 4.15 (a) hay un nodo padre de un nivel superior al Node3D (RobotCamera). Este nodo superior representa la vista del usuario y gracias a esta conexión entre ellos permite que la pantalla que proyecta la imagen se mueva siempre hacia la zona a la que está mirando el operario.

---

```
func _http_request_completed(result, response_code, headers, body):

    if response_code == 200:
        #-- Load new image
        var image = Image.new()
        var image_error = image.load_jpg_from_buffer(body)
        if (image_error != OK and DEBUG):
            print("An error occurred while trying to display the image.")

        #-- Transform image to texture and assign to the 3D
        $RobotCameraWindow.texture = ImageTexture.create_from_image(image)
```

---

Fragmento de código 4.1: Función recepción imagen por protocolo HTTP y carga en Sprite3D

- b Modelo 3D del brazo robótico: Este grupo como podemos observar está formado

por un gran conjunto de nodos. El nodo padre UR3 Animated al igual que en el grupo anterior, es el que contiene el *script* que controla el comportamiento de los nodos hijos. Este lee los valores de la posición angular de cada uno de los servomotores recibidos del robot real (a través de un nodo que explicaremos en el siguiente apartado) y modifica las rotaciones de los ejes correspondientes de los nodos MeshInstance3D que forman cada una de las articulaciones del robot. En el *snippet* 4.2 podemos ver una versión simplificada del código que actualiza las rotaciones de cada elemento del modelo 3D.

---

```
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):

    var ur3_current_joints = urbot_node.GetActualJoints()

    #-- Rotation of each joint of te robot
    $Yup2Zup/UnityRobotics_RF3_s1/UR3.rotation[1] = -ur3_current_joints[0] +
        BASE_ROT_OFFSET
    $Yup2Zup/UnityRobotics_RF3_s1/UR3/Shoulder.rotation[2] =
        ur3_current_joints[1] + SHOULDER_ROT_OFFSET
    $Yup2Zup/UnityRobotics_RF3_s1/UR3/Shoulder/Elbow.rotation[2] =
        ur3_current_joints[2]
    $Yup2Zup/UnityRobotics_RF3_s1/UR3/Shoulder/Elbow/Wrist01/Wrist02.rotation[2]
        = ur3_current_joints[3] + WRIST1_ROT_OFFSET
    $Yup2Zup/UnityRobotics_RF3_s1/UR3/Shoulder/Elbow/Wrist01/Wrist02/Wrist03.rotation[1]
        = -ur3_current_joints[4]
    $Yup2Zup/UnityRobotics_RF3_s1/UR3/Shoulder/Elbow/Wrist01/Wrist02/Wrist03/EffectorJoin
        = ur3_current_joints[5]
```

---

Fragmento de código 4.2: Función de recepción de valores de robot real y asignación modelo 3D.

Se utiliza la función *GetActualJoints()* del nodo *urbot* para leer la posición angular de los *joints* del robot y después modificarla en los respectivos ejes del modelo 3D, teniendo en cuenta el eje sobre el que cada uno esta montado.

c Cono de rotación: siguiendo una implementación muy similar a los dos grupos anteriores este está compuesto por un nodo padre que contiene el *script*, en este caso un MeshInstance3D que a su vez contiene a la geometría cónica de este. El tratar de usar este tipo como nodo padre en lugar de un Nodo3D como en los anteriores se realizó con la idea de experimentar con diferentes opciones de desarrollo. Por otro lado, este contiene otros dos nodos hijo y nieto que marcan las colisiones de este elemento.



Figura 4.15: Nodos de visualización dinámica del árbol de nodos de la aplicación VR

En lo que respecta a nodos dinámicos de interacción virtual, en esta aplicación existe un grupo de cumple con sus requisitos. Este grupo es el encargado de toda la interacción del usuario con el entorno. Se encargará tanto de permitir el movimiento de este por la escena, como de la captación de las pulsaciones de los botones y posición de las manos del jugador. En la figura 4.16 puede verse la composición de este grupo en el árbol de nodos de la aplicación.

Este grupo está dividido en dos partes, una para cada una de las manos del usuario. Cada una de estas manos contiene un XRNode3D que contiene el *script* que se encarga de actualizar el estado de este en la escena en función a los movimientos y pulsaciones del usuario y de almacenarla. Además, contienen dos nodos hijos. Un XRTToolsHand que se encarga de la visualización de esta mano en la escena y otro de tipo XRTToolsFunctionTeleport que activa la función de teletransporte que permite al usuario desplazarse por la escena.

