UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Diseño, desarrollo y simulación de un gemelo digital de un brazo colaborativo Universal Robots mediante comunicaciones OPC UA y CoppeliaSim.

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: Gutiérrez Aparicio, Cesar

Tutor/a: Valera Fernández, Ángel

Cotutor/a: Blanes Noguera, Juan Francisco

CURSO ACADÉMICO: 2022/2023

Agradecimientos

"Quiero aprovechar esta oportunidad para expresar mi profundo agradecimiento por el cariño y el apoyo incondicional de mi familia a lo largo de estos años de carrera. También quiero dar las gracias a mis amigos y compañeros de estudios del máster, así como a Ángel y a mis profesores del máster, por compartir sus valiosos conocimientos conmigo. No puedo pasar por alto agradecer a mis compañeras en el equipo de AI2; su apoyo y amistad han sido invaluables. Y, por supuesto, a Paco, por brindarme la oportunidad de participar en este proyecto y por su constante apoyo."

Resumen

En esta tesis se presenta el diseño, desarrollo y simulación de un brazo colaborativo Universal Robots mediante comunicaciones OPC UA y CoppeliaSim, con el objetivo de crear un gemelo digital. Un gemelo digital es una réplica virtual de un objeto o sistema que permite simular su comportamiento en tiempo real y obtener datos para su análisis. En este caso, se busca obtener datos de un brazo robot en tiempo real para su visualización y análisis.

La justificación de este trabajo radica en la importancia actual de los sistemas ciberfísicos y los modelos de datos en la Industria 4.0, donde la creación de gemelos digitales es una herramienta clave para la toma de decisiones y la optimización de procesos industriales. La creación de un gemelo digital de un brazo robot puede permitir la optimización de su funcionamiento, el diseño de nuevas estrategias de control y la reducción de tiempos de mantenimiento y reparación.

Para desarrollar el gemelo digital, en primer lugar, se han de extraer los datos en tiempo real del brazo robot, para lo que se utiliza la librería RTDE con Python. Esta librería posibilita la conexión al controlador del brazo robot mediante una vía Ethernet, y la obtención de datos, tales como la posición, velocidad, aceleración, fuerzas y pares, entre otros. Estos datos se almacenan en el servidor implementado en lado del robot para que puedan ser accedidos por los clientes.

El acceso a los datos se realiza por medio de una estructura cliente-servidor mediante comunicación OPC UA. Esta arquitectura proporciona una gran interoperabilidad y seguridad para el intercambio de datos en tiempo real. Para establecer la comunicación cliente-servidor, se desarrolla un modelo de información en el que se modelan los objetos y variables necesarios exponer los datos en el servidor, y se generan los archivos XML necesarios para la creación del servidor OPC UA con Open62541. El cliente que puede ser implementado en varios lenguajes de programación, se comunica con el servidor a través de una conexión TCP/IP para acceder a los datos.

Por último, se realiza la simulación en CoppeliaSim importando el modelo del brazo robot y configurando sus propiedades y parámetros para permitir la simulación en tiempo real. Para integrar los datos recibidos en el cliente OPC UA con la simulación en tiempo real del brazo robot en CoppeliaSim, se desarrollan scripts en Python que permiten la comunicación entre ambos entornos. De esta forma, se dispone de una simulación que permite visualizar los datos de manera sencilla e integrarse en sistemas más complejos que involucran otros elementos de la cadena de producción o de la fábrica.

En resumen, en esta tesis se utiliza la librería RTDE con Python para la extracción de datos en tiempo real del brazo robot, se desarrolla una arquitectura cliente-servidor mediante el estándar OPC UA y se crea un servidor OPC UA con Open62541 para la transmisión segura y confiable de datos entre dispositivos. Además, se integran los datos recibidos por el cliente en la simulación en tiempo real del brazo robot en CoppeliaSim. El resultado que se espera obtener es un gemelo digital fiel a su homólogo físico en cuanto a estructura y comportamiento, que permita la simulación y análisis del brazo robot en tiempo real.

Palabras Clave: Gemelo Digital, Industria 4.0, Robótica colaborativa, OPC UA, CoppeliaSim

Índice general

Resumen I

Índice general V

Índice de figuras VII

Índice de tablas XI

I Memoria 1

1 Introducción 3

1.1 Contexto y justificación. 3
1.2 Objetivos 4
1.3 Estructura del trabajo 5

2 Fundamentos Teóricos 7

2.1 La Industria 4.0 y los Gemelos Digitales. 7
2.2 Robots Colaborativos Universal Robots 10
2.3 OPC UA: Comunicación Cliente-Servidor. 12
2.4 Software de Simulación 20

3 Universal Robots y la librería RTDE 29

3.1 Librería RTDE: Descripción y funcionamiento. 30
3.2 Comunicación con el brazo robot 31

4 Comunicación OPC UA 37

4.1 Consideraciones Previas a la Implementación 37
4.2 SDK OPC UA. 40

4.3 Implementación del Modelado de Información . . . 44
4.4 Implementación del servidor OPC UA . . . 50
4.5 Implementación del cliente OPC UA . . . 56

5 Simulación en CoppeliaSim 61
5.1 Configuración del Modelo y la Escena . . . 61
5.2 Entorno de programación de plugins en CoppeliaSim . . . 68
5.3 Programa principal del plugin del gemelo digital . . . 73

6 Resultados y análisis 77
6.1 Mejora y Optimización del Funcionamiento de la Aplicación . . . 78
6.2 Comparación Cuantitativa con un Robot UR5 Real . . . 83
6.3 Análisis Cualitativo en Diferentes Escenarios . . . 92

7 Conclusión 97
7.1 Objetivos Logrados . . . 98
7.2 Líneas de mejora . . . 99
7.3 Reflexión Final . . . 100

II Presupuesto 101

Presupuesto general del proyecto 103
Resumen del presupuesto . . . 103
Presupuesto de ejecución detallado . . . 103

III Apéndices 109

A Modelo de Información 111
A.1 Estructura del modelo de información . . . 112
A.2 Código del modelo de información . . . 117

B Planificación del proyecto 119
B.1 Planificación de tareas del proyecto . . . 120

C Despliegue de la Solución 123
C.1 Compilación del Modelo de Información . . . 124
C.2 Compilación de la Aplicación . . . 125

D Alineación ODS 129
D.1 Desarrollo de la Alineación con los ODS . . . 130

Índice de figuras

1.1. Diagrama global del desarrollo del proyecto mostrando el flujo de información entre los componentes 6

2.1. Brazo robótico UR5 simulado en el entorno de simulación URSim. 12
2.2. Jerarquía de la clase base ReferenceType de OPC UA. 16
2.3. Simulación de dos brazos UR3 en CoppeliaSim. 24
2.4. Diagrama de la estructura interna del software de simulación CoppeliaSim. Fuente: Coppelia Robotics. 26

4.1. Sistema empotrado LattePanda Delta 3. Fuente: lattepanda.com 39
4.2. Bash script para automatizar el proceso de construcción y compilación del servidor OPC UA. 41
4.3. Diagrama de etapas del desarrollo de la arquitectura cliente-servidor 42
4.4. Estructura del Robotics Companion Specification. Fuente: OPC Foundation, VDMA 45
4.5. Comienzo de la implementación del modelo de información en XML. 48
4.6. Creación de una instancia de brazo robot en el modelo de información. 49
4.7. Esquema resumen del funcionamiento multithreading del programa del servidor. . 51
4.8. Diagrama de flujo resumiendo el funcionamiento de los thread creados con la clase RobotManager. 52
4.9. Diagrama de flujo del thread para la gestión del servidor con la clase ServerManager. 53
4.10. Fragmento del archivo CMakeList.txt del servidor mostrando el uso de targets. . 55
4.11. Diagrama del funcionamiento del cliente OPC UA. 57
4.12. Diagrama de la ejecución de la función EmergencyStop del cliente OPC UA. . . . 59

5.1. Modelo de brazo robot UR5 mallado y dividido en secciones en CoppeliaSim. Fuente: Coppelia Robotics. 62

5.2. Segmentos y ejes del modelo de brazo robot UR5 en CoppeliaSim. Fuente: Coppelia Robotics. 63

5.3. Jerarquía completa del modelo de brazo robot UR5 en CoppeliaSim. Fuente: Coppelia Robotics. 64

5.4. Fragmento del código xml que implementa la estructura de la interfaz de usuario personalizada en CoppeliaSim. 66

5.5. Fragmento del código que implementa distintos callback en la interfaz de usuario personalizada en CoppeliaSim. 66

5.6. Diagrama de la navegación en la GUI personalizada creada en CoppeliaSim. . . . 67

5.7. Fragmento del archivo CMakeLists.txt para la compilación del plugin. 69

5.8. Diagrama de árbol con los principales componentes del plugin para CoppeliaSim. 73

5.9. Esquema completo de la estructura del plugin para CoppeliaSim. 74

5.10. Diagrama de flujo del hilo de actualización de la simulación del plugin para CoppeliaSim. 76

6.1. Fragmento del archivo CMake implementando un test con la herramienta AddressSanitizer. 80

6.2. Salida en línea de comandos de la herramienta AddressSanitizer mostrando un desbordamiento de búfer. 81

6.3. Diagrama de los puntos de adquisición de datos del ensayo con el robot UR5. . . 83

6.4. Posiciones programadas en el robot UR5 del laboratorio del instituto ai2. 85

6.5. Fragmento de código C++ del programa utilizado para la recolección de datos del robot UR5. 86

6.6. Gráficas de las tres coordenadas de posición del TCP del robot UR5 leídas del servidor OPC UA con el cliente Prosys Monitor. 86

6.7. Fragmento de código C++ del plugin implementando la obtención del nodo de la coordenada X de la posición del TCP 87

6.8. Fragmento de código C++ del plugin implementando 87

6.9. Gráfica temporal de las coordenadas de posición del TCP del robot UR5 extraídas con la interfaz RTDE. 88

6.10. Gráfica temporal de las coordenadas de posición del TCP obtenidas del servidor OPC UA. 89

6.11. Gráfica temporal de las coordenadas de posición del TCP obtenidas con el cliente OPC UA (Prosys Monitor). 89

6.12. Gráfica temporal de las coordenadas de posición del TCP obtenidas de la simulación en CoppeliaSim. 90

6.13. Gráfica 3D mostrando la trayectoria del TCP en metros del robot UR5 real. . . . 91

6.14. Robot UR5 real (derecha) y su gemelo digital simulado en CoppeliaSim (izquierda) moviéndose entre dos posiciones. 92

6.15. Dos robot UR3 simulados (derecha) y sus gemelos digitales en CoppeliaSim (izquierda) moviéndose en una trayectoria. 93

6.16. Diagrama de los elementos y conexiones de la prueba con dos robots UR3. 93

6.17. Prueba de operación con varios robots UR3 interaccionando con el entorno en el DISA 94

A.1. Símbolos estandarizados para representar los modelos de información de OPC UA. 112

A.2. Símbolos adicionales utilizados en los diagrama de bloques OPC UA. 112

A.3. Diagrama de bloques OPC UA estandarizado del modelo de información personalizado del gemelo digital. 113

A.4. Diagrama de bloques OPC UA estandarizado del tipo personalizado SensorType. 114

A.5. Diagrama de bloques OPC UA tipo MotionDevices cuyo hijo es RobotArm. . . . 114

A.6. Diagrama de bloques OPC UA del modelo de los nodos de los ejes (Axis) de las instancias RobotArm. 115

A.7. Diagrama de bloques OPC UA del modelo de los nodos de los sistemas de potencia (PowerTains) de las instancias RobotArm. 115

A.8. Diagrama de bloques OPC UA de los nodos de los estados y funciones del robot. 116

A.9. Diagrama de bloques OPC UA de los nodos del controlador del robot. 116

A.10. Sección del código xml de la implementación del primer eje del robot en el modelo de información. 117

A.11. Sección del código xml de la implementación de los sensores de fuerza del robot en el modelo de información. 117

A.12. Sección del código xml de la implementación de las funciones de parada del robot en el modelo de información. 118

B.1. Diagrama de Gantt de la planificación de las tareas del proyecto. 122

C.1. Estructura mínima necesaria de los ficheros y carpetas del proyecto para su correcto funcionamiento. 126

Índice de tablas

2.1. Namespaces utilizados en el proyecto 17
2.2. Métrica para la elección del software de simulación 23

7.1. Resumen del presupuesto general de ejecución 103

B.1. Hitos del proyecto 120
B.2. Planificación detallada del proyecto por tareas 120

D.1. Alineación del trabajo con los Objetivos de Desarrollo Sostenible 130

Parte I

Memoria

Capítulo 1

Introducción

1.1 Contexto y justificación

Según diversos informes de mercado (Wang y Edmondson 2022) (BRC 2023), el mercado de los robots colaborativos se valora actualmente por encima de los mil millones de euros en 2023 y se espera que alcance los 1.876 millones para 2026. Las aplicaciones en las que más se usan este tipo de robots son las del sector automovilístico, en donde los robots colaborativos son utilizados en tareas de montaje, pulido, inspección, etc. Según los grandes productores del sector, los fallos en robots son frecuentes, lo que conlleva pérdidas importantes al verse obligados a detener la cadena de montaje para realizar reparaciones o sustituir las unidades. Sin embargo, con el uso de robots colaborativos, es posible acceder a las unidades averiadas sin necesidad de detener el resto de la línea de montaje garantizando en todo momento la seguridad de los operarios.

Debido a este y otros motivos, muchas de las grandes compañías automovilísticas como Audi, Volkswagen o Nissan, han propuesto en los últimos años, el desarrollo de ´´fábricas inteligentes" que mejoren la flexibilidad y productividad apoyándose en el concepto de la colaboración humano-robot. Este concepto nace en el contexto actual de la Industria 4.0 y la nueva industria 5.0.

En este contexto global, en el que la demanda de robots colaborativos va en aumento, el diseño y desarrollo de gemelos digitales para esto brazos colaborativos se ha vuelto cada vez más relevante. Los gemelos digitales permiten crear réplicas virtuales precisas de los sistemas físicos, lo que facilita entre otras cosas, la simulación y el análisis de su comportamiento. El uso de los gemelos digitales juega un papel clave en la planificación de nuevas lineas de producción o el mantenimiento de líneas existentes, así como la de tareas específicas, como la generación de trayectorias complejas o interacciones con otros elementos del entorno.

La justificación de esta tesis viene originada por la necesidad de la obtención de datos para la monitorización del estado de varios robots colaborativos en el contexto de un proyecto europeo. Los cobots tienden a sufrir de mayores problemas de desgaste que los robots industriales tradicionales y es por ello por lo que es necesario el mantenimiento predictivo, objeto de dicho proyecto europeo. Este concepto, que se ampliará en el Capítulo 3, se basa en la toma masiva de datos

sobre el estado del robot y su posterior análisis mediante técnicas avanzadas de análisis de datos y de aprendizaje artificial, para lo que la disposición de un gemelo digital es imprescindible.

La creciente demanda de gemelos digitales en la industria, también ha revelado la necesidad de desarrollar metodologías efectivas para la creación de gemelos digitales de brazos colaborativos que permitan integrar los todos los elementos de las lineas de producción y las células en las que se instalan estos robots. Aunque algunos de los fabricantes de brazos robóticos industriales disponen de software para implementar gemelos digitales de sus productos, por lo geneal, estos software comerciales ofrecen poca flexibilidad a la hora de integrar elementos de otros fabricantes. Antecedentes de gemelos digitales de Kuka o ABB pueden encontrarse en la bibliografía de este documento.

En este sentido, el presente trabajo aborda directamente estas problemáticas actuales. Por una parte, el uso del protocolo OPC UA proporciona una gran interoperatividad y robustez, y el desarrollo de un modelo de información personalizado para el proyecto ofrece grandes ventajas en cuanto a escalabilidad y flexibilidad se refiere. Mientras tanto, por otra parte, las capacidades de un software de simulación como CoppeliaSim permite la visualización, análisis y optimización del brazo robot y su interacción con el entorno.

En resumen, este trabajo surge en el seno de un proyecto europeo, como respuesta a la creciente importancia y demanda de los gemelos digitales en aplicaciones de brazos colaborativos en el contexto de la Industria 4.0. El uso de técnicas como el mantenimiento predictivo, permite evitar costes de tiempos de inactividad y reducir los gastos asociados con reparaciones y reemplazos de componentes en el actual panorama de desabastecimiento que sufren muchas empresas justifica este trabajo. Se espera que los resultados obtenidos en esta tesis sienten las bases para futuras aplicaciones y mejoras en el campo de los gemelos digitales y la robótica colaborativa, fomentando así la innovación y el progreso en este ámbito en constante evolución.

1.2 Objetivos

El presente trabajo tiene como objetivo principal diseñar, desarrollar y simular un gemelo digital funcional de brazos colaborativos Universal Robots mediante la integración de comunicaciones OPC UA y la plataforma de simulación CoppeliaSim. El principal uso del gemelo digital será la monitorización y el registro de datos para el mantenimiento predictivo del robot UR. La aplicación debe ser también extensible a varios brazos robot UR. Para lograr estas características, se han planteado una serie de objetivos específicos que recogen los principales requerimientos de un gemelo digital:

- Comportamiento preciso: Se asegurará que el gemelo digital pueda interactuar con los sensores y otros dispositivos físicos, así como proporcionar una comunicación suave y fluida para visualizar los datos relevantes.
- Sincronización: Asegurar que el gemelo digital y su contraparte física se mantengan en sincronía. Esto permitirá la recepción de datos desde el sistema hacia el gemelo digital, y facilitará la toma de decisiones.