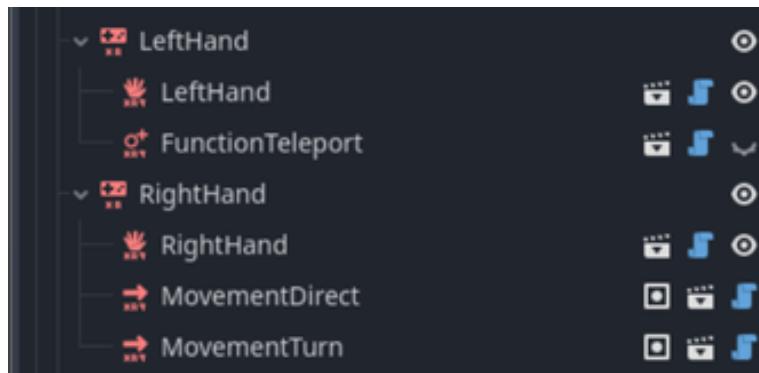


Figura 4.16: Nodo de interacción dinámica del árbol de nodos de la aplicación VR

## Nodos de actuación

Consideramos como nodos actuación a los que llevan a cabo modificaciones no solo sobre el entorno simulado, sino también sobre el entorno real. En este caso solo existe un nodo de este tipo y este encapsula un gran abanico de funcionalidades. Este nodo se define como *urbot* y está implementado en lenguaje C# (subsección 2.2.1) para permitir una comunicación más eficaz y una estructura de programación más completa. Se puede considerar como uno de los grandes elementos, junto con el URscript, que hace que funcione el sistema.

En lo que respecta a su definición en el árbol de nodos de Godot este es de tipo Node3D y está ligado a un script *urbot.cs*. En la figura 4.17 podemos ver la línea que representa a este dentro del árbol.



Figura 4.17: Nodo de actuación urbot

En la figura 4.19 podemos ver el flujo de funcionamiento de este nodo. A continuación se detallarán cada uno de los pasos dentro de este flujo.

- **Inicialización de variables:** se otorgan valores iniciales a las variables necesarias para la ejecución. Además, se crean los respectivos objetos de clases como las manos del usuario, que nos permitirá tomar información de este otro nodo.
- **Establecimiento de comunicación RTDE con robot:** creamos el respectivo *socket* que se comunicará con el robot real y arrancamos la conexión. Dentro de este nodo para poder utilizar toda la funcionalidad del protocolo RTDE se implementan todas las funciones necesarias. Entre estas funciones destacan algunas como las que nos permiten codificar y decodificar los paquetes que transfiere este protocolo o las que manejan los eventos de recepción de datos.

Estos últimos eventos se iniciarán en este paso e irán actualizando los datos que maneja el resto del programa cada vez que reciban nueva información.

- **Bucle de control:** una vez se entra en este paso se puede decir que el sistema se encuentra en marcha. A partir de aquí el programa irá continuamente comprobando las acciones llevadas a cabo por el usuario y actuando en consonancia. Los siguientes pasos que explicaremos se encontrarán dentro de este bucle.
- **Lectura de botones del mando del usuario:** dentro de este bucle lo primero será leer la situación de los botones del usuario para saber como debe actuar el sistema.
- **Botón Trigger pulsado:** este botón se encuentra en el mayor nivel de prioridad. Si este es pulsado se ignorarán el resto y se procederá a realizar el mecanismo de reinicio del sistema robótico. Para esto se establecerá una comunicación con el robot, pero en este caso a través del protocolo Dashboard. Este nodo también implementa funciones para realizar una comunicación utilizando este protocolo, pero su desarrollo fue considerablemente más simple debido a que no necesita codificar y decodificar información en paquetes, sino que solo permite el envío de una serie de cadenas de texto que actúan como comandos en el sistema robótico. Un ejemplo de comando enviado a través de este protocolo sería por ejemplo: `dashboardClient.SendString("unlock protective stop\n");` que se encarga de desbloquear el robot tras una parada de emergencia.
- **Botón A pulsado:** este botón se encuentra en el siguiente nivel de prioridad después de trigger. Si este es pulsado se estará ejecutando el modo de *operación normal*. Este modo envía al robot a la posición en la que se encuentra la mano del usuario y con la propia rotación de esta. Para esto, la primera acción que se toma en este caso es la extracción de las posiciones y rotaciones de la mano del usuario para posteriormente transformar esta posición a la comprendida en los ejes de coordenadas del robot. En (figura 4.18) se pueden ver los ejes de coordenadas de cada uno de los entornos. Una vez estos datos se encuentran preparados para el envío se procede a comprobar si el botón grip se encuentra pulsado. En caso afirmativo, también se enviará esta información en el paquete RTDE para que el robot abra o cierre la garra y se desplace a la nueva posición.

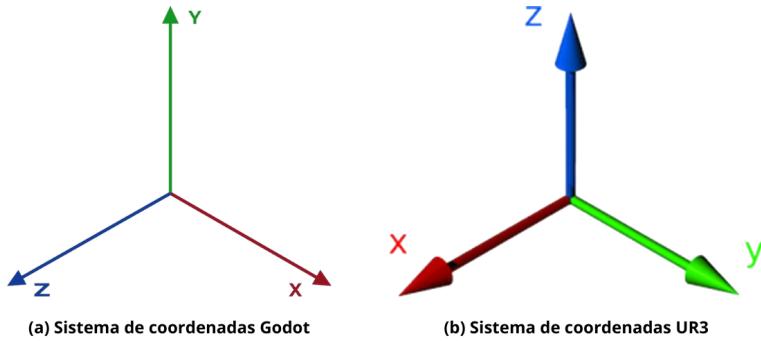


Figura 4.18: Comparación ejes de coordenadas Godot y UR3

- **Botón B pulsado:** Este botón se encuentra en el último nivel de prioridad. Es decir, solo se activa su funcionamiento si no se pulsan ni Trigger, ni A. Este será el encargado de inducir al sistema en modo retorno a la posición inicial. Para conseguirlo este completará el paquete RTDE con las posiciones de *home* y procederá a enviarlo de la misma forma que en la secuencia de pulsación del botón A.

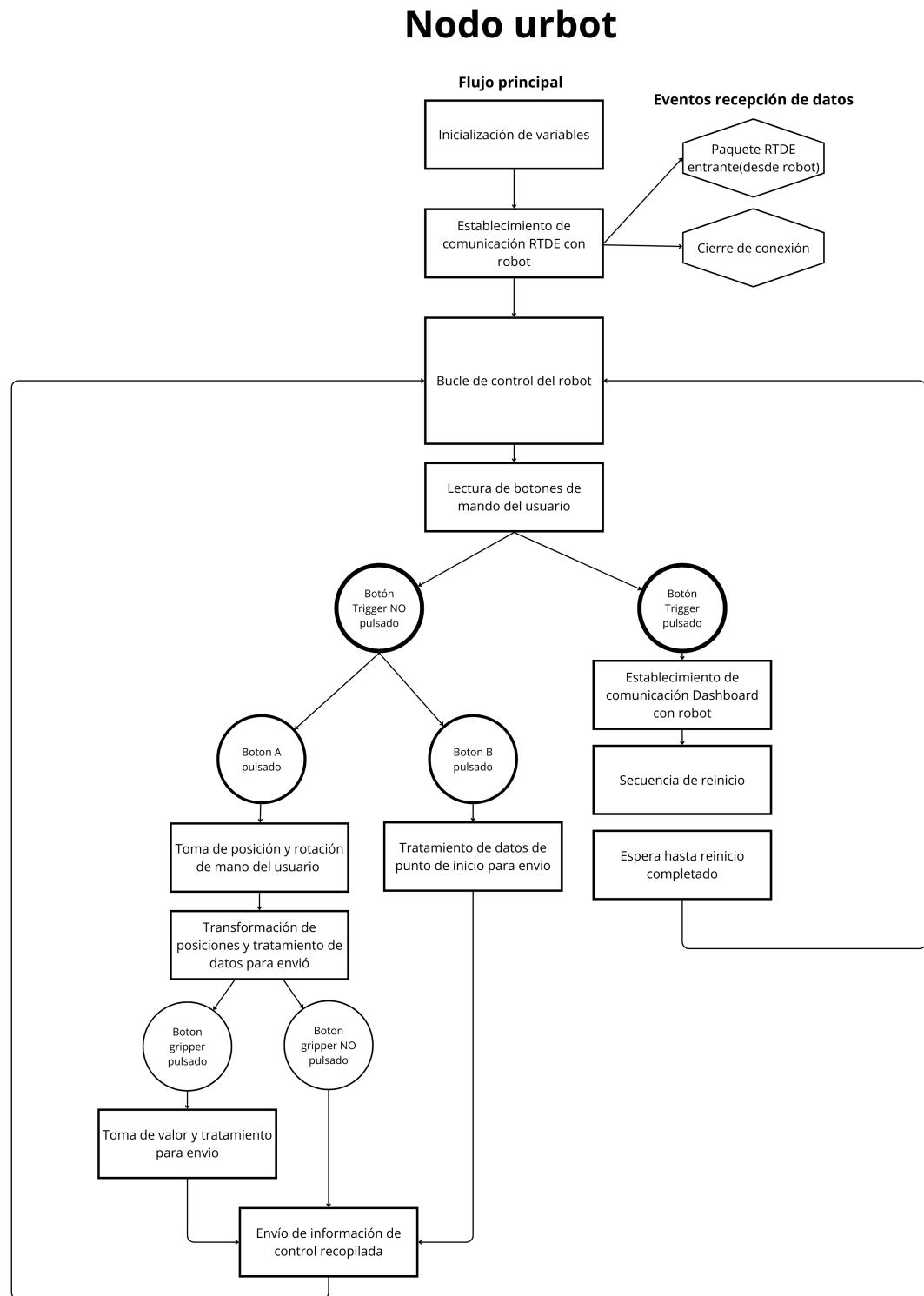


Figura 4.19: Flujo de control robot nodo urbot

### Visor Meta Quest 2

Como ya explicamos en la subsección 2.4.3 las Meta Quest 2 serán el dispositivo que permita al usuario visualizar el entorno simulado. Como todo el procesamiento