- Convergencia: Entre los espacios físicos y digitales, identificando posibles diferencias y optimizando el gemelo digital en base a su contraparte física. Se garantizará que el gemelo digital esté siempre sincronizado con los sensores y otros componentes físicos relevantes.
- Verificación y validación: Proporcionar mecanismos de verificación y validación para el gemelo digital antes de su despliegue. Esto implicará la realización de pruebas para garantizar su correcto funcionamiento y sus capacidades.
- Protocolos de automatización: Integrar los protocolos de automatización necesarios para permitir la comunicación entre los diferentes dispositivos y sistemas involucrados en el gemelo digital. Se asegurará que la interacción entre diferentes partes se realice de manera eficiente y efectiva.
- Interoperabilidad: Desarrollar una arquitectura independiente de la plataforma que puede adaptarse a distintos sistemas operativos Windows, Linux o cualquier otro. Además, que sea compatible con diferentes lenguajes de programación como C/C++, Java, .NET, Python, entre otros. Esto permite una mayor flexibilidad y facilidad de implementación.
- Deslocalización del sistema: Facilitar la comunicación e interacción entre diferentes gemelos digitales de dispositivos físicos. Esto permitirá el intercambio de datos entre gemelos digitales aunque se encuentren en diferentes ubicaciones.
- Modificabilidad: Permitir la posibilidad de modificar la aplicación del gemelo digital de forma que pueda adaptarse a varios tipos de robots UR o incluso como la incorporación de nuevos sensores o actuadores.
- Reusabilidad: Fomentar la reusabilidad del gemelo digital y sus componentes, especialmente en entornos donde la línea de productos está en constante cambio. Se buscará mejorar la modularidad del gemelo digital y permitir la reutilización de componentes individuales.

1.3 Estructura del trabajo

Con el objetivo de establecer una estructura clara y coherente, el trabajo se divide en diferentes capítulos que abordan los aspectos clave del desarrollo de la aplicación. En la Figura 1.1 se muestra un diagrama detallado de las principales partes y componentes del desarrollo del gemelo digital para brazos UR con el fin de facilitar la comprensión del flujo de información entre las distintas partes.

En este primer capítulo se ha presentado una introducción al tema, contextualizando la importancia de los gemelos digitales y estableciendo los objetivos del estudio. A continuación, en el Capítulo 2 se proporciona una base teórica sólida que abarca los conceptos fundamentales relacionados con los gemelos digitales, las comunicaciones OPC UA y el software de simulación de CoppeliaSim.

La implementación del gemelo digital propiamente dicha se aborda en el Capítulo 3, Capítulo 4 y Capítulo 5. El desarrollo del gemelo digital se ha estructurado en 3 partes, correspondiendo cada una de estas a los capítulos citados anteriormente. Comenzando por el Capítulo 3, se presenta la comunicación con los robots físicos mediante el uso de la librería RTDE de Universal Robots y

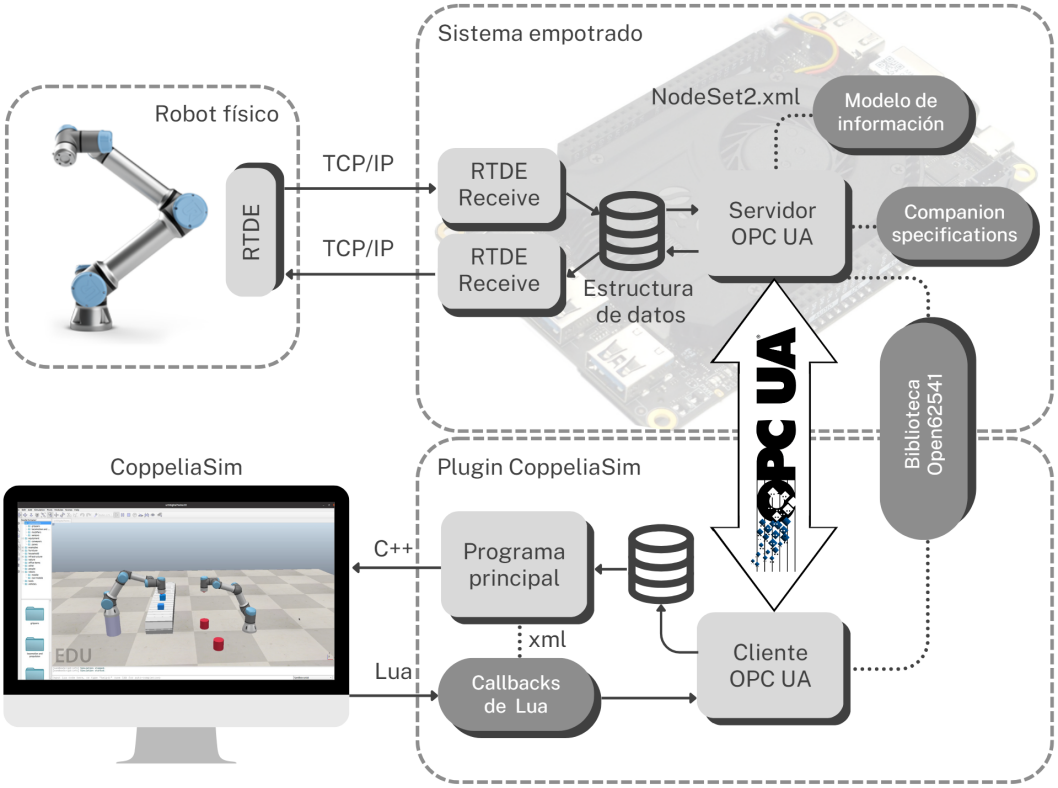


Figura 1.1: Diagrama global del desarrollo del proyecto mostrando el flujo de información entre los componentes

se desarrollan la extracción de datos y el control del robot mediante las interfaces RTDE Receive y RTDE Control respectivamente.

A continuación, en el Capítulo 4 se profundiza en el diseño del gemelo digital, detallando la arquitectura de las comunicaciones OPC UA y las decisiones de diseño que permiten la sincronización efectiva con el brazo colaborativo físico. En esta parte se presentan el programa que implementa el servidor OPC UA que corre en el sistema empotrado, y el cliente OPC UA implementado en el plugin de CoppeliaSim tal y como se muestra en el diagrama de la Figura 1.1.

Luego, en el Capítulo 5, se explora la integración de los datos del gemelo digital en la plataforma de simulación CoppeliaSim, describiendo los pasos necesarios para lograr una simulación y visualización precisa y efectiva. En el Capítulo 6 se realiza una evaluación de los resultados obtenidos, comparando el comportamiento y rendimiento del gemelo digital con su contraparte física mediante varios ensayos en aplicaciones prácticas.

Finalmente, se presentan las conclusiones del estudio, resumiendo los logros alcanzados, discutiendo las implicaciones de los resultados y ofreciendo recomendaciones para investigaciones futuras en el campo de los gemelos digitales y la robótica colaborativa.

La tesis incluye dos partes que añaden información relevante al proyecto. La Parte II desarrolla un presupuesto básico del proyecto incluyendo materiales y mano de obra, mientras que la Parte III incluye varios apéndices de distinta naturaleza. En el Apéndice A se presentan varios diagramas describiendo en detalle el modelo de información OPC UA completo, mientras que, el Apéndice B, recoge un resumen de la planificación llevada a cabo para el desarrollo del proyecto.

Capítulo 2

Fundamentos Teóricos

2.1 La Industria 4.0 y los Gemelos Digitales

La Cuarta Revolución Industrial, también denominada, industria 4.0, se caracteriza por los avances digitales que han transformando la forma de operar de muchas empresas. La historia de la industria 4.0 se remonta a la década de 1950, cuando se introdujeron los primeros sistemas de control numérico (CN) en la fabricación. A medida que la tecnología avanzaba, surgieron nuevos avances, como los robots industriales en la década de 1960 y los sistemas de planificación de recursos empresariales (ERP) en la década de 1990. Con la rápida popularización de internet y la creciente conectividad global, la industria 4.0 se ha ido acelerando en las últimas décadas, brindando nuevas oportunidades y desafíos para las empresas de todo el mundo.

2.1.1 El Marco de la Cuarta Revolución Industrial

En la actualidad, se reconoce de manera amplia que la industria 4.0 se sustenta en cuatro pilares fundamentales que son esenciales para su funcionamiento. Los dos primeros pilares son:

- Interconexión: Es un concepto esencial en la industria 4.0 y se refiere a la conexión y comunicación entre máquinas, sistemas y personas. Esto se materializa, por ejemplo, en el Internet de las cosas (IoT), donde los dispositivos y sensores están conectados a la red y pueden intercambiar datos entre sí. Esta interconexión permite un flujo de información constante a alta velocidad, lo que brinda una mayor visibilidad y control sobre los procesos de producción. Además, la interconexión facilita la toma de decisiones basadas en datos y la optimización de la eficiencia y la productividad.
- Computación en la nube: La computación en la nube desempeña un papel fundamental en la industria 4.0 al proporcionar un almacenamiento y procesamiento de datos escalables y accesibles desde cualquier lugar. Los datos generados por los dispositivos y sistemas conectados se pueden almacenar en la nube, lo que permite un acceso instantáneo y seguro a la información. Esto es especialmente beneficioso para las empresas tecnológicas, ya que pueden almacenar grandes cantidades de datos, analizarlos y extraer conocimientos

valiosos para mejorar la toma de decisiones. Además, la computación en la nube facilita la colaboración y el intercambio de información entre diferentes actores de la cadena de producción.

Estos dos pilares han sido la base sobre la cual se comenzó a construir la industria 4.0 y los cuales impulsan los avances tecnológicos que la caracterizan. La interconexión y la computación en la nube permiten una mayor comunicación y acceso a datos, lo que a su vez impulsa los otros dos pilares clave:

- Big data: La industria 4.0 genera grandes volúmenes de datos gracias a la interconexión de dispositivos y sistemas. Estos datos, conocidos como big data, contienen información valiosa que, mediante el uso de técnicas de análisis y procesamiento, puede proporcionar conocimientos profundos sobre los procesos de producción, la calidad del producto, el mantenimiento predictivo, entre otros aspectos relevantes. El análisis de datos permite identificar patrones, tendencias y anomalías, y utilizar esta información para tomar decisiones informadas y optimizar los procesos. El análisis de big data también puede ayudar a mejorar la eficiencia energética, reducir costos y mejorar la calidad y la personalización de los productos entre otras cosas.
- Inteligencia artificial y machine learning: La inteligencia artificial (IA) y el aprendizaje automático (machine learning) son pilares clave de la industria 4.0. Estas tecnologías permiten a las máquinas y sistemas adquirir conocimiento, aprender de los datos recogidos y tomar decisiones de manera autónoma. La IA se utiliza en la industria 4.0 para mejorar la automatización de procesos, optimizar la eficiencia y permitir la toma de decisiones inteligentes. Por ejemplo, los sistemas de visión artificial pueden identificar y clasificar objetos en tiempo real, los algoritmos de aprendizaje automático pueden predecir y evitar futuros fallos en la maquinaria, y los chatbots impulsados por IA pueden proporcionar asistencia en tiempo real a los operadores.

2.1.2 Antecedentes de los gemelos digitales

La robótica desempeña un papel central en la transformación de los procesos industriales en los avances tecnológicos de la industria 4.0. Los robots industriales se han vuelto indispensables en la fabricación de muchos productos, ya que permiten la automatización de tareas repetitivas y peligrosas, mejorando la eficiencia y seguridad de los operarios en el entorno de trabajo. Estos robots están equipados con sensores y sistemas de visión que les permiten adaptarse y colaborar con los seres humanos de manera segura, lo que impulsa la llamada "colaboración hombre-máquina" en la industria 4.0 (Nath y Schalkwyk 2021).

Esta colaboración ha propiciado la surgimiento de los robots colaborativos o cobots. Estos cobots están diseñados para trabajar junto a los seres humanos en la línea de producción, brindando una mayor flexibilidad, adaptabilidad y seguridad en el entorno laboral. Equipados con una gran variedad de sensores y programados para interactuar de manera segura con los operadores humanos, los robots colaborativos permiten una colaboración directa y sin barreras, optimizando la eficiencia y facilitando la personalización en la producción.

En el contexto de la industria 4.0, los gemelos digitales se han convertido en una herramienta esencial para la optimización de los procesos industriales (Øvern 2018). En palabras sencillas, un

gemelo digital es una réplica virtual de un producto, proceso o sistema. Estos modelos digitales capturan información detallada sobre el rendimiento, el mantenimiento y el comportamiento de un activo o proceso físico y, permiten el monitoreo continuo, el análisis predictivo y la simulación de escenarios; lo que ayuda a mejorar la toma de decisiones y la optimización de los procesos en la industria 4.0.

Los gemelos digitales se integran con la robótica de manera natural. Los robots conectados a los gemelos digitales pueden recibir información en tiempo real y ajustar su comportamiento y rendimiento de acuerdo con los datos recopilados. Esto dota a los robots una mayor autonomía y adaptabilidad, así como una mayor capacidad para realizar tareas complejas y personalizadas en entornos cambiantes. La integración de los gemelos digitales con la robótica también ha tenido un impacto significativo en el ámbito del mantenimiento predictivo. Tradicionalmente, el mantenimiento en la industria operaba bajo el enfoque reactivo o preventivo, lo que significaba que las intervenciones en los robots se realizaban después de que ocurriera el fallo o en intervalos regulares independientemente del estado real de los equipos. Esto conllevaba costes extra de tiempos de inactividad y posibles daños en la línea de producción.

Sin embargo, con la introducción de los gemelos digitales, ha impulsado un el denominado mantenimiento predictivo. Al contar con un gemelo digital, que reproduce el comportamiento y rendimiento del robot físico, es posible monitorizar constantemente su estado y recolectar datos precisos sobre su funcionamiento. Estos datos son analizados utilizando técnicas avanzadas de análisis de datos y aprendizaje automático, permitiendo identificar patrones y anomalías que podrían indicar futuros fallos o desviaciones en el rendimiento.

Mediante el mantenimiento predictivo basado en gemelos digitales, las empresas pueden tomar decisiones más informadas y proactivas sobre cuándo realizar mantenimiento, evitando costes de tiempos de inactividad y reduciendo los gastos asociados con reparaciones y reemplazos de componentes. Además, esta aproximación al mantenimiento optimiza el uso de recursos, ya que las intervenciones se llevan a cabo cuando realmente son necesarias y no de manera periódica, independientemente del estado del robot. Lo que se alinea con los objetivos de desarrollo sostenible y economía circular impulsados por las instituciones.

La estructuración e implementación de gemelos digitales están intrínsecamente ligadas a la aplicación para la cual se diseñan. Dado que el enfoque de esta tesis se centra en el desarrollo de un gemelo digital destinado a brazos UR, en el marco de un proyecto europeo con carácter de investigación y desarrollo, es fundamental reconocer la falta de referencias específicas en este ámbito. A pesar de la ausencia de antecedentes directos en la creación de gemelos digitales específicos para brazos UR, es relevante destacar la existencia de una solución comercial denominada Rocketfarm. Este software se orienta a habilitar la comunicación OPC UA en robots UR, permitiendo la conectividad de estos con otros dispositivos y software compatibles con OPC UA mediante un estándar de conectividad de datos ampliamente aceptado. No obstante, puesto que la solución se centra sólo en la parte de las comunicaciones, no implementa la mayor parte de las funciones esperadas de un gemelo digital.

En conclusión, la industria 4.0 ha sido una revolución tecnológica que ha transformado profundamente la forma en que operan muchas las empresas. No obstante, aún mientras la industria 4.0 sigue evolucionando y transformando la sociedad, ya se vislumbran nuevos horizontes. Uno de ellos es el concepto de industria 5.0, que se centra en la colaboración estrecha entre huma-

nos y máquinas. En contraste con la automatización total de la industria 4.0, la industria 5.0 busca aprovechar la inteligencia y habilidades humanas en combinación con las capacidades de la tecnología. Esto implica una mayor integración de la robótica colaborativa y la inteligencia artificial en los entornos de trabajo, donde los seres humanos y los robots trabajan juntos en tareas complejas, compartiendo conocimientos y habilidades. Los gemelos digitales juegan un rol importante en este nuevo paradigma, permitiendo la detección temprana de riesgos y la simulación de escenarios para garantizar condiciones óptimas de trabajo. Además, fomentarán el aprendizaje continuo y la mejora de las habilidades tanto de los trabajadores como de las máquinas, a través de la retroalimentación constante y la actualización de los modelos virtuales. Este papel crucial de los gemelos digitales, sirviendo como puente entre humanos y máquinas, permite aprovechar las fortalezas y habilidades únicas de cada uno, convirtiéndose aí en uno de los pilares fundamentales de la nueva industria 5.0.

2.2 Robots Colaborativos Universal Robots

Esta sección pretende definir qué es un cobot y detallar algunas de sus características y aplicaciones tomando como ejemplo los robots colaborativos de la marca Universal Robots (UR). Por definición el término de cobot o robot colaborativo es el de un robot industrial articulado que se caracteriza por ser de pequeñas y medianas dimensiones cuyas características los hace muy seguros para trabajar con los humanos hasta el punto de no necesitar vallados de seguridad. Puesto que cobots fueron creados para ayudar a las personas a realizar tareas de automatización, la interacción con los operarios es fundamental. Se podría decir que esta es la principal diferencia entre los cobots y los robots industriales clásicos, aunque realmente se diferencian en muchos aspectos. A continuación se exponen algunas de las principales características de los robots colaborativos, las cuales también los diferencian de los robots industriales:

- Tamaño: Los robots colaborativos de UR son diseñados para ser compactos y fácilmente transportables. Aunque su tamaño reducido limita su capacidad de carga, los cobots UR, como el nuevo modelo UR20, pueden manejar cargas de hasta 20 kg, lo cual es suficiente para muchas aplicaciones industriales.
- Seguridad: Los cobots son capaces de, en cierta manera, conocer su entorno y detenerse automáticamente en caso de detectar una intrusión en su espacio gracias a su peso reducido y a los sensores con los que están equipados. Ello permite la reducción de medidas de seguridad entorno estos robots e incluso la eliminación de vallados protectores.
- Programación: A diferencia de los robots industriales convencionales, la programación de los cobots de UR es especialmente sencilla e intuitiva. Los cobots UR ofrecen por ejemplo, un modo de programación por puntos, en el cual un operario puede guiar físicamente al robot a través de los puntos deseados, utilizando su mano. Esta interfaz de programación amigable permite que cualquier persona, incluso sin experiencia previa en robótica, pueda programar y modificar rápidamente el comportamiento del cobot para adaptarlo a diferentes tareas.
- Versatiliad: Los cobots de Universal Robots están diseñados para ser altamente versátiles en su aplicación. Pueden ser utilizados para realizar una amplia variedad de tareas, desde ensamblaje y manejo de materiales hasta soldadura y paletización. Además, la capacidad

de reprogramación rápida y sencilla permite adaptar el cobot a diferentes necesidades de producción, lo cual es especialmente beneficioso en entornos cambiantes.