se realiza en el ordenador mencionado en el apartado anterior a nivel técnico este sistema no ha requerido ninguna implementación específica. Cabe resaltar, como ya mencionamos anteriormente, que la comunicación entre el ordenador y este visor se realiza a través del sistema *Quest Link* desarrollado por Meta. Esta comunicación se puede llevar a cabo tanto de forma inalámbrica, como por cable. Esta segunda opción es la utilizada en este proyecto debido a su mayor velocidad y fiabilidad.

---

# Capítulo 5

# Experimentación y Validación

---

En este capítulo se detallan los diferentes experimentos realizados para probar el sistema y los resultados de estos, comprobando si se ajusta a los objetivos planteados. Los experimentos realizados se pueden dividir en dos tipos que estudiaremos a continuación.

## 5.1. Experimentación técnica

El primer tipo de experimentación llevado a cabo es la desarrollada a nivel técnico. Esta tiene como objetivo evaluar las capacidades y límites del sistema de forma experimental. Además, gracias a esta podemos encontrar las mayores debilidades del sistema y plantear posibles mejoras futuras.

A continuación se enumeran cada una de las pruebas llevadas a cabo y sus resultados:

- Fluidez de trayectorias: uno de los elementos más importantes para conseguir una buena experiencia para el usuario y para no desgastar o dañar los componentes del robot es que sea capaz de realizar trayectorias limpias y controladas. Con esto se hace referencia a que el control del robot esté ajustado de forma que su movimiento no presente oscilaciones. Como se mencionó en la subsección 4.2.1, el robot utiliza el tipo de movimiento *ServoJ*. Este de por sí contiene un controlador proporcional que puede ser configurado por el desarrollador a través del parámetro *gain*. Además, existe otro parámetro que influye en el control de este denominado como *t* o *lookahead time*. Este segundo parámetro marca el tiempo que se bloquea el control una vez recibe una orden. Es decir, hasta que no termine ese tiempo no se podrá modificar la trayectoria que está siguiendo el

robot.

Para poder ajustar estos parámetros de forma correcta se han llevado a cabo varias pruebas hasta poder asegurar un buen comportamiento. Los resultados observados llevan a la conclusión de que el parámetro  $t$  debe encontrarse en el rango [0.008 - 0.014] siendo 0.008 el mínimo al que puede operar el robot, es decir con una tasa de cambio de trayectoria de 125Hz y 0.014 representa la máxima tasa de refresco a la que puede trabajar nuestro sistema de VR, es decir, 72Hz. Cualquier valor dentro de este rango permite trabajar con la máxima velocidad y fluidez permitida por nuestro *hardware*. Si estableciésemos un valor mayor el sistema empezaría a perder reactividad, lo que sería interpretado por el usuario como movimientos más torpes. Por otro lado, esta tasa tan alta genera en nuestro sistema la necesidad de un cálculo de posición y transformación preciso para evitar oscilaciones. Dentro de estas oscilaciones entra el segundo parámetro *gain*. Si el valor de este se encuentra en un rango alto, se genera un desplazamiento más rápido pero a costa de generar oscilaciones en el control. Nuestro movimiento se plantea como una sucesión de puntos continua, es decir, no deberían existir grandes cambios de posición entre iteraciones, debido a las limitaciones humanas. Es por esto que este parámetro se configura en un rango de valores bajo [100 - 300], consiguiendo un movimiento controlado y veloz que resulta en un control fluido para el operario.

- Seguridad: la seguridad es un factor sumamente importante tanto para los activos humanos, como los físicos. En nuestro caso, la seguridad de las personas que puedan encontrarse cerca del área de actuación del robot está solventada gracias al uso de un *cobot*, como explicamos en la subsección 1.1.1. Aun así, para asegurar una respuesta de calidad a posibles riesgos durante la operación, se han realizado varias pruebas simulando posibles choques del brazo con elementos dinámicos del entorno. En todas ellas, el robot ha respondido con la suficiente velocidad para asegurar que es seguro de operar en entornos transitados por humanos. En lo que respecta a la seguridad del propio sistema robótico se ha utilizado una configuración de planos de seguridad como se detalló en la subsección 3.4.2. Después de varias pruebas estos han asegurado que evitan de forma anticipada cualquier posible impacto del robot con los elementos estáticos de su entorno.