Fundada en 2005 en Dinamarca por Esben Østergaard, Universal Robots fue una de las primeras compañías en introducir robots colaborativos en la industria manufacturera. Su primer modelo, el UR5, se lanzó en 2008 y marcó el inicio de una revolución en la automatización industrial al ofrecer una solución más segura y accesible para la colaboración entre humanos y robots. Con el tiempo, la compañía ha desarrollado una línea completa de robots colaborativos, incluyendo el UR3, UR10 y modelos más avanzados con características mejoradas en términos de precisión, carga útil y capacidades de programación. La visión de Universal Robots ha sido democratizar la robótica, permitiendo a empresas de todos los tamaños y sectores implementar la automatización de manera eficiente y rentable. Su innovadora tecnología y enfoque centrado en el cliente les han permitido mantener una posición influyente en la industria de la robótica colaborativa.

Universal Robots dispone de una amplia gama de herramientas abiertas a sus usuarios para la programación y el desarrollo de aplicaciones. Al igual que los robots industriales, los cobots de UR disponen de una tableta (teach pendant o también denominada polyscope) que sirve como interfaz principal con el usuario. El teach pendant ofrece la posibilidad de programación mediante interfaces gráficas de usuario (GUI) intuitivas. Estas interfaces permiten la creación de programas utilizando bloques predefinidos y diagramas de flujo, simplificando aún más el proceso de programación. El usuario dispone de funcionas comunes organizadas en grupos para facilitar la creación de programas. Las características de comunicación integradas en los cobots UR permiten también la conexión con dispositivos externos, como manipuladores, sensores o sistemas de visión, lo que amplía las posibilidades de interacción y control del robot.

Además de la programación desde la interfaz de usuario, una de las formas más comunes de programar los robots UR es a través del modo de programación por puntos. En este modo, el robot desactiva los frenos de los ejes compensado únicamente la fuerza de la gravedad, es entonces cuando un operario puede guiar físicamente el brazo del robot moviéndolo a través de los puntos deseados en su trayectoria de movimiento. El robot registra los movimientos realizados y genera automáticamente el código de programación correspondiente. Esta metodología permite una programación rápida y visual, eliminando la necesidad de aprender lenguajes de programación complejos.

Otra herramienta destacada realmente útil es el URSim, un entorno de simulación de Universal Robots que permite probar y depurar programas antes de implementarlos en un entorno de producción real. El URSim dispone de representaciones virtuales de varios modelos de cobots de UR lo que, de forma parecida a como lo haría un gemelo digital, permite a los operarios evitar posibles errores y optimizar el rendimiento del robot antes de llevarlo a la línea de producción. En la Figura 2.1 se muestra un robot UR5 simulado en el entorno virtual de URSim. Esta simulación virtual también permite verificar la viabilidad de las tareas programadas, lo que contribuye a una mayor eficiencia y seguridad en la implementación final del robot. URSim está disponible tanto como aplicación para sistemas de Windows y Linux, así como encapsulada en un contenedor de Docker lo que permite ejecutar la aplicación en cualquier sistema operativo.

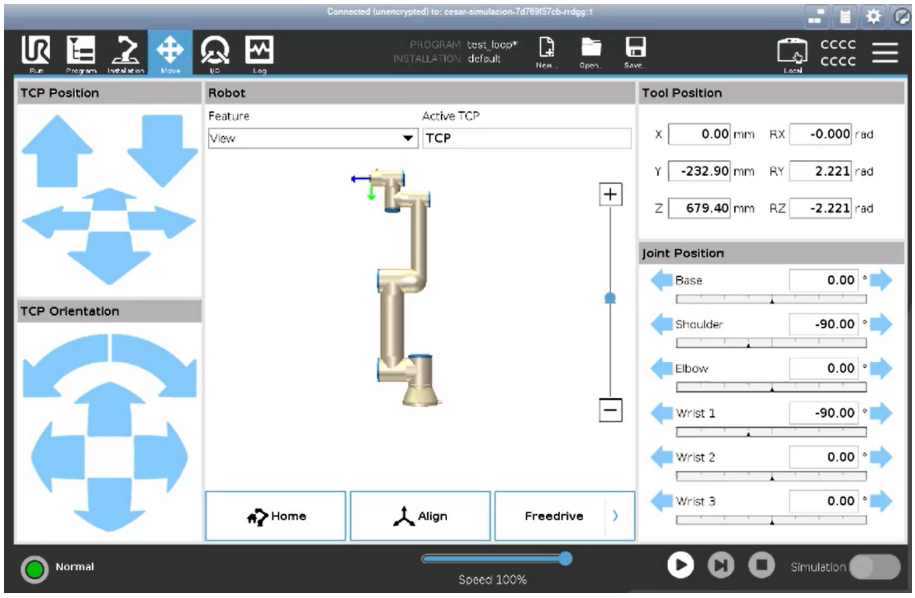


Figura 2.1: Brazo robótico UR5 simulado en el entorno de simulación URSim.

2.3 OPC UA: Comunicación Cliente-Servidor

OPC UA (Open Platform Communications Unified Architecture) es un estándar de comunicación industrial que ha ganado popularidad en la industria debido a las ventajas y características que ofrece en comparación con otros protocolos de comunicación utilizados en entornos industriales. En esta sección se realiza una breve presentación del estándar OPC UA, se discuten algunas de sus principales características y se introducen algunos conceptos sobre la arquitectura de comunicaciones cliente-servidor. Esta última información se ampliará en el Capítulo 4 profundizando en los detalles del modelo de información y la implementación de servidores y clientes OPC UA entre otros.

2.3.1 Introducción a OPC UA

La historia de OPC UA se remonta a la necesidad de superar las limitaciones del protocolo OPC clásico (OLE for Process Control), que había sido un estándar clave en la automatización industrial desde finales del siglo XX. Sin embargo, a medida que los requerimientos de la industria evolucionaron y la conectividad se volvió más crucial, se hicieron evidentes las limitaciones del protocolo OPC clásico. Mientras que OPC clásico tuvo éxito en proporcionar una forma estandarizada de intercambio de datos entre dispositivos y sistemas, carecía de la flexibilidad necesaria para abordar las complejidades de la Industria 4.0 (Mahnke, Leitner y Damm 2009).

A diferencia de su predecesor, OPC UA se ha diseñado desde cero para satisfacer las demandas cambiantes de la automatización moderna (OPC Foundation 2018). Una de las diferencias más notables entre ambos es la arquitectura de servicio orientada a objetos de OPC UA, en contraste con la arquitectura basada en DCOM (Distributed Component Object Model) de OPC clásico. Esta transición a una arquitectura orientada a objetos brinda mayor modularidad y capacidad para representar datos y funcionalidades de manera más eficiente y estructurada.

Además, OPC UA aborda la necesidad de una comunicación segura y confiable en la era digital. Mientras que OPC clásico carecía de mecanismos de seguridad inherentes, OPC UA prioriza la seguridad mediante la implementación de cifrado, autenticación y firmas digitales, asegurando que la integridad y la confidencialidad de los datos estén protegidas durante todo el proceso de comunicación.

Dentro del entorno de la automatización industrial, junto a OPC UA, existen otros protocolos que son utilizados en diferentes contextos. El ejemplo más conocido es quizás Modbus, un protocolo de comunicación serial ampliamente adoptado para la adquisición de datos y el control de dispositivos en diversos sistemas. Aunque Modbus ha demostrado su valía en aplicaciones específicas, carece de algunas de las capacidades de escalabilidad y seguridad integral que distinguen a OPC UA.

Otro protocolo comercial destacado es Profibus, que ha sido ampliamente utilizado en sistemas de automatización industrial. Profibus se ha centrado típicamente en la comunicación en tiempo real en redes industriales, pero no ha abordado de manera exhaustiva los conceptos relacionados con la interoperabilidad y seguridad avanzada.

Por otro lado, Ethernet/IP es un protocolo basado en Ethernet que se ha posicionado como una solución muy interesante para las comunicaciones industriales. Aunque es adecuado para aplicaciones en tiempo real y ha ganado terreno en entornos de fabricación, carece de ciertas características de seguridad y flexibilidad que OPC UA ha integrado en su diseño.

A pesar de que estos protocolos han demostrado ser útiles en situaciones específicas, OPC UA ha ganado popularidad en la industria debido a las ventajas y características que ofrece en comparación con estos otros protocolos de comunicación industriales. Estas ventajas y características son las siguientes:

- Interoperabilidad: OPC UA ha sido rediseñado desde cero para garantizar la interoperabilidad entre diferentes sistemas y tecnologías. Esto significa que los dispositivos y sistemas basados en diferentes plataformas y desarrollados por diferentes proveedores pueden comunicarse sin problemas, sin importar las diferencias en el hardware, el sistema operativo o los lenguajes de programación utilizados. Proporciona además una capa de abstracción que permite a los sistemas intercambiar información de manera transparente, lo que simplifica enormemente la integración de sistemas heterogéneos.
- Flexibilidad y escalabilidad: La alta flexibilidad y escalabilidad de OPC UA, lo hacen adecuado para una amplia gama de aplicaciones y entornos industriales. Puede adaptarse a diferentes requisitos de rendimiento, desde aplicaciones de baja velocidad hasta sistemas en tiempo real. Además, OPC UA permite una fácil ampliación y configuración de sistemas distribuidos, lo que permite su adaptación a medida que las necesidades de la industria evolucionan.
- Seguridad: OPC UA ofrece mecanismos de seguridad robustos de nivel industrial para proteger la confidencialidad, integridad y autenticidad de los datos intercambiados. Utiliza tecnologías como la encriptación y la firma digital para garantizar que la comunicación y los datos transmitidos sean seguros. Además, incluye funciones de autenticación y autorización para controlar el acceso a los recursos y garantizar que solo los usuarios autorizados puedan realizar operaciones específicas.

- Independencia de plataforma y transporte: Como ya se ha comentado, uno de los mayores puntos fuertes de OPC UA es su independencia de la plataforma y el transporte subyacente, lo que significa que puede funcionar en diferentes sistemas operativos, como Windows, Linux o sistemas embebidos. Además, puede también utilizar diferentes protocolos de transporte, como TCP/IP, HTTPS o incluso redes inalámbricas.
- Arquitectura orientada a servicios: OPC UA continua siendo una arquitectura orientada a servicios, lo que significa que los dispositivos y sistemas se exponen a través de servicios estándar que permiten el acceso, la lectura, la escritura y el monitoreo de datos. Esto facilita la interoperabilidad y el desarrollo de aplicaciones que pueden aprovechar estos servicios para interactuar con los sistemas conectados.

Tal como expone la OPC Foundation (2022a), estándar OPC UA se basa en dos pilares fundamentales: los mecanismos de transporte y la modelización de datos. Estos componentes son esenciales para proporcionar una plataforma de comunicación robusta y flexible.

Por un lado, los mecanismos de transporte definen diferentes métodos optimizados para el intercambio de datos en diversos casos. La base de OPC UA define un protocolo binario TCP optimizado para comunicaciones de alto rendimiento en redes locales (intranet), así como una adaptación de estándares de internet ampliamente aceptados, como son XML y HTTP, para permitir comunicaciones a través de firewalls en internet. Ambos mecanismos de transporte utilizan el mismo sistema de basado en mensajes seguros conocido en Web Services. De esta forma, el modelo de comunicación no depende de los mecanismos de transporte, y permite añadir nuevos protocolos de transporte en el futuro.

Por otro lado, la modelización de datos en OPC UA establece las reglas y los bloques de construcción necesarios para exponer un modelo de información. Define también los puntos de entrada en el espacio de direcciones (address space) y los tipos base utilizados para construir la jerarquía que organiza los tipos de nodos. Este modelo base puede ampliarse mediante la creación de modelos de información adicionales que se construyen sobre los conceptos de modelización información establecidos por los denominados “OPC UA Specifications”, los cuales actúan como guías o manuales para la estandarización de las comunicaciones. Además, se definen conceptos mejorados, como la descripción de máquinas de estado utilizadas en diferentes modelos de información. Los fundamentos de la modelización de información se describen en las siguientes secciones, y se presenta la implementación en el Capítulo 4.

Los servicios de OPC UA (UA Services) son el mecanismo de contacto entre los servidores, que proveen el modelo de información y los clientes que consumen ese modelo de información. Estos servicios se definen de manera abstracta y utilizan los mecanismos de transporte para intercambiar datos entre el cliente y el servidor. Esto permite que un cliente pueda acceder a los paquetes de datos más pequeños sin necesidad de comprender el modelo completo expuesto por el servidor en su totalidad. Este acceso específico se logra mediante un mapeo del modelo de información del cual se habla en la subsección 2.3.4. Por otra lado, los clientes de OPC UA que pueden entender los modelos de información completos pueden aprovechar características más avanzadas definidas para administradores de dominios y otros casos de uso especiales.

En resumen, OPC UA se basa en mecanismos de transporte y modelización de datos. Los mecanismos de transporte permiten la comunicación eficiente y segura entre los sistemas, mientras que la modelización de datos establece las reglas y los bloques de construcción para exponer un

modelo de información consistente. Estos pilares fundamentales son la base de la arquitectura robusta, flexible y escalable de OPC UA, lo que lo convierte en un estándar ampliamente utilizado en la comunicación industrial.

2.3.2 Conceptos sobre el Modelado Información en OPC UA

Como ya se ha comentado en la sección anterior, el modelo de información es uno de los pilares fundamentales de OPC UA y es esencial para comprender cómo se estructuran y representan los datos en este estándar. El modelo de información define cómo se organizan y describen los objetos, variables y funciones en el entorno OPC UA. El objetivo principal del modelo de información en OPC UA es proporcionar una estructura uniforme y coherente para la representación y el intercambio de datos entre diferentes sistemas de forma estandarizada. Esto se logra mediante la definición de una serie de conceptos y principios que permiten la interoperabilidad entre los dispositivos y aplicaciones que utilizan OPC UA (Profanter 2023).

Una de las características clave del modelo de información es su enfoque en la abstracción y la reusabilidad. En lugar de definir una estructura de datos específica para cada aplicación o dispositivo, OPC UA utiliza un conjunto de tipos de datos genéricos y flexibles que pueden adaptarse a diferentes contextos y necesidades. Esto se consigue gracias a un enfoque orientado a objetos, donde los datos se representan como objetos con propiedades, métodos y relaciones definidas. Los objetos son instancias de los tipos. Estos tipos pueden ser: los tipos base de OPC UA, tipos creados por otro modelo o Companion Specification, o bien creados por el usuario en el mismo modelo de información. Estos objetos, en OPC UA se denominan nodos, y se organizan en una estructura jerárquica llamada "address space" o espacio de direcciones, que proporciona un contexto y una forma de navegar a través de los diferentes elementos del modelo (OPC Foundation 2022b). Existen ocho clases de nodos (NodeClass) en el modelo base de OPC UA (derivadas de Base NodeClass), que se definen en la especificación base de OPC UA y se utilizan para establecer las características y los comportamientos de los nodos en un servidor OPC UA:

- Objeto (Object): Representa una entidad en el sistema, como un dispositivo, una máquina o una aplicación.
- Variable: Representa un valor que puede cambiar en el sistema, como una temperatura, una presión o un estado de un dispositivo.
- Método (Method): Representa una operación o una función que se puede invocar en un objeto.
- Tipo de objeto (ObjectType): Define las propiedades y los métodos que se aplican a un tipo de objeto en el sistema.
- Tipo de variable (VariableType): Define las propiedades de un tipo de variable en el sistema.
- Tipo de referencia (ReferenceType): Define una relación entre dos nodos en el modelo de información del sistema.
- Tipo de datos (DataType): Define un tipo de datos utilizado por una variable o un método.
- Vista (View): Define una vista específica de los nodos en el modelo de información del sistema.

Los nodos tipo *Object* son elementos fundamentales para representar objetos físicos o abstractos en un sistema. Estos nodos encapsulan propiedades, métodos y eventos relacionados con el objeto que representan. Además,este tipo de nodos proporcionan una forma estructurada de organizar y acceder a información, funcionalidad y estados dentro de un servidor OPC UA. Al definir un objeto como un nodo tipo *Object*, se puede acceder y controlar de manera coherente, lo que facilita la gestión de sistemas complejos y la exposición de funcionalidades específicas de manera estandarizada.

Por otro lado, las denominadas "referencias", son creadas mediante nodos ReferenceType, que se utilizan para representar las relaciones jerárquicas y de asociación entre otros nodos. Estos nodos son visibles en el address space y son definidos como instancias de la NodeClass ReferenceType. En la Figura 2.2 se presenta a jerarquía básica de las ReferenceType.

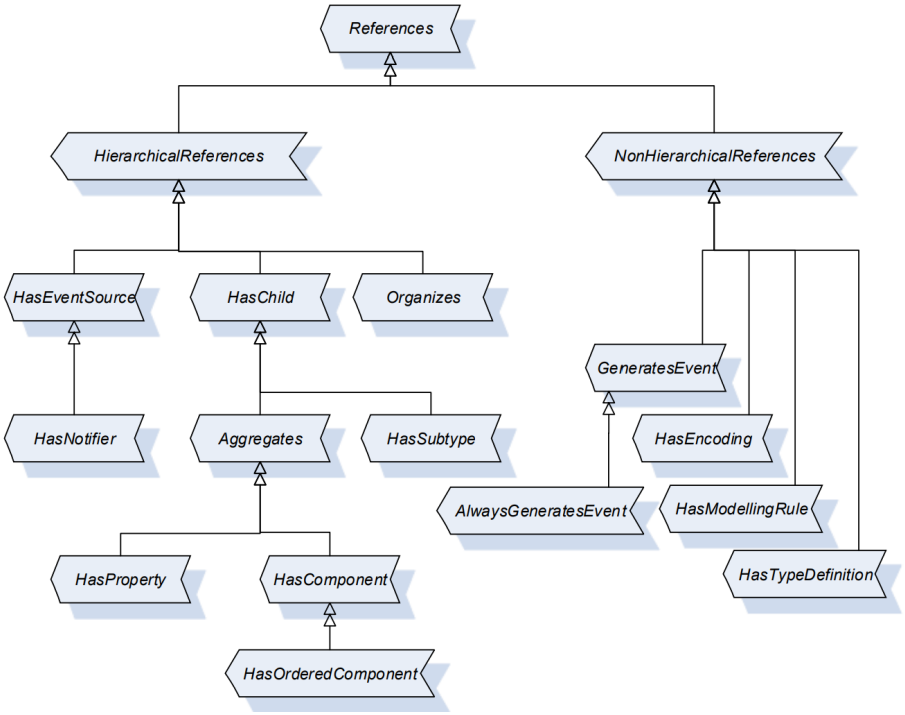


Figura 2.2: Jerarquía de la clase base ReferenceType de OPC UA.

Además, de las técnicas orientadas a objetos, el modelo de información en OPC UA se beneficia de los Companion Specifications. Estas denominadas "especificaciones complementarias", definen cómo se deben representar y describir los objetos, variables y funciones específicas de un dominio industrial o aplicación particular. Las Companion Specifications permiten una mayor semántica y coherencia en la representación de los datos, lo que facilita la interoperabilidad y la comprensión común entre los sistemas que utilizan OPC UA.

La portabilidad a nivel del modelo de información se consigue mediante el uso de namespaces o espacios de nombres. El uso de namespaces permite a diferentes organizaciones definir sus propios conjuntos de nodos y valores de datos dentro del sistema OPC UA sin conflictos al crear su Companion Specification. En la práctica, los namespaces se utilizan a menudo para agrupar nodos y valores de datos relacionados. Por ejemplo, se puede utilizar un namespace para definir todos los nodos relacionados con un equipo específico o un proceso concreto dentro de una planta

de fabricación. De este modo, los clientes y servidores OPC UA pueden identificar fácilmente los nodos y acceder a ellos.

Un namespace se identifica mediante dos elementos: el URI del namespace y un índice del namespace. El URI del namespace es un identificador único global representado como una cadena, normalmente en forma de URL, que identifica a la organización o entidad que define el namespace. Mientras que el índice del namespace es un identificador numérico que se utiliza para hacer referencia al espacio de nombres dentro del entorno OPC UA. Cada namespace del modelo de información debe tener un índice de espacio de nombres único. En el caso de esta proyecto se han utilizado los siguientes namespaces:

Tabla 2.1: Namespaces utilizados en el proyecto

Namespace URI	Índice	Descripción
http://opcfoundation.org/UA/	0	Tipos base de OPC UA
URI del servidor local	1	Depende del servidor local
http://opcfoundation.org/UA/DI/	2	Devices Integation Companion
http://opcfoundation.org/UA/Robotics/	3	Robotics Companion
http://http://ai2.upv.es/UA/UniversalRobots/	4	Modelo de información propio

Además de los nodos de objetos y variables, OPC UA también proporciona soporte para la gestión y el intercambio de eventos. Los eventos en OPC UA permiten la notificación y el seguimiento de condiciones, alarmas y sucesos dentro de un sistema. Aunque no se abordan de manera extensa en este trabajo, es importante mencionar que los eventos en OPC UA pueden enriquecer la información y la interacción entre los sistemas al proporcionar una forma estandarizada de comunicar cambios y situaciones relevantes. Estos eventos pueden ser definidos en las Companion Specifications y se representan como objetos en el address space, brindando así una forma estructurada y escalable de gestionar situaciones que requieren notificación y atención inmediata.

2.3.3 Servicios en OPC UA

Los servicios en OPC UA son la columna vertebral de la comunicación y la interacción en este estándar. Representan las operaciones que pueden ser realizadas por los clientes y los servidores para acceder a la información y las funcionalidades expuestas en el modelo de información. Los servicios en OPC UA se dividen en diferentes categorías según su propósito y funcionalidad. Algunos ejemplos de estos servicios son la lectura y escritura de valores, la suscripción a cambios y eventos, el descubrimiento de nodos y la administración de seguridad. Estos servicios permiten a los clientes solicitar información, notificaciones y acciones específicas a los servidores, estableciendo así una forma estructurada y eficiente de comunicación bidireccional.

Los servicios en OPC UA se utilizan para una variedad de propósitos en la comunicación industrial. Permiten a los clientes acceder a los valores de las variables, obtener información sobre los objetos y las funcionalidades disponibles en el servidor, monitorizar cambios y eventos, y administrar la seguridad en la comunicación. Estos servicios son fundamentales para habilitar la interacción entre dispositivos y sistemas en entornos industriales, ya que proporcionan una forma estandarizada y segura de acceder y controlar la información.

La utilización de servicios en OPC UA es esencial para lograr la interoperabilidad y la eficiencia en la comunicación industrial. Proporcionan una interfaz consistente y bien definida para que los clientes y servidores puedan intercambiar datos y funcionalidades. Al estandarizar la forma en que se solicita y se entrega la información, los servicios en OPC UA simplifican la integración de sistemas heterogéneos y garantizan que los datos se transmitan de manera coherente y comprensible entre dispositivos y aplicaciones.

Para utilizar los servicios en OPC UA, un cliente debe establecer una conexión con el servidor y enviar solicitudes de servicio específicas. Estas solicitudes contienen información sobre el tipo de operación que se desea realizar, así como los parámetros necesarios para llevar a cabo la acción. El servidor procesa la solicitud y responde con los resultados correspondientes. La comunicación entre el cliente y el servidor se basa en mensajes estructurados, que contienen información detallada sobre la solicitud y la respuesta. Cada servicio tiene su propio conjunto de parámetros y protocolos específicos que determinan cómo se realiza la operación. Los servicios más utilizados son los siguientes:

- Servicios de Lectura y Escritura: Estos servicios son fundamentales para acceder y actualizar los valores de las variables en los servidores. El servicio de lectura permite a los clientes obtener los valores actuales de una o varias variables, mientras que el servicio de escritura permite a los clientes modificar estos valores. Estos servicios son ampliamente utilizados para supervisar y controlar el estado de los dispositivos y procesos industriales. Por ejemplo, en un entorno de fabricación, los operadores pueden utilizar los servicios de lectura para obtener información en tiempo real sobre las temperaturas, presiones y niveles de producción, mientras que los servicios de escritura pueden utilizarse para ajustar los parámetros de funcionamiento de las máquinas.
- Servicios de Descubrimiento: Los servicios de descubrimiento en OPC UA permiten a los clientes encontrar información sobre los nodos y objetos disponibles en los servidores. El servicio de navegación (Browsing) permite a los clientes explorar la estructura jerárquica del modelo de información, obteniendo detalles sobre los objetos, variables y funcionalidades expuestas. Estos servicios son esenciales para la visualización y la comprensión del entorno industrial. Por ejemplo, en un sistema de gestión de activos, los operadores pueden utilizar los servicios de descubrimiento para buscar equipos específicos, acceder a sus propiedades y determinar las capacidades que ofrecen.
- Servicios de Suscripción: Permiten a los clientes recibir notificaciones en tiempo real sobre los cambios y eventos que ocurren en el sistema. El servicio de suscripción permite a los clientes monitorizar valores de variables y recibir actualizaciones automáticas cuando dichos valores cambian. Estos servicios son vitales para el monitoreo en tiempo real y la detección temprana de condiciones anormales. Por ejemplo, en un sistema de control de procesos, los ingenieros pueden utilizar los servicios de suscripción para recibir alertas cuando los niveles de presión o temperatura excedan ciertos límites establecidos.
- Servicios de Seguridad: Estos servicios juegan un papel crucial en la protección de la confidencialidad, integridad y autenticidad de los datos intercambiados. Los servicios de autenticación y autorización permiten a los clientes demostrar su identidad y obtener acceso a recursos específicos en el servidor. Los servicios de cifrado y firma digital garantizan que la información transmitida esté protegida contra posibles intentos de intercepción o ma-

nipulación. Estos servicios son esenciales para garantizar la seguridad de la comunicación en entornos industriales. Por ejemplo, en una planta de fabricación, los administradores pueden utilizar los servicios de seguridad para asegurarse de que solo el personal autorizado tenga acceso a los datos y pueda realizar operaciones críticas.

2.3.4 Mapeo y Protocolos de OPC UA

El mapeo (mapping) y la elección de protocolos son aspectos críticos en OPC UA para garantizar la comunicación eficiente, segura y confiable entre sistemas industriales heterogéneos. Estos elementos permiten que los datos sean transmitidos, interpretados y procesados de manera coherente, independientemente de las diferencias en las plataformas, formatos y entornos de los dispositivos y aplicaciones involucrados.

Por una parte, el mapeo de datos en OPC UA se refiere al proceso de convertir la información entre diferentes representaciones y formatos. OPC UA ofrece dos formas principales de mapeo de datos: OPC UA Binary y XML.

- OPC UA Binary: Es el formato de codificación predeterminado y más eficiente en OPC UA. Utiliza un esquema de codificación compacto que minimiza la sobrecarga en la transmisión de datos. Los valores son codificados en binario, lo que reduce el tamaño de los mensajes y mejora la velocidad de transferencia. Este formato es ideal para aplicaciones donde el ancho de banda y la eficiencia son críticos, como en entornos industriales con recursos limitados.
- XML: Aunque menos eficiente en términos de tamaño de datos y velocidad de transferencia en comparación con OPC UA Binary, XML es más legible y comprensible para los seres humanos. OPC UA utiliza XML para representar el modelo de información en archivos de definición y documentos. Esta representación es valiosa para tareas como la configuración y el diseño de sistemas, ya que permite una vista más clara y descriptiva del modelo.

La seguridad es una preocupación central en las comunicaciones industriales. OPC UA implementa protocolos de seguridad para garantizar la confidencialidad, la integridad y la autenticación de los datos transmitidos. Dos protocolos de seguridad clave en OPC UA son WS-SecureConversation y UA-SecureConversation.

- WS-SecureConversation: Este protocolo de seguridad se basa en Web Services y proporciona seguridad a nivel de mensaje. Utiliza firmas digitales y cifrado para garantizar la autenticidad y la confidencialidad de los datos transmitidos. WS-SecureConversation es especialmente adecuado para comunicaciones a través de firewalls y en entornos de Internet, donde la seguridad es esencial para proteger la información.
- UA-SecureConversation: Este protocolo de seguridad está diseñado específicamente para OPC UA y se basa en WS-SecureConversation. Proporciona una capa adicional de seguridad en las comunicaciones, asegurando que los mensajes se transmitan de manera segura entre los sistemas involucrados. UA-SecureConversation ofrece una mayor flexibilidad en términos de autenticación y administración de claves, lo que lo convierte en una opción poderosa para aplicaciones OPC UA.

Por último, los protocolos de transporte en OPC UA determinan cómo se envían y reciben los mensajes entre los sistemas. OPC UA admite varios protocolos de transporte, cada uno diseñado para abordar diferentes requisitos y escenarios.

- UA TCP: Este protocolo de transporte se basa en TCP (Transmission Control Protocol) y es una elección común para la comunicación punto a punto. Proporciona una transferencia confiable de datos y es altamente eficiente en términos de rendimiento. UA TCP es ideal para aplicaciones en redes locales (intranet) y entornos donde la confiabilidad es esencial.
- SOAP/HTTP: Este protocolo de transporte utiliza el protocolo HTTP (Hypertext Transfer Protocol) y SOAP (Simple Object Access Protocol) para la comunicación. Es adecuado para entornos en Internet y a través de firewalls. SOAP/HTTP permite a los sistemas interactuar a través de la infraestructura web, lo que lo hace una elección sólida para aplicaciones que requieren acceso remoto seguro.

2.4 Software de Simulación

En esta sección, se aborda el proceso de selección del software de simulación utilizado para representar el gemelo digital. Se discuten los puntos fuertes y las limitaciones de algunas de las opciones de software más comunes para la creación de gemelos digitales. A continuación, se presenta el software de simulación elegido, CoppeliaSim, y se detallan algunas de sus características y su modo de uso.

2.4.1 Alternativas de Software de Simulación

El primer paso en el proceso de selección del software de simulación ha sido definir las características requeridas para satisfacer los conceptos de los gemelos digitales presentados en la subsección 2.1.2. A continuación, se enumeran algunas de las características clave que se han considerado:

- Representación precisa: El software de simulación debe ser capaz de representar de manera precisa y realista el comportamiento y las propiedades del brazo colaborativo Universal Robots. Esto implica la simulación precisa de la cinemática, dinámica y restricciones del robot.
- Entorno de simulación: En muchos casos los gemelos digitales se usan para verificar la correcta interacción de los robots reales su entorno o con otros robots de una misma célula pro lo que es necesario poder cargar un entorno de simulación realista o incluso varios gemelos digitales.
- Compatibilidad con bibliotecas y herramientas de desarrollo: El software de simulación debe ser compatible con bibliotecas y herramientas de desarrollo (SDK) comunes utilizadas en el proyecto, como la biblioteca open62541 usada en la comunicación OPC UA y la biblioteca RTDE. Esto facilita la integración del gemelo digital con otros sistemas y componentes de la cadena de producción.
- Interfaz gráfica intuitiva: Es importante que el software de simulación cuente con una interfaz gráfica fácil de usar e intuitiva. Esto facilita la configuración de la simulación, la visualización de los resultados y la interacción con el gemelo digital.
- Flexibilidad y personalización: El software de simulación debe permitir la personalización y adaptación a las necesidades específicas del proyecto. Esto implica la capacidad de ajustar

parámetros, agregar componentes adicionales y modificar la lógica de control según sea necesario.

- Multiplataforma: Dado que una de las principales características de OPC UA es su interoperabilidad, es importante seleccionar un software de simulación que sea compatible con múltiples plataformas. La capacidad de utilizar el software de simulación en varias plataformas facilita la integración con otros sistemas y la colaboración entre diferentes usuarios, maximizando así la flexibilidad y el alcance del gemelo digital.
- Funcionalidad: El software de simulación debe ofrecer un entorno de desarrollo que permita la integración de herramientas, plugins y módulos adicionales. Esto posibilita la representación de simulaciones complejas que involucran distintos protocolos de comunicación y diversos tipos de elementos.
- Coste: Puesto que uno de los objetivos de esta tesis es la realización de una aplicación para simular gemelos digitales económica, se buscarán alternativas open source o que dispongan de versiones de prueba para que los usuarios puedan evaluar las capacidades del modelo antes de realizar la inversión.

En base a estas características deseadas, se han evaluado han evaluado varias alternativas de software para la simulación del gemelo digital. A continuación, se presentan los principales puntos fuertes y debilidades de las alternativas más destacadas:

MATLAB Robotics System Toolbox:

- Puntos fuertes:
  - Interfaz familiar e intuitiva para aquellos con experiencia en MATLAB.
  - Incluye muchas funciones y herramientas integradas para aplicaciones de robótica.
  - Buena documentación y soporte disponible de MathWorks.
  - Amplia comunidad de usuarios de MATLAB y recursos disponibles.
- Puntos débiles:
  - Puede ser menos flexible que otras opciones si necesita personalizar la aplicación más allá de las funciones y herramientas proporcionadas
  - Requiere una licencia de MATLAB y la toolbox que puede ser costosa.
  - Puede no tener una comunidad tan grande o tantos recursos disponibles para robótica como las otras opciones.
  - Puede ser complicado de integrar con otros software y entornos.

RoboDK:

- Puntos fuertes:
  - Interfaz fácil de usar, especialmente para usuarios sin experiencia en programación.
  - Buena compatibilidad con muchos robots industriales y cobots.

  - Incluye una amplia biblioteca de modelos y entornos predefinidos para facilitar la configuración de la simulación.

- Puntos débiles:

  - Puede ser menos flexible para la personalización en comparación con otras opciones.
  - Limitado en términos de características y funcionalidades avanzadas en comparación con otras alternativas más especializadas.
  - Requiere una licencia de pago para uso comercial y algunas funciones adicionales pueden requerir costos adicionales.

CoppeliaSim:

- Puntos fuertes:

  - Diseñado específicamente para la simulación y visualización de robótica.
  - Incluye muchos modelos y entornos predefinidos con los que trabajar. Además de la posibilidad de utilizar una gran variedad de plugins.
  - Buena documentación y soporte disponibles, incluyendo tutoriales y ejemplos.
  - Es ampliamente utilizado en la comunidad de robótica y cuenta con una comunidad activa de usuarios.

- Puntos débiles:

  - Menos flexible para la personalización en comparación con otras opciones más genéricas.
  - Puede requerir recursos computacionales más potentes en función de la complejidad de las simulaciones.
  - Puede tener una curva de aprendizaje más pronunciada para aquellos sin experiencia previa en el software.
  - Está centrado en la simulación de robots y entornos y la interfaz con un robot real puede ser más complicada.

ROS:

- Puntos fuertes:

  - Framework altamente flexible y personalizable para aplicaciones robóticas.
  - Gran comunidad y muchos recursos disponibles para el aprendizaje y la resolución de problemas.
  - Compatible con muchos lenguajes de programación, como Python, C++ y MATLAB.
  - Proporciona un conjunto completo de herramientas para el desarrollo de software robótico, incluyendo comunicación, control y percepción.