- Velocidad de cómputo: como mencionamos en las pruebas de control de trayectorias, el conseguir que el sistema responda de forma fluida es fundamental para la experiencia de usuario. Otro factor limitante que puede ralentizar estos movimientos es la velocidad de cómputo del sistema de Realidad Virtual. Como hemos mencionado, este tiene la capacidad de funcionar a 72Hz, pero se puede ver reducido por cálculos muy costosos o malas prácticas de programación. En concreto, durante estas pruebas se encontraron esperas activas en el programa de control que generaban grandes ralentizaciones, llevando a que cada secuencia tardase 0.5 segundos (2 Hz). Con esta frecuencia el sistema respondía de manera prácticamente incontrolable y muy poco precisa. Es por esto que se decidió el uso de eventos e hilos (*threads*) para la actualización de datos y así no tener que realizar espera activa en nuestro bucle principal. Una vez realizados estos cambios, los resultados cambiaron drásticamente, consiguiendo tasas de 70Hz. En la figura 5.1, podemos ver los tiempos de ejecución de cada iteración del bucle de control.



Figura 5.1: Velocidades de funcionamiento de sistema capturadas por las herramientas de depuración de Godot.

En las gráficas superiores podemos ver como cada una de las iteraciones ronda los 14 ms, lo que llevan a tener una tasa de refresco de aproximadamente 72 FPS, como puede observarse en la gráfica superior izquierda. En la figura inferior podemos ver la consistencia de estos valores en el tiempo.

## 5.2. Experimentación con usuarios

Este segundo tipo de experimento busca la validación por parte de los usuarios del sistema. Es decir, poder comprobar como es la experiencia de un usuario sin conocimientos previos tratando con el sistema. En concreto, en este proceso han participado dos usuarios completamente externos al desarrollo y sin ninguna noción previa en el uso de estos sistemas.

Durante las pruebas con ambos se ha seguido la misma metodología y evaluación. En concreto se ha empezado explicando brevemente, alrededor de 5 minutos, la composición del entorno simulado, el control del sistema y algunas consideraciones para evitar paradas o movimientos bruscos del sistema. Es decir, se han comentado cada uno de los puntos de la sección 4.1 y una breve introducción a lo que busca el proyecto. Después de esta explicación se le otorga al usuario el control del sistema sin un objetivo predefinido para que pueda acostumbrarse a los movimientos. Pasados unos minutos probando cada una de las funcionalidades se le propone al usuario una misión de *Pick & Place* en la que debe tomar un vaso que se encuentra en la mesa sobre la que se encuentra el robot y depositarlo en la papelera que se encuentra detrás de este.

La evaluación de los resultados obtenidos en función a la observación y a los comentarios realizados por los usuarios se ha dividido en una serie de puntos:

- Comprensión de escena: según los usuarios, la comprensión de los diferentes elementos de la escena resulta bastante orientativa. Destacan la utilidad del modelo 3D especialmente una vez se acostumbra a los movimientos. Es decir, ambos indican que una vez operando el sistema durante unos minutos solo utilizan la imagen de la cámara para saber donde se encuentran los elementos sobre los que tienen que actuar, pero que una vez reconocidos en la imagen utilizan la mesa virtual y el modelo para alcanzarlos. Destacan que la cámara tiene ciertas limitaciones visuales, debidas principalmente al solapamiento por el resto de elementos (mesa, modelo 3D, suelo...) y la bidimensionalidad. Por otro lado, destacan que el añadir algún elemento más a la mesa que actúen como referencias de posición podrían ser de utilidad para ser conscientes de la posición real.
- Comprensión de controles: sobre la comprensión y uso de los botones, los usuarios

destacan una distribución bastante orientativa y uso sencillo. En concreto indican que el botón de retorno al punto de inicio resulta de mucha utilidad especialmente al comienzo para entender los movimientos del brazo apoyándose en el modelo 3D. Por otro lado, valoran negativamente que el movimiento se detenga mientras se abre y cierra la garra, ya que el sistema pierde fluidez. Las observaciones realizadas muestran que después de unos minutos el usuario identifica bien el funcionamiento de cada uno de los botones.