- Puntos débiles:
  - Curva de aprendizaje pronunciada.
  - Requiere más configuración manual en comparación con otras opciones.
  - Puede no ser la mejor opción si solo se busca visualizar el gemelo digital del brazo UR, ya que agregar ROS puede añadir complejidad adicional.
  - Tiene cierta sobrecarga de rendimiento debido a su sistema de comunicación y paso de mensajes, aunque en este caso específico podría no ser significativo.

Tras el estudio exhaustivo de las distintas alternativas de software de simulación se ha decidido realizar una métrica específica con las características clave expuestas anteriormente. Dicha métrica se presenta en la Tabla 2.3. El uso de una métrica para la toma de decisiones permite llegar a una conclusión objetiva e informada. En base a esta métrica se ha llegado a la elección de utilizar CoppeliaSim como el software preferido para representar el gemelo digital del brazo colaborativo Universal Robots.

Tabla 2.2: Métrica para la elección del software de simulación

Característica	**MATLAB**	**RoboDK**	**CoppeliaSim**	**ROS**	**Peso (%)**
Representación precisa	4	3	4	3	15 %
Entorno de simulación	3	4	4	2	10 %
Compatibilidad SDK	3	2	3	4	15 %
Interfaz gráfica	4	3	4	2	10 %
Flexibilidad	3	3	4	4	10 %
Tiempo real	3	2	4	4	10 %
Funcionalidad	3	2	4	3	10 %
Plataforma	4	3	4	3	10 %
Coste	2	3	3	4	10 %
TOTAL	77.35	68.4	82.55	74.3	100 %

CoppeliaSim ha destacado debido a su enfoque especializado en la simulación y visualización de robótica, lo que garantiza una representación precisa y realista del gemelo digital y su entorno de trabajo. Su amplia biblioteca de modelos y entornos predefinidos proporciona una sólida base para la simulación, permitiendo una configuración eficiente y una experiencia inmersiva.

La disponibilidad de una amplia documentación y soporte, incluyendo tutoriales y ejemplos, ha sido un factor determinante en la elección de CoppeliaSim. Esto facilitará el proceso de aprendizaje y la resolución de problemas durante el desarrollo del gemelo digital, asegurando un progreso fluido y eficaz en el proyecto.

Un aspecto destacado de CoppeliaSim es su reconocimiento y uso extendido en la comunidad de robótica. Esta comunidad activa ofrece un acceso invaluable a recursos adicionales, como foros de discusión, grupos de usuarios y contribuciones de la comunidad. Esta red de apoyo y colaboración

garantiza que los desafíos comunes puedan abordarse de manera efectiva y se pueda compartir conocimiento con otros expertos y desarrolladores.

Además, se ha tenido en cuenta la capacidad de integración de CoppeliaSim con otras herramientas y bibliotecas, como la comunicación OPC UA y la biblioteca RTDE. CoppeliaSim ofrece módulos y plugins adicionales que permiten añadir funcionalidades avanzadas, como cómputo de alto rendimiento, visión artificial o la integración de nuevos protocolos de comunicaciones, lo cual es esencial para el desarrollo integral del gemelo digital. Estas capacidades de integración permiten establecer una conexión fluida y eficiente entre el gemelo digital y otros sistemas y componentes en un entorno de Industria 4.0.

2.4.2 Software de Simulación CoppeliaSim

En esta sección, se explora en detalle el software de simulación elegido, CoppeliaSim. Se describen sus características principales, su interfaz gráfica, el entorno de trabajo y las funcionalidades clave que ofrece para la creación del gemelo digital del brazo colaborativo Universal Robots. Además, se examinan los diferentes enfoques de programación disponibles en CoppeliaSim para la personalización y extensión de la aplicación.

CoppeliaSim presenta una interfaz gráfica intuitiva que facilita la configuración y visualización del gemelo digital. Su diseño bien estructurado y su amplia variedad de paneles y herramientas permiten una navegación sencilla y eficiente.

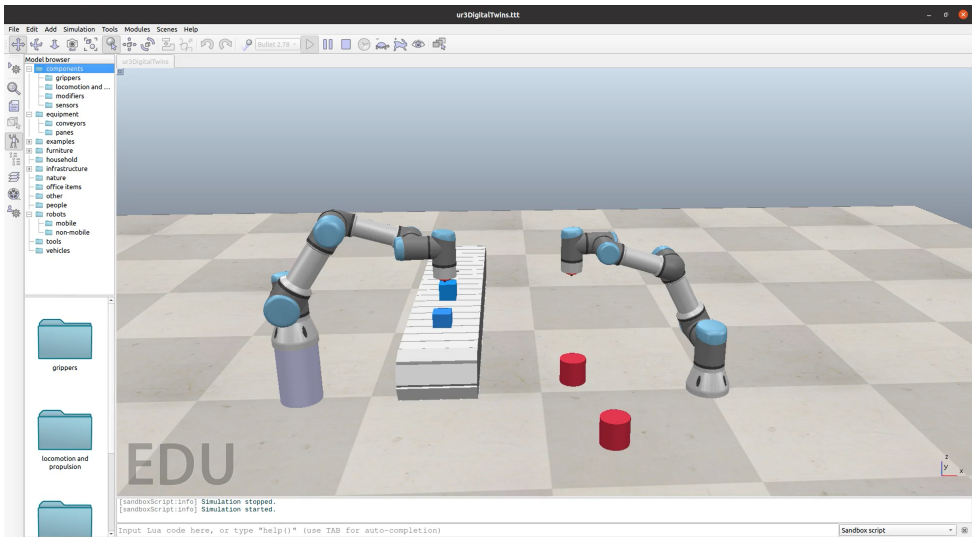


Figura 2.3: Simulación de dos brazos UR3 en CoppeliaSim.

El entorno de trabajo ofrece una amplia gama de características y opciones para la simulación de múltiples tipos de robots. Dentro del entorno, los usuarios pueden configurar y personalizar diversos aspectos de la simulación, como las propiedades físicas, la dinámica del robot y las condiciones ambientales. CoppeliaSim proporciona una representación visual precisa de los elementos simulados, lo que facilita la comprensión y el análisis de los resultados (CoppeliaRobotics 2023). Además, el entorno de simulación permite la interacción con los objetos simulados de manera que los usuarios pueden seleccionar, mover y manipular los modelos 3D, así como definir restricciones

y colisiones entre ellos. También se pueden aplicar efectos visuales y gráficos avanzados, como iluminación, sombras y texturas, para mejorar la apariencia y la realismo de la simulación.

CoppeliaSim ofrece una gran variedad de herramientas para la creación y el uso de modelos en el entorno de simulación. Los usuarios pueden importar modelos 3D existentes o utilizar el conjunto de herramientas de modelado que disponen para la creación personalizada de modelos in-situ desde cero. Estas herramientas incluyen opciones para la creación de geometría, la aplicación de texturas, la definición de cinemática y restricciones, entre otras. Una vez que se han creado o importado los modelos, los usuarios pueden configurar sus propiedades específicas para lograr una simulación precisa del brazo colaborativo Universal Robots. Estas propiedades incluyen características físicas, como masa, inercia y fricción, así como propiedades cinemáticas, como articulaciones, restricciones y colisiones.

Además de la creación y configuración de modelos individuales, CoppeliaSim permite la composición de modelos complejos y la creación de jerarquías de objetos. Esto es especialmente útil para representar sistemas robóticos más complejos que involucran múltiples componentes y subsistemas. Los usuarios pueden agrupar modelos, establecer relaciones entre ellos y crear estructuras en árbol para reflejar la organización y la interconexión de los componentes físicos en el mundo real. Una vez desarrollado el modelo, este se puede exportar o guardar con la escena de CoppeliaSim. Las escenas permiten la reutilización y el intercambio de configuraciones entre diferentes simulaciones además de poder integrar múltiples escenas complejas con varios robots, objetos interactivos y condiciones dinámicas para simular situaciones realistas de células robóticas o partes de la cadena de producción.

Al tratarse de un software de alto nivel, CoppeliaSim cuenta con una amplia gama de funcionalidades disponibles para los usuarios que resultan fundamentales para la simulación del gemelo digital del brazo colaborativo Universal Robots. Algunas de las funcionalidades más destacadas incluyen: cálculos cinemáticos precisos para determinar la posición y orientación del brazo robot en función de sus articulaciones y restricciones, opciones para visualizar y mostrar los datos generados durante la simulación incluso en tiempo real, herramientas para la manipulación y transformación de los datos generados durante la simulación, mecanismos de comunicación y interfaces que permiten la conexión con otros sistemas y dispositivos ...

CoppeliaSim permite también la integración de sensores virtuales en las simulaciones. Esta característica permite simular la operación de una variedad de sensores, como cámaras, sensores de proximidad y otros dispositivos de detección. Los sensores virtuales generan datos que reflejan condiciones del entorno y de los objetos simulados, y estos datos pueden utilizarse para retroalimentar el comportamiento y las decisiones del gemelo digital. Esta capacidad de emular y analizar datos sensoriales es esencial para evaluar la interacción de robot con su entorno y para optimizar su rendimiento en escenarios reales de operación. CoppeliaSim proporciona una plataforma versátil para configurar y ajustar estos sensores virtuales, permitiendo a los usuarios adaptar la simulación a las condiciones y requisitos específicos del gemelo digital.

Por otro lado, el entorno de desarrollo de CoppeliaSim está altamente estructurado y bien documentado. La API del software dispone de hasta 6 enfoques distintos de programación para personalizar y extender la funcionalidad de la aplicación base. Los enfoques que se utilizarán para el desarrollo del gemelo digital del brazo UR son:

- Scripts incrustados

- Plugins
- API Regular de LUA

Utilizando estos enfoques de programación, es posible adaptar y personalizar el entorno base de CoppeliaSim para satisfacer las necesidades específicas del gemelo digital del brazo colaborativo Universal Robots e integrar las bibliotecas necesarias para el desarrollo de la aplicación. En la siguiente sección se entra en detalle en cada uno de estos enfoques.

2.4.3 Programación en CoppeliaSim

El fundamento de la versatilidad de CoppeliaSim reside en su arquitectura modular, donde los componentes interconectados colaboran para habilitar la simulación robótica avanzada. La Figura 2.4 presenta un esquema del framework de CoppeliaSim, resaltando la interacción entre las diferentes partes que lo componen. En los siguientes párrafos se explican las principales partes de este framework.

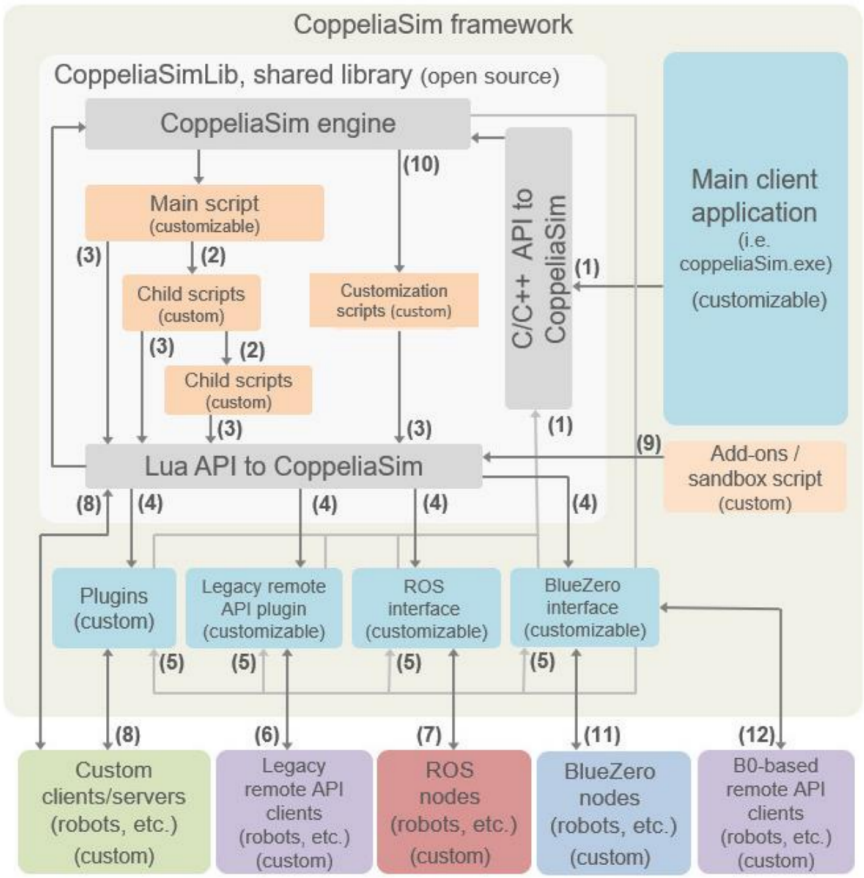


Figura 2.4: Diagrama de la estructura interna del software de simulación CoppeliaSim. Fuente: Coppelia Robotics.

Se podría decir que la "main client application" (aplicación principal del cliente) es la parte principal del software de CoppeliaSim. Se trata de una pequeña aplicación ejecutable que se incluye en el paquete de instalación y desempeña varias funciones clave para el entorno de CoppeliaSim. En primer lugar, ejecuta el simulador utilizando la función "simRunSimulator".

Además, se encarga de cargar y descargar complementos (plugins), cargar los archivos de escena o modelos y manejar la ejecución de simulaciones a través del "Main Script" (Script Principal). Si bien esta aplicación puede personalizarse, se recomienda hacerlo solo cuando el uso de scripts o complementos no sea adecuado, ya que puede haber un alto riesgo de perder la compatibilidad con el comportamiento predeterminado de CoppeliaSim si no se implementa correctamente.

La parte central de la simulación en CoppeliaSim (dejando a un lado los motores de simulación) es el "Main Script". Por defecto, cada escena en CoppeliaSim tiene un Script Principal que contiene el código básico que permite que una simulación se ejecute correctamente. Sin un Main Script, una simulación en ejecución no tendría ninguna funcionalidad. Este script contiene una serie de funciones de callbacks que el sistema llama en momentos apropiados, a excepción de la función de inicialización, que es llamada automáticamente por defecto. El Main Script por defecto se divide generalmente en cuatro callbacks:

- Función de Inicialización (sysCall_init): Esta función se ejecuta una vez al comienzo de la simulación. Se encarga de preparar la simulación y otros aspectos necesarios.
- Función de Actuación (sysCall_actuation): Esta función se ejecuta en cada paso de simulación. Maneja la funcionalidad de actuación de la simulación.
- Función de Detección (sysCall_sensing): Esta función se ejecuta en cada paso de simulación y se encarga de la funcionalidad de detección en la simulación (sensores de proximidad, etc.) de manera genérica.
- Función de Restauración (sysCall_cleanup): Esta función se ejecuta una vez al final de la simulación. Restaura la configuración inicial de los objetos y limpia el estado de los sensores.

Los callbacks pueden ser de dos tipos: callbacks de usuario y callbacks de sistema (como los cuatro mostrados en la lista anterior). Los primeros son llamados directamente por el usuario o un plugin. Por ejemplo, cuando el usuario llama a una función de script en otro script o hace clic en un botón personalizado de la interfaz de usuario. Los callbacks de sistema, por otro lado, son desencadenados por CoppeliaSim en respuesta a eventos específicos, como la inicialización de un script o la transición a otra escena.

Por otro lado, los "Embedded Scripts" (también conocidos como "Child Scripts") son otros tipos de scripts de simulación en CoppeliaSim, con la diferencia de que en cada escena se puede tener un número ilimitado de Child Scripts. Estos scripts representan colecciones de rutinas que manejan funciones específicas en una simulación. Han de estar asociados a objetos en la escena, ya que se guardan y cargan junto con los objetos a los que están asociados. Además, su contenido se duplica cuando se duplica el objeto, lo que permite una gran flexibilidad y capacidad de reutilización en la simulación.

La "API regular" de CoppeliaSim comprende una amplia gama de funciones predefinidas que permiten la interacción programática con el simulador. Estas funciones se agrupan bajo el espacio de nombres "sim" y son accesibles desde los scripts, plugins y otros enfoques de programación. La API regular es extremadamente versátil y cubre muchas áreas, como manipulación de objetos, control de simulación, movimiento de robots y más. El uso de la API regular es especialmente útil cuando se necesita operar dentro del entorno de script de CoppeliaSim o cuando se desean automatizar tareas específicas en la simulación. A través de esta API, se pueden realizar acciones

como mover objetos, cambiar propiedades, acceder a datos de sensores virtuales y controlar la simulación en general.

CoppeliaSim dispone además de 4 tipos de APIs externas que se comunican con framework a través de la API Regular de CoppeliaSim:

- Plugin: Este enfoque se usa a menudo para proporcionar comandos API personalizados a una simulación, a la par del uso de "Embedded Scripts". Otras veces, los plugins brindan a CoppeliaSim funciones especiales que requieren un cálculo rápido (los scripts suelen ser más lentos que los lenguajes compilados), interfaces de hardware específicas (como un robot real) o una comunicación externa particular.
- Cliente de API Remota: A través de esta metodología, una aplicación externa puede conectarse a CoppeliaSim de manera sencilla mediante comandos API remotos. Esto permite la personalización de la simulación y su interacción desde aplicaciones externas.
- Interfaz ROS (Robot Operating System): Mediante esta técnica, una aplicación externa puede conectarse a CoppeliaSim a través del sistema operativo de robots ROS.
- Interfaz BlueZero: Este método permite que aplicaciones externas se conecten a CoppeliaSim utilizando varios medios de comunicación.

De estos enfoques se ha elegido utilizar los plugin por su capacidad de abstracción a bajo nivel y su enorme flexibilidad. Estos plugins se implementan en lenguajes como C++ y se pueden compilar en una biblioteca compartida que CoppeliaSim cargará al inicio. Los comandos API personalizados proporcionados por el plugin se invocarán y funcionarán de manera similar a los comandos de la API regular. En la sección 5.3 se entra en más detalle en el desarrollo y el uso de plugins en CoppeliaSim.

Capítulo 3

Universal Robots y la librería RTDE

Este capítulo se centra en la comunicación con los robots colaborativos de Universal Robots para la adquisición de datos y control de estos. Los cobots de UR ofrecen diversas interfaces de comunicación que les permiten interactuar con dispositivos externos:

- Interfaces Primaria/Secundaria. El controlador UR es capaz de enviar datos del estado del robot y recibir comandos URScript. La interfaz primaria transmite tanto datos del estado del robot como mensajes adicionales, mientras que la interfaz secundaria solo transmite datos del estado del robot. Estos datos son utilizados principalmente para la comunicación entre la interfaz gráfica y el controlador, lo que permite controlar el robot de forma remota sin necesidad de un programa específico.
- Interfaces en tiempo real. Al igual que las interfaces primaria y secundaria, el controlador transmite los datos del estado del robot y recibe comandos URScript. La diferencia principal radica en que en este caso se utiliza un mayor frecuencia de actualización de hasta 500 Hz.
- Panel de control (Dashboard Server).Esta forma de control remoto utiliza comandos sencillos que llegan a la interfaz gráfica a través de sockets TCP/IP. Esta interfaz, permite cargar, reproducir, pausar y detener programas de robot, además de configurar los niveles de acceso del usuario y recibir información sobre el estado actual del robot.
- Comunicación mediante sockets. Los robots UR pueden establecer comunicación con dispositivos externos mediante el protocolo TCP/IP. La transferencia de datos se realiza a través de la comunicación mediante sockets, donde el robot actúa como cliente y el dispositivo externo asume el rol de servidor. URScript proporciona comandos que permiten abrir y cerrar sockets, así como enviar y recibir datos en diferentes formatos.
- XML-RPC. El protocolo XML-RPC se utiliza para la llamada remota a procedimientos (callbacks), empleando el lenguaje XML para transferir datos entre programas mediante la conexión por sockets. El controlador UR puede invocar métodos/funciones (con parámetros) en un programa/servidor remoto y recibir datos estructurados como respuesta. Esta capacidad permite realizar cálculos complejos e integrar de otros paquetes de software adicionales.

- Intercambio de datos en tiempo real (RTDE). La RTDE (Real-Time Data Exchange) se presenta como una alternativa robusta a la interfaz en tiempo real. Esta permite que el controlador UR transmita datos personalizados del estado del robot y acepte puntos de ajuste y registros de datos personalizados. Mientras que esta interfaz tiene la misma frecuencia de actualización que la interfaz en tiempo real nativa del controlador, permite el desarrollo de aplicaciones en lenguajes de alto nivel como Python y C++ mediante la interfaz de programación de aplicaciones (API) de la librería.

Estas herramientas y características del entorno de programación de Universal Robots facilitan la programación de los cobots y permiten a los usuarios implementar soluciones personalizadas de automatización de forma rápida y eficiente. Gracias a su enfoque orientado al usuario, Universal Robots ha logrado llevar la robótica colaborativa a un nivel accesible mejorando la eficiencia, la productividad y la seguridad en los entornos industriales.

3.1 Librería RTDE: Descripción y funcionamiento

La interfaz de Intercambio de Datos en Tiempo Real proporciona una forma de sincronizar aplicaciones externas con el controlador UR a través de una conexión TCP/IP estándar, con una latencia mínima. Esta funcionalidad es útil, entre otras cosas, para establecer la comunicación con el controlador mediante buses de campo (por ejemplo, Ethernet/IP), manipular las E/S del robot y monitorizar el estado del robot (por ejemplo, trayectorias del robot).

Para establecer la comunicación mediante RTDE, el controlador UR utiliza una estructura de cliente-servidor. El controlador actúa como el servidor, transmitiendo los datos del estado personalizado del robot, mientras que los dispositivos externos actúan como clientes, enviando comandos de ajuste y registro al controlador. Al conectarse a la interfaz RTDE, el cliente es responsable de configurar las variables o combinaciones de registros E/S que desea sincronizar. La conexión admite un límite máximo de 2048 bytes para representar la lista de nombres de campos de E/S a los que el cliente desea suscribirse. Cada paquete de entrada que se haya configurado correctamente y no supere el número máximo de bytes, obtendrá un identificador de paquete único, entonteces se da por completada la configuración, y se puede iniciar la sincronización de datos.

Cuando se inicia el bucle de sincronización, el interfaz RTDE envía al cliente los datos solicitados en el mismo orden en que fueron solicitados por el cliente. Además, se espera que el cliente envíe entradas actualizadas al interfaz RTDE cuando haya un cambio de valor. Además, la RTDE proporciona mecanismos de seguridad para garantizar la integridad y confidencialidad de los datos transmitidos. Se utilizan protocolos de cifrado y autenticación para proteger la comunicación entre el controlador UR y los dispositivos externos, lo que garantiza que los datos se transmitan de manera segura y solo sean accesibles para los usuarios autorizados.

La RTDE está disponible de forma predeterminada cuando el controlador UR está en funcionamiento el puerto 30004. Los usuarios pueden utilizar la API que dispone la RTDE para desarrollar aplicaciones personalizadas que aprovechen al máximo la interfaz en tiempo real, brindando una mayor flexibilidad y capacidad de adaptación a los procesos de automatización.

3.2 Comunicación con el brazo robot

3.2.1 Extracción de datos

Para la implementación se a decidido utilizar la API de la RTDE en un sistema externo al robot para ganar flexibilidad en la puesta en marcha de la aplicación. De esta forma resulta mucho más fácil el despliegue de la solución y la depuración de fallos. El lenguaje elegido para la comunicación con el robot es C++ ya que este sera también el lenguaje utilizado en la implementación del servidor OPC UA mediante la biblioteca de open62541. Aunque la API de la librería RTDE está también disponible en Python, C++ presenta importantes ventajas en cuanto a rendimiento, control de recursos y acceso a características de bajo nivel que pueden resultar muy útiles en la implementación del gemelo digital.

La interfaz de la API en C++ se encuentra divida en partes según la funcionalidad buscada tal y como detalla la Guía de Universal Robots (2023). Estas partes son:

- RTDE Control Interface: Esta parte de la interfaz se utiliza para enviar comandos y controlar las acciones del brazo robot. Permite enviar señales de control, configurar movimientos y establecer parámetros específicos.
- RTDE Receive Interface: Se utiliza para recibir y leer datos del estado del brazo robot. Proporciona acceso a información como la posición, velocidad, fuerzas, sensores y otros datos relevantes del robot.
- RTDE IO Interface: Permite controlar y monitorizar los dispositivos de entrada y salida conectados al brazo robot. Permite la comunicación y el control de periféricos externos, como sensores, actuadores y otros equipos auxiliares.
- RTDE Class API: Proporciona una interfaz en C++ para interactuar con las funciones y características principales de RTDE. Permite el acceso a todas las funcionalidades y métodos necesarios para una integración completa con la librería RTDE.
- Script Client API: Se utiliza enviar y recibir comandos y datos utilizando el lenguaje URScript. Permite la comunicación y el control del brazo robot utilizando scripts escritos en URScript directamente desde la aplicación en C++.
- Dashboard Client API: Se utiliza para interactuar con el panel de control (dashboard) del brazo robot. Permite enviar comandos al robot, cargar programas, establecer configuraciones de seguridad y recibir información en tiempo real sobre el estado y el rendimiento del robot.
- Robotiq Gripper API: Interfaz específica para controlar y monitorizar los grippers Robotiq, dispositivos de agarre utilizados en aplicaciones de manipulación y agarre. Permite enviar comandos de apertura y cierre, configurar parámetros y recibir información sobre el estado del gripper.

Puesto que el gemelo digital de esta aplicación está orientado a la monitorización y el registro de los datos para su posterior análisis, la aplicación se centrará en la comunicación unidireccional del robot al servidor. Para ello, sólo es necesario el uso de la interfaz RTDE Receive. De esta forma, se puede conseguir obtener la mayor cantidad de datos manteniendo las propiedades de la adquisición requeridas. No obstante, con el fin de no limitar las funcionalidades del gemelo

digital, se implementará una opción para disponer de la comunicación desde el servidor al robot, aunque esta opción se encuentre por defecto desactivada.

En el contexto del mantenimiento predictivo, se requiere recolectar y analizar una variedad de datos relacionados con el funcionamiento y estado del equipo o sistema que se está monitorizando. Estos datos permiten identificar patrones, anomalías y tendencias que puedan indicar posibles problemas futuros. Algunos de estos datos necesarios incluyen:

- Datos operativos: Información sobre el rendimiento del robot durante su funcionamiento normal. Esto incluye datos como la velocidad de movimiento, aceleración, posición, fuerzas y par aplicados, tiempo de ejecución de tareas, ciclos de operación, entre otros.
- Datos de sensores: Los sensores son fundamentales para monitorizar diferentes aspectos de los robots colaborativos. Por ejemplo, sensores de temperatura, vibración, presión, corriente eléctrica, consumo de energía, entre otros, proporcionan información sobre el estado físico y condiciones de operación.
- Datos de mantenimiento: Registros históricos de intervenciones de mantenimiento preventivo o correctivo, fechas de reemplazo de componentes, ajustes realizados y cualquier otra actividad relacionada con el mantenimiento.
- Datos ambientales: Información sobre las condiciones ambientales en las que opera el cobot, como la temperatura, humedad, y otras variables que puedan afectar su rendimiento y vida útil.
- Datos de fallos históricas: Información sobre fallos y problemas pasados que haya experimentado el equipo, incluyendo la naturaleza de estos, las causas subyacentes y las acciones tomadas para resolverlos.
- Datos de producción: Información relacionada con la producción y carga de trabajo del robot, como el número de ciclos realizados, las tareas completadas, la producción diaria o semanal, etc.
- Datos de calidad del producto: En algunas aplicaciones, puede ser relevante registrar datos relacionados con la calidad del producto final generado por el robot para identificar posibles correlaciones con su funcionamiento.

Dada la distinta naturaleza de los datos que se necesita recoger en las aplicaciones de mantenimiento predictivo, es gran importancia disponer de un gemelo digital altamente flexible que pueda incorporar dispositivos y sensores de distinta naturaleza y fabricantes, lo cual justifica el uso de un estándar de comunicaciones flexible y seguro como es el estándar OPC UA descrito en la sección 2.3.

Centrándose en el objetivo de la tesis que es la obtención de datos del brazo robot para el desarrollo de su gemelo digital, se van a adquirir los datos disponibles de carácter operativo sobre el estado del robot y de sus sensores. Los cobots de Universal Robots disponen de varios sensores de fuerza y par en el extremo del brazo, denominado por sus siglas en inglés TCP (Tool Center Point). Se disponen también de datos de temperatura en las articulaciones, así como de medidas cinemáticas y de consumo energético. A continuación se presenta un resumen de los datos que se pretende obtener:

- Posición de las articulaciones: La RTDE permite obtener la posición angular de las articulaciones del brazo en radianes. La posición se utilizará para replicar los movimientos del brazo robótico real en el software de simulación.
- Velocidad de las articulaciones: Se obtendrá la velocidad angular de las articulaciones en unidades del sistema internacional (rad/s). Aunque la librería RTDE no dispone de ninguna función para la medición de las aceleraciones angulares, el análisis de la velocidad puede ofrecer información suficiente sobre las aceleraciones.
- Corriente y voltaje de los motores: Se obtendrán las corrientes y voltajes de los motores, lo cual permitirá analizar los ciclos de carga de las articulaciones y el consumo de estas, entre otras cosas.
- Temperatura en las articulaciones: Es importante monitorizar la temperatura de los motores de las articulaciones para prolongar su vida útil y detectar posibles anomalías.
- Pose del TCP: Se obtendrán las coordenadas cartesianas de la posición y la orientación del TCP. En los robot de Universal Robots, ambas magnitudes se refieren al punto definido como "Home" en la configuración del robot.
- Velocidad del TCP: Se extraerán las velocidades lineales y angulares referidas al punto definido como "Home".
- Fuerzas generalizadas sobre TCP: Se obtendrán las fuerzas no compensadas en el lado de carga en las coordenadas cartesianas del TCP junto con los momentos producidos por el peso del propio brazo y la carga.

La implementación de las funciones para extraer estos datos de la API se describen en la sección subsección 4.4.2. La descripción detallada del uso de estas funciones puede encontrarse en la documentación de la librería RTDE. [1]

Uno de los mayores problemas surgen al utilizar esta librería es la compatibilidad entre las versiones de la librería instalada en los robots y la utilizada en los programas. Para evitar cualquier tipo de problema de compatibilidad se debe utilizar la API de la misma versión de la librería que se haya instalado en el robot. De lo contrario se encontrarán distintos mensajes de error.

En el caso de la interfaz RTDE Receive, utilizando una versión de la API más reciente que la librería instalada en el robot puede resultar en intentar utilizar funciones no implementadas en dicha librería del robot. Se ha dado el caso por ejemplo de utilizar la versión de Poliscope 1.4.3 en el robot, la cual no dispone de la función para extraer los pares de las articulaciones, la cual está disponible en la versión de la RTDE 1.5.6. Errores más serios se encuentran con el uso de la interfaz RTDE Control. Un error recurrente que indica incompatibilidad entre las versiones del robot y la API son los mensajes del tipo:

> Couldn't connect to RTDE Control!
>
> Unable to find the robot!

[1] `https://sdurobotics.gitlab.io/ur_rtde/api/api.html#`

Estos mensajes indican un error a la hora de conectar desde esta interfaz con el robot. La incompatibilidad de versiones afecta la función *Connect* de la RTDE Control, lanzando un error crítico al intentar establecer la conexión con el robot.

Por lo tanto, para evitar los posibles quebraderos de cabeza a los que pueden llevar este tipo de errores, en el caso de que se deseen utilizar las últimas versiones de la librería, el software del robot debe estar actualizado a alguna de las versiones más recientes.

3.2.2 Control del brazo robot

A pesar de que la aplicación se centra principalmente en la extracción de datos, se ha considerado la implementación de métodos que permitan la comunicación bidireccional entre el servidor y el brazo robot colaborativo, con el fin de mantener la fidelidad de un verdadero gemelo digital. No obstante, se ha decidido diseñar esta parte de la comunicación del gemelo digital al robot físico como una opción, que en un principio se en encontrará desactivada, dejando al usuario la posibilidad de activarla desde una interfaz gráfica para tomar el control del robot. La decisión de habilitar esta interfaz de control surge de dos motivos fundamentales.

En primer lugar, es importante destacar que, el control del brazo robot no es realmente esencial para los objetivos del proyecto, ya que este se enfoca en la monitorización y adquisición de datos del robot. De hecho, a adición de métodos de control en un entorno con recursos limitados, como es el sistema empotrado en el que corre el servidor, podría aumentar la latencia en la toma de datos o sobrecargar el sistema, potencialmente causando fallos.

El segundo motivo se basa en el funcionamiento intrínseco de las interfaces RTDE Receive y RTDE Control. En la mayor parte de aplicaciones industriales típicas, los robots colaborativos de Universal Robots se programan para llevar a cabo tareas especializadas o repetitivas mediante scripts específicos. Estos scripts se ejecutan en el controlador del robot y, en algunos casos, se controlan directamente desde la terminal del robot (Polyscope o teach pendant). En este caso, se podría decir que el control sobre el robot lo tiene el script o si este ha sido lanzado desde el Polyscope, el control lo tendrá este mismo, permitiendo al operador detener o modificar el programa desde el terminal.

En este punto es relevante comprender cómo funciona el control del robot a través de la interfaz RTDE Control. Cuando se establece una conexión entre el servidor y el robot mediante esta interfaz, el robot recibe un nuevo script, que puede estar vacío si no estaba ejecutando un programa en ese momento o contener instrucciones de control introducidas por el usuario desde la API de la interfaz RTDE Control. En cualquier caso, una vez conectado a través de esta interfaz, el controlador cederá el control al nuevo script recibido, lo que significa que cualquier script anterior se olvidará por completo y el robot ejecutará las instrucciones del nuevo script.

Por lo tanto, para mantener la simplicidad y evitar interferencias con el controlador del robot, se han implementado únicamente dos métodos de control que detienen el funcionamiento del brazo robot. El primer método corresponde a una parada de emergencia que detiene inmediatamente el robot y devuelve el control al Polyscope, donde se muestra un mensaje de emergencia que requiere confirmación del operador. El segundo método lleva al robot a su posición de origen, es decir, una posición de cero grados en todas sus articulaciones. Este método simplemente toma el control del robot y envía la instrucción correspondiente para posicionarlo en su estado inicial.

Con la implementación de estos métodos de control, se demuestra la capacidad bidireccional de la comunicación establecida entre el servidor y el brazo robot mediante el uso de la librería RTDE. Estos métodos, junto con la opción de habilitar o desactivar la interfaz de control desde una interfaz gráfica, ofrecen una solución equilibrada que se adapta a los objetivos del proyecto de mantenimiento predictivo. Con la estructura de estos métodos implementados, la aplicación estará preparada para interactuar con el brazo robot de manera segura y eficiente.

Capítulo 4

Comunicación OPC UA

En la primera sección del capítulo, se aborda la configuración y las herramientas necesarias para el desarrollo del modelo de información y la creación del servidor OPC UA, los cuales se consideran los elementos principales de esta aplicación. Los procesos de desarrollo de esta parte han sido llevados a cabo de manera iterativa, fundamentados en una extensa investigación bibliográfica previa sobre los conceptos esenciales para implementar el protocolo OPC UA como puede comprobarse en la planificación del proyecto mostrada en el Apéndice B. Es por ello, por lo que este capítulo presenta una extensión superior al resto.