- Familiarización con movimientos: ambos usuarios coinciden en que el movimiento del sistema robótico resulta muy reactivo y fácil de comprender. Destacan que no se detecta ningún retardo en el movimiento y que las trayectorias son limpias y claras. Por otra parte, indican que el primer movimiento después de haber parado puede resultar un poco brusco en función a la cercanía del mando con el último punto y la orientación de este. Durante la observación se aprecia que existen puntos en los que el cálculo de trayectoria indica unas posiciones para cada uno de los ejes que pueden resultar complicadas para el resto de movimientos de la trayectoria. Esto puede provocar cambios de posición rápidos que podrían llegar a detener el sistema.
- Paradas y secuencias de reinicio: durante los primeros instantes de uso del sistema el número de paradas resulta bastante elevado. Estas se deben en gran medida a la realización de movimientos muy rápidos sin tener en cuenta el punto en el que se encuentra el robot y en menor medida al traspaso de los límites de seguridad del sistema. La secuencia de reinicio funciona correctamente para estos casos, pero ambos usuarios manifiestan que el proceso es poco visual y que algún tipo de indicador en el entorno simulado ayudaría a su comprensión. Después de varios minutos con el sistema, el número de paradas se reduce de forma exponencial hasta llegar a ser prácticamente nulo.
- Tiempo ejecución de operación: el tiempo promedio de la primera operación de *Pick & Place* oscila entre tres y cinco minutos. Una vez el usuario ha realizado la operación se observa una disminución considerable del tiempo de las siguientes, que empieza a oscilar entre uno y dos minutos. Después de unas pocas iteraciones más los tiempos llegan a oscilar por debajo de un minuto en ambos casos.

---

# **Capítulo 6**

# **Conclusiones**

---

En este capítulo se valorarán los resultados obtenidos del desarrollo de este proyecto en relación con los objetivos marcados y se plantean posibles trabajos futuros. Además se incluye la información sobre el presupuesto de este sistema y el desglose de tiempo dedicado al desarrollo.

## **6.1. Objetivos cumplidos**

En esta sección se recorren los objetivos señalados durante la sección 1.3 desglosando los resultados obtenidos de cada uno de ellos.

### **6.1.1. Objetivo principal**

Como se explicó en la subsección 1.3.1, el objetivo principal de este trabajo puede resumirse en la creación de un sistema de teleoperación para un brazo robótico a través de un entorno de Realidad Virtual, tratando de ser lo más natural, sencillo y preciso para el usuario que lo utilice. Este objetivo se puede considerar completado de forma satisfactoria. El proceso y pruebas que indican su cumplimiento se puede ver reflejado a lo largo del documento. A continuación recorremos los diferentes capítulos para dar fundamento a esta afirmación.

Siguiendo el desarrollo descrito en este documento en el capítulo 2, se detallan las tecnologías y los componentes *hardware* utilizados a lo largo de toda la implementación. Entre estas destacan el uso de la plataforma Godot, que además se enmarcó inicialmente como uno de los objetivos específicos debido a todas las ventajas descritas en este capítulo y a los conocimientos previos sobre ella. También destaca el uso del robot UR3 de Universal Robots debido a su facilidad de comunicación y control externo y otras tecnologías software como C# o protocolos de comunicación como TCP y HTTP,

encargados del envío de datos entre los distintos componentes.

En el capítulo 3, se aprovecha el empleo de una metodología de trabajo basada en el desarrollo *Agile* para simplificar la comprensión de las diferentes etapas de implementación y los problemas encontrados al lector.

En el capítulo 4, se explica el resultado final del proyecto y el funcionamiento de cada uno de sus componentes. Al tratarse de una plataforma diseñada para ser utilizada por usuarios, primero se describe el funcionamiento del sistema a ese nivel y posteriormente a un nivel inferior para que este sistema pueda ser entendido y modificado por desarrolladores experimentados.

Por último, en el capítulo 5, se lleva a cabo la validación experimental, la cual se divide en dos partes. En la primera de estas partes se evalúa el rendimiento general del sistema. Haciendo hincapié en la calibración de parámetros y la reducción de tiempos de cómputo. En la segunda parte se evalúa la experiencia de usuario del sistema para comprobar que este cumpla con los objetivos de diseño esperados.

Teniendo en cuenta todo lo mencionado y en vista de los resultados, el objetivo principal se considera cumplido en su totalidad.

### 6.1.2. Objetivos específicos y relacionados

A continuación se recorren los diferentes objetivos específicos y se detalla brevemente sus resultados. Por simplicidad se usará una abreviación del nombre del objetivo, pero se mantendrá el mismo orden que en la subsección 1.3.2:

- Comunicación bidireccional entre robot y entorno VR: completado de forma satisfactoria a través de la creación de un canal de comunicación mediante el protocolo RTDE.
- Interpretación de posiciones y cálculo de trayectorias: completado de forma satisfactoria a través del cambio de formato de posiciones y rotaciones del entorno VR al entorno del robot y uso de las funciones de cálculo de cinemática inversa de Universal Robots.