El diseño del modelo de información y la construcción del servidor OPC UA abordados en este capítulo, han sido aspectos fundamentales del desarrollo, permitiendo establecer la comunicación cliente-servidor. Asimismo, se detallará la implementación del cliente OPC UA, el cual, aunque se ejecuta directamente en el entorno de CoppeliaSim, se presenta en esta sección desde la perspectiva de la comunicación OPC UA. En el próximo capítulo, se profundizará en el desarrollo del cliente desde el punto de vista de CoppeliaSim y su integración con el gemelo digital del brazo robot Universal Robots.

4.1 Consideraciones Previas a la Implementación

En esta sección se tratan los aspectos del hardware que soportará la conexión RTDE y el servidor OPC UA, considerados el núcleo de la aplicación del Gemelo Digital. También es relevante la elección del lenguaje de programación y las herramientas de compilación, que condicionarán el camino por el que se desarrolle la aplicación.

4.1.1 Selección del Sistema Empotrado

Los sistemas empotrados son dispositivos diseñados específicamente para llevar a cabo tareas concretas y suelen contar con características que los hacen ideales para ciertos escenarios, como el control de sistemas y procesos industriales. El desarrollo del servidor OPC UA en un sistema empotrado se eligió debido a diversas razones.

Los sistemas empotrados suelen estar diseñados con hardware y software altamente optimizados para ejecutar tareas específicas. Esto les permite alcanzar un rendimiento superior en comparación con sistemas de propósito general, asegurando una respuesta rápida y eficiente para aplicaciones en tiempo real. Además, suelen tener un tamaño compacto y un bajo consumo de energía, lo que los hace adecuados para aplicaciones donde se requiere un factor de forma reducido y un uso eficiente de los recursos. También tienden a ser más robustos y estables en comparación con sistemas de propósito general, lo que es especialmente importante para garantizar la operatividad continua y segura del servidor OPC UA para la recogida de datos.

En cuanto a la elección del LattePanda Delta v3, esta decisión se basó en una serie de consideraciones que hicieron de este sistema empotrado una opción idónea para el proyecto:

- Rendimiento: El LattePanda Delta v3 cuenta con un procesador Intel Apollo Lake N3350, que ofrece un buen equilibrio entre rendimiento y consumo de energía, lo que lo hace adecuado para ejecutar el servidor OPC UA de manera eficiente y con capacidad para manejar las tareas requeridas.
- Robustez y estabilidad: Esta placa está diseñada para ofrecer una vida útil prolongada y un funcionamiento confiable durante largos períodos de tiempo. Esta durabilidad es esencial para aplicaciones de gemelos digitales y mantenimiento predictivo, donde se espera que el servidor OPC UA esté en funcionamiento continuo para recopilar y analizar datos en tiempo real.
- Conectividad y expansión: El LattePanda Delta v3 ofrece diversas opciones de conectividad, como puertos USB, Ethernet y Wifi, lo que permite establecer una comunicación estable con otros dispositivos y redes industriales. Además, su capacidad de expansión mediante puertos GPIO y otros interfaces proporciona la flexibilidad necesaria para adaptar el sistema a futuras mejoras y necesidades adicionales.

Además de estas razones, el LattePanda Delta v3 trae preinstalado Ubuntu 20.04. Lejos de opiniones personales, desarrollar aplicaciones en Linux en lugar de Windows ofrece numerosas ventajas que hacen que esta plataforma sea preferida por muchos desarrolladores en diferentes campos. Uno de los principales motivos es la naturaleza de código abierto de Linux que permite disponer de una amplia variedad de herramientas y bibliotecas de desarrollo, la mayoría de forma gratuita. Linux destaca también el ámbito industrial para el que, al ser menos vulnerable a virus y malware en comparación con Windows, se considera una opción más segura para proteger los datos y asegurar el funcionamiento ininterrumpido de las aplicaciones críticas industriales.

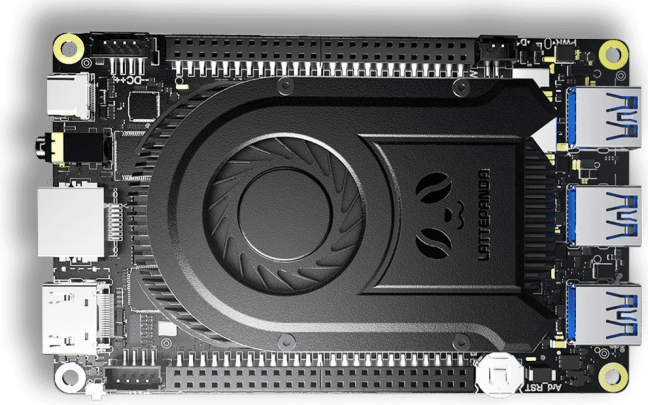


Figura 4.1: Sistema empotrado LattePanda Delta 3. Fuente: lattepanda.com

4.1.2 Selección del Lenguaje de Programación

Habiendo ya seleccionado el sistema empotrado que alojará el servidor, se han de plantear una serie de consideraciones previas al desarrollo del código del servidor OPC UA. Es importante abordar las razones que llevaron a la elección de C++ en lugar C, como lenguaje de programación para esta aplicación. Puesto que muchos proyectos de OPC UA está escritos en C, al comenzar el proyecto, se analizaron los pros y contras de utilizar tanto C como C++ para la implementación del servidor OPC UA en esta aplicación.

En el contexto del desarrollo de servidores OPC UA, ambos lenguajes, C y C++, son opciones viables y ampliamente utilizadas. C es conocido por su eficiencia y bajo nivel de abstracción, lo que lo hace ideal para aplicaciones de tiempo real y sistemas embebidos. Por otro lado, C++ ofrece características adicionales, como la programación orientada a objetos (OOP), que permiten una estructuración más modular y un diseño más elegante y mantenible del código .

En el caso específico de este proyecto, se determinó que C++ era la elección más adecuada para el desarrollo del servidor OPC UA, ya que la necesidad de una estructura más organizada y modular en el programa, así como la facilidad para trabajar con librerías y frameworks modernos eran requisitos necesarios en el proyecto. El enfoque OOP de C++ ha permitido diseñar una jerarquía de clases que facilita la gestión de los componentes del servidor y mejora la usabilidad, claridad y legibilidad del código.

Además, la elección de C++ como lenguaje de programación para el servidor OPC UA ha resultado en una integración más fluida con otras partes del proyecto, especialmente con la librería RTDE utilizada para extraer datos del robot. Esta librería no está disponible en C, por lo que si se hubiera optado por esta opción, habría sido necesario desarrollar lo que se conoce como un "wrapper" para adaptar cada función individual de la librería en C++ al lenguaje C. El proceso de desarrollar wrapper de C++ a C puede ser complicado y tedioso, lo que podría haber supuesto un incremento significativo en la complejidad del proyecto. La elección de C++, que es el lenguaje nativo de la librería RTDE, ha permitido evitar esta complicación y acelerar el desarrollo del servidor OPC UA.

4.1.3 Herramientas de Compilación

Una de las herramientas fundamentales que se utiliza para gestionar el proceso de compilación y construcción de la aplicación ejecutable a partir del código es CMake (*CMake Documentation* 2023). CMake es una herramienta de código abierto que permite generar sistemas de compilación de manera independiente del compilador y del sistema operativo. Proporciona una interfaz altamente flexible para configurar y automatizar el proceso de compilación, facilitando el desarrollo de proyectos complejos y evitando problemas de portabilidad entre diferentes plataformas.

En el contexto del proyecto, CMake se convierte en una elección natural para el manejo del proceso de construcción del servidor OPC UA. Una de las ventajas más destacables es la capacidad de CMake para gestionar las dependencias y las librerías desde el archivo de compilación CMake denominado CMakeLists. Esto proporciona la posibilidad de organizar el código del servidor OPC UA de manera estructurada, lo que es especialmente relevante para el proyecto, ya que implica la integración de varias librerías y múltiples dependencias.

Otra ventaja importante que ofrece CMake es la posibilidad de configurar y personalizar el proceso de compilación según las necesidades específicas del proyecto. Se pueden definir opciones de compilación, activar o desactivar características y establecer variables de entorno, lo que permite adaptar el servidor a diferentes escenarios de uso y configuraciones. De esta forma se pueden generar test específicos para probar el funcionamiento de partes aisladas del código del servidor, conocidos como "Test unitarios". Es posible también crear test para encontrar y corregir errores (debugging tests) mediante herramientas específicas de debugging como *AddressSanitizer* o *Valgrind*, de las cuales se hablará en la siguiente sección.

Se han utilizado además otras herramientas nativas del entorno Linux como *make*. Con ella se compilan los archivos MakeFile generados por el proceso de construcción con CMake. Se han utilizado también bash scripts como el mostrado en la Figura 4.2 para automatizar los procesos de construcción y compilación repetitivos, mejorando el flujo de trabajo en la aplicación.

4.2 SDK OPC UA

La implementación de la comunicación cliente-servidor mediante el protocolo OPC UA con lleva de una serie de pasos para la configuración del SDK (Software Development Kit) del entorno de OPC UA. Las principales herramientas utilizadas en el desarrollo de la comunicación son:

- Biblioteca open62541
- UA-Model Compiler
- UA Model Editor

Es importante destacar que, antes comenzar con la configuración del entorno de desarrollo, se ha realizado una exhaustiva investigación bibliográfica sobre los conceptos fundamentales de OPC UA, así como el uso de las herramientas de desarrollo y bibliotecas asociadas. Esta investigación previa ha sido esencial para adquirir una sólida comprensión de los principios y prácticas necesarios para implementar con éxito el protocolo OPC UA y asegurar un enfoque informado y efectivo en el desarrollo del sistema. En la Figura 4.3 se presentan los pasos seguidos y las herramientas utilizadas en este desarrollo sobe las cuales se hablará a continuación.

```bash
#!/bin/bash

# Verificar si ya estás en la carpeta UR_Digital_Twin
if [[ "$PWD" != *ur-digital-twin* ]]; then
    cd ~/ur-digital-twin
fi

# Si hay un directorio /build ya creado, se borra para compilar limpiamente
if [[ -d build ]]; then
    rm -r build
fi

# Verificar y definir el tipo de compilación según el argumento recibido
build_type="Release"
if [[ "$#" -eq 1 ]]; then
    if [[ "$1" == "Release" || "$1" == "Debug_asan" || "$1" == "Debug" ]]; then
        build_type="$1"
    else
        echo "Argumento no válido. Las opciones válidas son: Release, Debug_asan,
        Debug"
        exit 1
    fi
fi

# Nuevo directorio /build y compilar con el tipo de compilación seleccionado
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE="$build_type" ..
make -j
```

Figura 4.2: Bash script para automatizar el proceso de construcción y compilación del servidor OPC UA.

4.2.1 Biblioteca open62541

Para implementar el protocolo OPC UA, es necesario utilizar una biblioteca que facilite el desarrollo de aplicaciones compatibles con el estándar. El protocolo OPC UA es una arquitectura compleja y altamente estructurada por lo que implementar OPC UA totalmente desde cero sería un proceso laborioso y propenso a errores, ya que implica lidiar con diversos aspectos técnicos, como el manejo de diferentes tipos de datos, encriptación, autenticación, codificación de mensajes, y otros detalles que son cruciales en este tipo de comunicaciones industriales. Es por ello por lo que se utilizan bibliotecas de alto nivel para su implementación.

Las bibliotecas de implementación de OPC UA son conjuntos de código predefinidos que contienen funciones y rutinas para manejar los aspectos más complejos del protocolo OPC UA. Estas bibliotecas suelen ofrecer una interfaz de programación de aplicaciones (API), en algún lenguaje de alto nivel como C++ o Python. Ello permite la abstracción de los detalles técnicos del protocolo, lo que facilita su uso y reduce el proceso de aprendizaje necesario para los desarrolladores. Entre las bibliotecas más utilizadas se encuentran open62541, OPC Foundation UA-.NET, Prosys OPC UA SDK y Eclipse Milo.

Open62541 es ampliamente considerada como una de las mejores bibliotecas para la implementación del protocolo OPC UA debido a diversas razones (open62541 2021). En primer lugar, open62541 es una biblioteca de código abierto y licencia dual (*Mozilla Public License* y *Lesser General Public License*), lo que la convierte en una opción muy atractiva en proyectos de código abierto y comerciales, al ofrecer flexibilidad en términos de licencias. En segundo lugar, open62541 está escrita en lenguajes C y C++, lo que le proporciona un rendimiento eficiente y un bajo consumo de recursos, lo que resulta crítico para muchas aplicaciones industriales.

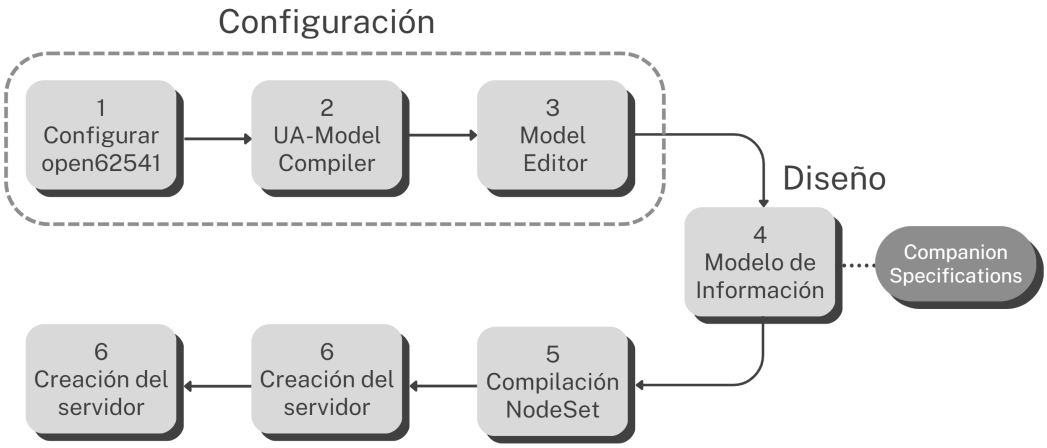


Figura 4.3: Diagrama de etapas del desarrollo de la arquitectura cliente-servidor

Además, esta biblioteca ofrece una amplia gama de características y extensiones del protocolo OPC UA, lo que permite implementar soluciones robustas y avanzadas. Esto es posible gracias a que open62541 cuenta con una comunidad activa de desarrolladores y una base de usuarios sólida, lo que además asegura un soporte continuo, actualizaciones y correcciones de errores. La biblioteca es también compatible con los perfiles de información del cliente y del servidor, permitiendo la interoperabilidad y comunicación entre diferentes dispositivos y sistemas.

Otra ventaja de open62541 es su arquitectura modular, la biblioteca está compuesta por varios módulos independientes que pueden ser utilizados de forma individual o combinada, según los requisitos del proyecto, esto permite que los desarrolladores puedan incluir únicamente las funcionalidades necesarias, evitando cargar el código con características superfluas que no se utilizarán en su proyecto particular, lo que resulta en aplicaciones más livianas y eficientes en términos de recursos y rendimiento. También proporciona una interfaz de programación de aplicaciones (API) clara y bien documentada, lo que facilita el desarrollo y mantenimiento del código (*open62541 repository* 2023).

4.2.2 UA-Model Compiler y Model Editor

Por otra parte, el *UA-Model Compiler* (UA-MC) es una herramienta esencial en el mundo de OPC UA, especialmente cuando se trabaja con modelos de información complejos y estructuras de datos personalizadas. En OPC UA, los modelos de información definen cómo se representan los datos y objetos en el servidor OPC UA, lo que permite una comunicación efectiva entre el cliente y el servidor. Sin embargo, crear estos modelos de información manualmente puede ser una tarea complicada y propensa a errores. Es aquí donde el UA-MC entra en juego, ya que su función principal es simplificar y automatizar el proceso de creación de modelos de información OPC UA. Esta herramienta toma como entrada un archivo de definición de los tipos, generalmente escrito en XML, que describe la estructura de datos y objetos que se desea exponer en el servidor OPC UA. A continuación, el UA-MC genera automáticamente el código necesario en diferentes lenguajes de programación, como C++, Java o C#, para implementar el servidor OPC UA con los objetos y datos especificados. De esta forma se facilita en gran medida la creación de servidores OPC UA, ahorrando tiempo y minimizando errores en la implementación. Además, se

asegura la coherencia y compatibilidad entre diferentes sistemas que utilizan los mismos modelos de información.

Una de las ventajas clave del *UA-Model Compiler* es su capacidad para trabajar con diferentes perfiles OPC UA, lo que permite personalizar la generación de código según las necesidades específicas de una aplicación o industria. Además, esta herramienta se integra perfectamente con otras bibliotecas y herramientas OPC UA, lo que la convierte en una opción poderosa y versátil para desarrolladores que buscan simplificar la implementación de servidores OPC UA (*UA-ModelCompiler repository* 2023).

Por último, los conocidos como *Model Editor*, son unas herramientas que simplifican y agilizan la creación y edición de los modelos de información OPC UA. En lugar de tener que definir manualmente los tipos de datos, nodos y relaciones, esta herramienta proporciona una interfaz gráfica intuitiva para diseñar y configurar los modelos de información. Esto permite a los desarrolladores centrarse en la lógica y funcionalidades específicas de la aplicación, en lugar de preocuparse por los detalles de la implementación de los modelos. Es por ello que una de las principales razones para utilizar *Model Editor* es que aceleran considerablemente el desarrollo de aplicaciones OPC UA. Al proporcionar una representación visual de los modelos, los desarrolladores pueden crear, modificar y validar los modelos de información de manera más rápida y eficiente. Esto no solo ahorra tiempo, sino que también reduce la posibilidad de errores de implementación, lo que se traduce en una mayor confiabilidad del sistema.

Uno de los *Model Editor* más conocidos es "UA Modeler", de Unified Automation. Este software de pago, proporciona una interfaz gráfica especializada que facilita la creación y edición de modelos de información de una manera más intuitiva y centrada en OPC UA. UA Modeler suele ofrecer características avanzadas para el diseño y validación de modelos, lo que puede ser especialmente útil para proyectos con requisitos más complejos y detallados. No obstante, el precio de la licencia y las limitaciones encontradas por algunos usuarios al implementar aplicaciones dependiente en varios Companion Specification han llevado a descartar este software como Model Editor. Así como algunas de las alternativas open source debido a la compatibilidad con los Companion Specification utilizados en esta aplicación.