- Control de *gripper* (garra): completado satisfactoriamente a través de la utilización del URcap desarrollado por Robotiq para este elemento terminal y de mapeo de botones y control desde entorno simulado.
- Conexión local y remota del sistema: este objetivo se puede considerar como parcialmente resuelto. Todos los elementos del sistema han sido programados para utilizar la infraestructura de Internet para comunicarse, consiguiendo cumplir el requisito de ser un sistema generalista que no necesita configuraciones específicas de comunicación. Con estas bases podríamos lograr una comunicación tanto local como a distancia, pero debido a las limitaciones temporales y técnicas del proyecto solo ha sido validado y probado en área local a través de conexión Ethernet y WIFI.
- Extracción de posición y rotación de manos de usuario: completado satisfactoriamente a través del uso del paquete Godot XR Tools.
- Control de diferentes acciones del sistema usando los botones del propio visor de Realidad Virtual: captación de señal de botones completado satisfactoriamente a través de la creación de un árbol de acciones personalizado en Godot y control de acciones resuelto a través de nodo de control personalizado *urbot.cs*.
- Proyección de entorno real en entorno simulado: completado satisfactoriamente a través del montaje de un módulo personalizado con una placa Raspberry Pi. Este publica las imágenes obtenidas en un servidor web accesible por el ordenador encargado de la escena simulada.
- Uso de Godot para creación del escenario: completado satisfactoriamente.

Los objetivos relacionados parten de la premisa de haber aparecido de los resultados mostrados durante las fases de experimentación, por lo tanto, todos se han resuelto satisfactoriamente.

## 6.2. Aplicación de lo aprendido

En esta sección se comentarán los conocimientos aplicados a este trabajo. Por un lado, se tendrán en cuenta los derivados de la titulación y por otro lado, los que han necesitado de un aprendizaje personal o autodidacta.

### 6.2.1. A través de la titulación

A pesar de contar este proyecto con un importante factor personal y un considerable aprendizaje propio. La titulación me ha permitido desarrollar las bases necesarias para poder hacer frente tanto a la labor investigativa como a las necesidades técnicas demandadas. En esta sección recorreremos las asignaturas que han aportado un mayor valor a este desarrollo.

Primero de todo mencionar las asignaturas que me han introducido al mundo de la programación como: Fundamentos de Programación, Algoritmos y Estructuras de Datos, Diseño Software y Sistemas Operativos. Gracias a estas he comprendido el mundo del *Software* y he sido capaz de desarrollar un proyecto como este.

Cabe mencionar de forma individual a las asignaturas que han tenido un aporte más específico en este trabajo:

- Robótica Industrial: esta asignatura me enseñó los fundamentos de los brazos robóticos, su composición y movimiento. Además, me dio la oportunidad de trabajar con un brazo robótico de Universal Robots e interesarme por todas las posibilidades de estos.
- Fundamentos de Redes de Ordenadores: esta asignatura me dio los conocimientos básicos de comunicación entre máquinas. Basándonos en estos se ha desarrollado todo el esquema de comunicación de este proyecto.
- Fundamentos de Automática: esta asignatura me enseñó los fundamentos del control automático y el ajuste de controladores.
- Sensores y Actuadores: esta asignatura me enseñó a utilizar sensores y a integrarlos en placas como la Raspberry Pi utilizada en este trabajo.
- Modelado y Simulación de Robots: esta asignatura me enseñó los fundamentos del modelado 3D y la simulación de escenas. Gracias a ella he sido capaz de entender y configurar el entorno simulado de este trabajo.

### 6.2.2. De forma autodidacta

Además de los conocimientos desarrollados durante el grado, este proyecto me ha permitido aprender sobre nuevas disciplinas, mejorar mis conocimientos sobre otras y

trabajar con elementos nuevos para mí. A continuación se enumeran los principales conocimientos adquiridos:

- Comunicación externa con brazos robóticos a través del protocolo RTDE.
- Uso de distintos modos de movimiento y planificación de trayectorias de brazos robóticos.
- Diseño y creación de entornos de Realidad Virtual utilizando Godot.
- Control y actualización de elementos en escena virtual.
- Uso y configuración de gafas de Realidad Virtual.
- Uso del lenguaje C# y del *middleware* .NET.
- Toma, gestión y transmisión de imágenes en sistemas embebidos.
- Creación y comunicación con servidores HTTP.
- Implementación de APIs de comunicación.
- Uso avanzado de lenguaje LaTeX para la creación de textos técnicos.

### 6.3. Presupuesto y Tiempo Invertido

A continuación podemos ver la tabla 6.1 con los precios aproximados de cada uno de los elementos que componen el sistema desarrollado en este proyecto:

Elemento	Fabricante	Precio (EUR)
Brazo Robótico UR3	Universal Robots	23,681.00 €
Teach Pendant	Universal Robots	1,983.73 €
Oculus Quest 2	Meta	249.00 €
Router Archer AX23	Tp-link	79.83 €
HP VR Backpack G2	HP	3,073.38 €
Raspberry Pi 4 B	Raspberry	70.99 €
RaspiCam Camera Module 3	Raspberry	23.29 €
<b>Total:</b>		<b>29,161.22 €</b>

Cuadro 6.1: Presupuesto de proyecto

El tiempo invertido para la implementación y pruebas de este proyecto queda desglosado en la tabla 6.2:

<b>Fase</b>	<b>Tiempo (horas)</b>
Etapa 0	140h
Etapa 1	90h
Etapa 2	50h
Etapa 3	100h
Etapa 4	80h
Etapa 5	110h
Pruebas Finales	15h
Redacción	180h
<b>Total:</b>	<b>765h</b>

Cuadro 6.2: Desglose tiempo dedicación proyecto

## 6.4. Trabajos Futuros

A la vista del grado de cumplimiento de los objetivos y de las observaciones llevadas a cabo por los usuarios experimentales del sistema, en (sección 5.2), en esta sección se plantean una serie de líneas de desarrollo para el futuro de este proyecto:

- Operación totalmente remota del sistema: permitir el control totalmente remoto del sistema desde distintas posiciones geográficas. Se plantea hacer uso de una red 5G para reducir lo máximo posible la latencia.
- Mejora de visualización del entorno real: existen grandes limitaciones al utilizar una cámara 2D para observar el entorno real. Por ello se plantea la posibilidad de tratar de representar el entorno real de forma tridimensional, al igual que se hizo con el modelo 3D del brazo, utilizando por ejemplo dos sensores LIDAR y un algoritmo de correlación de puntos [16].
- Integración elementos terminales de distintas categorías: integrar y controlar elementos terminales de otras clases, como soldadores, lijas, pistolas de pegamento, etc. Con esto se dotaría al sistema de un mayor número de operaciones a realizar.
- Seguimiento visual de la secuencia de reinicio: introducir en el entorno simulado un indicador del estado del robot y en qué parte de la secuencia de reinicio se encuentra para que sea más fácil de comprender por el usuario.
- Sistema para evitar inicios bruscos: introducir un controlador que se adapte a cambios muy bruscos para no generar desplazamientos abruptos en los motores del robot.

- Integración del sistema en un entorno de trabajo real: se consideran varias vías para explotar sus capacidades como: integración en una factoría en la que se trabaje con materiales peligrosos, en un robot móvil de desactivación de explosivos o en cualquier otra actividad en la que este permita evitar posibles riesgos para los seres humanos.

# Bibliografía

---

- [1] David Vargas Frutos. Chaleco háptico multimodal para aplicaciones de realidad virtual. Paper, Universidad Politécnica de Madrid, Junio 2017.
- [2] Julio César Tafur Sotelo, César Peña, Cecilia Elisabet García Cena, and Rafael Aracil Santonja. Implementación de una plataforma experimental para un sistema de teleoperación robótica en tiempo real. *Revista Iberoamericana de Sistemas, Cibernetica e Informatica*, 7(1):69–74, 2010.
- [3] Ricardo John Palomares Orihuela. Robot da vinci: El quirófano del futuro. *Perfiles de Ingeniería*, 11(11), nov. 2016.
- [4] David Martínez Miranzo. Control mediante realidad virtual del robot Pepper, 2023-06-22.
- [5] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.
- [6] Sébastien Chazallet. *Python 3 - Los fundamentos del lenguaje (4<sup>a</sup> edición)*. 2024.
- [7] Mark J. Price. *C# 12 and .NET 8*. 2023.
- [8] Stephen Cleary. *Concurrency in C# Cookbook*. 2014.
- [9] Alan Thorn. *Scripting with C# in Godot: Common Tasks*, pages 53–103. Apress, Berkeley, CA, 2020.
- [10] Sander Vanhove. *Learning GDScript by Developing a Game with Godot 4*. 2024.
- [11] Universal Robots SL. Script Manual - CB-SERIES - SW3.15.4, 2021.
- [12] Enrique Soriano and Gorka Guardiola. *Fundamentos de Sistemas Operativos: Una Aproximación Práctica Usando Linux*, chapter Sockets(6.3), pages 335–339. 2022.
- [13] Universal Robots SL. Real-Time Data Exchange (RTDE) Guide, 2024.

- [14] Universal Robots SL. User Manual - UR3 CB-SERIES, 2023.
- [15] Matevž Borjan Zorec and Ulrich Baksh, Farnaz y Norbisrath. XR Teleoperation Demo Development. In *2023 International Conference on Intelligent Metaverse Technologies and Applications (iMETA)*, pages 1–6, 2023.
- [16] Abhishek Thakur and P Rajalakshmi. Real-time 3D Map Generation and Analysis Using Multi-channel LiDAR Sensor. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, pages 1–5, 2024.