El software Visual Studio (VS) ofrece una solución completa para el desarrollo de aplicaciones OPC UA. Visual Studio es un IDE potente y ampliamente utilizado en la industria del desarrollo de software, que cuenta con una amplia comunidad de desarrolladores y una gran cantidad de recursos y complementos disponibles. Una de las principales ventajas de VS son las extensiones de terceros, de esta forma, a través de la extensión *OPC UA Model Designer* Visual Studio ofrece un soporte sólido para el desarrollo de aplicaciones OPC UA.

La extensión *OPC UA Model Designer* proporciona un Model Editor integrado y altamente funcional que permite crear, editar y validar modelos de información OPC UA de una manera ágil y eficiente. Además, gracias a la integración con Visual Studio, los desarrolladores pueden aprovechar las capacidades completas del IDE para trabajar en todos los aspectos del proyecto, desde el desarrollo del modelo hasta la implementación del código y la depuración ofreciendo un flujo de trabajo cohesionado y una mayor productividad para el desarrollo de aplicaciones OPC UA.

4.3 Implementación del Modelado de Información

Como ya se explicó en la sección 2.3.2, en general, la implementación de OPC UA implica la creación o uso de un modelo de información, en el que se configuren los objetos y variables que se proporcionarán a los clientes OPC UA, la configuración de los permisos de acceso a los datos y la definición de los protocolos de seguridad que se utilizarán para proteger los datos. También se deben configurar los protocolos de red y los mecanismos de transporte utilizados para la comunicación entre el servidor y los clientes.

Cualquier Companion Specification o modelo de información ha de ser distribuido en un archivo de formato NodeSet2.xml. El formato NodeSet2.xml se basa en el lenguaje de marcado XML y se utiliza para definir los nodos, referencias, atributos y propiedades que componen el modelo de información. El formato XML proporciona una forma estandarizada de representar el modelo de información, lo que flexibiliza el intercambio de información entre diferentes sistemas y aplicaciones (*UA Nodeset repository* 2023).

Para generar los archivos NodeSet2.xml es necesaria alguna herramienta de conversión como el UA-MC, sobre el que se ha hablado en la sección anterior, que permite programar un modelo de información en XML u otro lenguaje de alto nivel como C++ El UA-MC es la herramienta oficial mantenida por la *OPC Foundation* para generar archivos NodeSet2.xml y como no tiene interfaz gráfica de usuario, es necesario generar el archivo del modelo de información en formato XML con un editor de texto u otra IDE, en este caso Visual Studio.

4.3.1 Devices y Robotics Companion Specifications

La asociación alemana *Mechanical Engineering Industry Association* (VDMA) mantiene el OPC UA Robotics Companion Specification (RCS) desde el 2019, año en el que se publicó la primera parte del Companion Specification. En esta primera parte se incluye una descripción básica del sistema de movimiento de los dispositivos y tiene como principal objetivo transmitir los datos de estado de un sistema de dispositivos de movimiento a sistemas de fabricación de nivel superior (por ejemplo, PLC de línea, MES, cloud...) con fines de información y diagnóstico. En la Figura 4.4 se presenta el diagrama de bloques de la estructura propuesta en la Parte 1 del RCS. En este diagrama se observan los principales tipos en los rectángulos en negrita, sus principales nodos hijo y algunas de las relaciones con el resto de nodos del modelo. En el Apéndice 1 se puede consultar una leyenda del significado de los símbolos estandarizados en los diagramas de OPC UA.

El Robotics Companion Specification proporciona una estructura bien definida y estandarizada para representar información relacionada con robots y sistemas de automatización en OPC UA. Esto garantiza que los desarrolladores puedan crear modelos de información coherentes y consistentes, independientemente de la plataforma o fabricante del robot y permite una comunicación más eficiente y sin ambigüedades entre los clientes y los servidores OPC UA que interactúan con los robots. La estructura de nodos en el Robotics Companion Specification sigue una jerarquía lógica, que facilita la navegación y organización de la información. Algunos nodos clave en la estructura incluyen:

- Nodo raíz: Representa el nodo raíz del modelo de información, este nodo se incluye por defecto en todo modelo de información. El tipo de nodo corresponde con un carpeta que alberga el

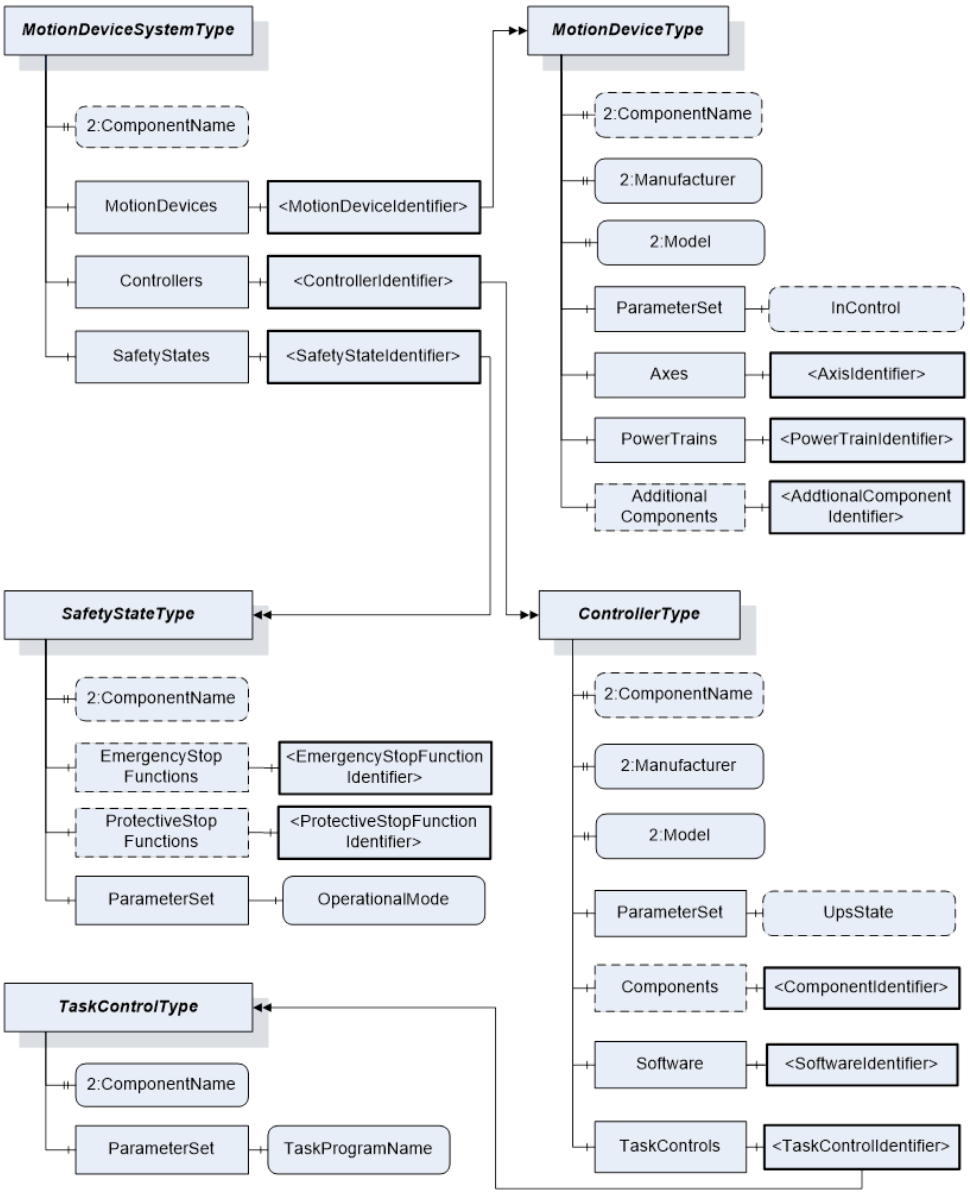


Figura 4.4: Estructura del Robotics Companion Specification. Fuente: OPC Foundation, VDMA

resto del modelo de información. Este nodo se denomina siempre *UA_NS0ID_OBJECTSFOLDER*, y es usado como *MotionDeviceSystem*: Es el primer nodo de cada instancia de robot o máquina. Organiza las principales partes de cada instancia de robot: elementos de movimiento, controladores y estados de seguridad.

- Elementos de movimiento (*MotionDevices*): Contienen información general sobre los sistemas físicos móviles, bien sean robots, ejes externos u otros. Cada elemento tiene como atributos: descripción del fabricante y modelo. Los objetos hijo de los Ejes (Axes) y Sistemas de Potencia (*PowerTrains*), definen las articulaciones de los robots y los motores (u otro sistema de actuación) respectivamente.
- Controladores (*Controllers*): Estos nodos definen las unidades de control de los sistemas físicos de movimiento. Los objetos hijo de este nodo especifican características el software

y Tareas del controlador (*TaskControl*). Estas últimas describen un los hilos de ejecución del controlador capaces de ejecutar programas.

- Estados de Seguridad (*SafetyStates*): Describe los estados de seguridad de los dispositivos de movimiento y controladores. Un sistema de dispositivos de movimiento puede estar asociado a una o varias instancias del *SafetyStateType*. Estos objetos permiten representar medidas de seguridad definidas de forma distinta por distintos fabricantes de una forma estandarizada.

Además, el RCS define referencias adicionales entre los nodos para establecer relaciones entre ellos. Algunas de estas referencias son:

- Controla a (*Controls*): Utilizada para establecer la relación de control de un nodo controlador sobre otro u otros. De forma similar puede definirse la relación en sentido inverso: *IsControlledBy*.
- Mueve a (*Moves*): Establece una relación de movimiento entre un nodo del tipo Sistema de Potencia y nodos tipo Ejes desde el punto de vista del nodo *PowerTrain*. Este nodo debe tener una referencia de este tipo con todos los ejes que mueve de forma única. La referencia inversa desde la perspectiva del eje es *IsMovedBy*.
- Es Conducido por (*IsDrivenBy*): Describe la dependencia entre un objeto que controla a otro en una relación maestro-esclavo como puede ser un motor electrico (*IsDrivenBy*) y un driver o controlador.
- Está Conectado a (*IsConnectedTo*): Relaciona dos objetos montados uno en el otro o conectados mecánicamente.
- Tiene Estados de Seguridad (*HasSafetyStates*): Se utiliza para mostrar qué objeto (controlador) es responsable de la ejecución de la funcionalidad de seguridad.

Por otra parte, el RCS está a su vez basado en el Devices Companion (DI), el cual también se utilizará en la creación del modelo de información del gemelo digital de los brazos robóticos UR. El DI es una especificación que define cómo se deben modelar y comunicar los dispositivos en el entorno industrial. En esencia, es una guía que estandariza la representación de dispositivos y sus capacidades a través de OPC UA. Este Companion permite que los dispositivos, como sensores, actuadores, controladores y otros equipos industriales, sean descritos de manera uniforme y consistente, lo que facilita su integración en sistemas de control, supervisión y gestión.

Este Companion es muy utilizado en la integración de sistemas del Internet de las cosas (IoT) y la automatización industrial ya que facilita la integración de dispositivos y software de distintos fabricantes ofreciendo la posibilidad de adaptar y personalizar tareas u operaciones. Puesto que brinda la posibilidad de que los dispositivos comparten información detallada sobre su estado y sus condiciones de funcionamiento, es también frecuentemente utilizado en la supervisión y el mantenimiento predictivo de equipos. Por esta razón se utiliza como base para el RCS y muchas aplicaciones de sistemas distribuidos y tiempo real como es la desarrollada en esta tesis.

El uso de la estructura y referencias definidas por el Robotics y el Devices Companion Specifications asegura que los datos del robot sean accesibles de manera coherente y estandarizada, lo que simplifica la integración de los robots con otros sistemas de automatización y permite una colaboración más eficiente en los entornos altamente conectados de la Industria 4.0.

4.3.2 Modelo de Información Personalizado

El modelo de información personalizado realizado para crear la estructura del servidor que contenga las variables necesarias expuestas en la sección subsección 3.2.1, comienza declarando los atributos mínimos necesarios para crear la base del modelo de información. En la Figura 4.5 a continuación se presenta el comienzo del fichero XML del modelo de información.

- *xmlns:uax* es el namespace se define para hacer referencia al namespace de tipos de OPC UA, que contiene la definición de todos los tipos de datos integrados de OPC UA.
- *xmlns:xsi* este namespace hace referencia a la Instancia de esquema XML, que se utiliza para especificar la estructura del esquema para el archivo XML.
- *xmlns:OpcUa* es prefijo que se utiliza para el namespace de OPC UA, que contiene la definición de todos los elementos específicos de OPC UA.
- *xmlns:DI* es el prefijo para referenciar al namespace OPC UA Device Integration Companion Specification.
- *xmlns:ROB* hace referencia al namespace OPC UA Robotics, que contiene la definición de elementos del RCS.
- *xmlns:ROB_UR* namespace propio creado para ampliar las características del RCS adaptado a los robots de Universal Robots.
- *xmlns:xsd* es el namespace XML Schema, que se utiliza para especificar la instancia de esquema para el archivo XML.
- *TargetNamespace* y *TargetXmlNamespace* especifican el espacio de nombres XML de destino del modelo de información.
- *TargetVersion* y *TargetPublicationDate* especifican la versión y fecha del modelo de información respectivamente.
- *xmlns* es el espacio de nombres por defecto de un modelo de información, que es el espacio de nombres en el que se definen todos los elementos.

En la siguiente sección del código del modelo de información mostrado en la Figura 4.5, se definen los namespaces que se van a utilizar. Las direcciones web de estos namespaces se presentaron en la Tabla 2.1. Además del nombre, prefijos y dirección web de los namespace (también se puede referenciar el namespace indicando la ruta al fichero de tipos), se ha de indicar la ruta en el sistema donde se encuentra el fichero NodeSet2.xml del namespace.

A continuación, el código comienza la definición del tipo de objeto *RobotType*, el cual se utilizará para crear las instancias de los brazos robóticos. Las instancias de los cobot pueden crearse desde el modelo de información como muestra la Figura 4.6 o bien desde código en el programa del servidor. La ventaja de la creación de instancias de forma programática es que no es necesario compilar de nuevo el modelo de información y estas instancias pueden ser editadas en el proceso de creación mediante el uso de funciones de la librería open62541.

El tipo de objeto *RobotType* se ha definido como un tipo de *MotionDeviceSystem* declarado en el RCS. Este tipo, es el tipo base que contiene el resto de tipos de objetos y referencias del RCS. El

```
<ModelDesign
    xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:OpcUa="http://opcfoundation.org/UA/"
    xmlns:DI="http://opcfoundation.org/UA/DI/"
    xmlns:ROB="http://opcfoundation.org/UA/Robotics/"
    xmlns:ROB_UR="http://ai2.upv.es/UA/Robotics/UniversalRobots/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    TargetNamespace="http://ai2.upv.es/UA/Robotics/UniversalRobots/"
    TargetXmlNamespace="http://ai2.upv.es/UA/Robotics/UniversalRobots/"
    TargetVersion="1.0.0"
    TargetPublicationDate="2023-05-10T00:00:00Z"
    xmlns="http://opcfoundation.org/UA/ModelDesign.xsd">

    <Namespaces>
        <Namespace Name="UniversalRobots" PublicationDate="2023-05-30T00:00:00Z"
          Prefix="UniversalRobots" XmlNamespace="http://ai2.upv.es/UA/Robotics/UniversalRobots/
          Types.xsd" XmlPrefix="UniversalRobots">http://ai2.upv.es/UA/Robotics/UniversalRobots/</
          Namespace>
        <Namespace Name="OpcUa" Version="1.04" PublicationDate="2019-05-01T00:00:00Z"
          Prefix="Opc.Ua" InternalPrefix="Opc.Ua.Server" XmlNamespace="http://opcfoundation.org/
          UA/2008/02/Types.xsd" XmlPrefix="OpcUa">http://opcfoundation.org/UA/</Namespace>
        <Namespace Name="OpcUaDi" Prefix="Opc.Ua.Di" XmlNamespace="http://opcfoundation.org/UA/DI/
          Types.xsd" XmlPrefix="DI" FilePath="OpcUaDiModel">http://opcfoundation.org/UA/DI/</
          Namespace>
        <Namespace Name="OpcUaRobotics" Prefix="Opc.Ua.Robotics" XmlNamespace="http://
          opcfoundation.org/UA/Robotics/Types.xsd" XmlPrefix="Robotics" FilePath="deps/Robotics/
          OpcUaRoboticsModel">http://opcfoundation.org/UA/Robotics/</Namespace>
    </Namespaces>

    <!-- ### RobotType ###-->
    <ObjectType SymbolicName="ROB_UR:RobotType" TypeDefinition="ROB:MotionDeviceSystemType">
        <Description>Root UR Robot object.</Description>
        <References>
            <Reference IsInverse="true">
                <ReferenceType>OpcUa:Organizes</ReferenceType>
                <TargetId>DI:DeviceSet</TargetId>
```

Figura 4.5: Comienzo de la implementación del modelo de información en XML.

tipo de robot es el objeto padre para el resto de objetos que constituyen el robot. A continuación se describen brevemente los principales objetos y variables que conforman el tipo de robot:

- Robot Arm (*ROB_ UR:RobotArm*): Este es el objeto hijo de Motion Devices que representa el brazo robótico físico del robot UR. Contiene referencias a objetos que representan los ejes y el sensor TCP (Tool Center Point) asociados al brazo robótico.
    - Axis J1, Axis J2, etc. (*ROB:AxisType*): Cada uno de estos objetos representa un eje específico del brazo robótico y está etiquetado con su nombre de eje correspondiente. Todos ellos están organizados bajo el objeto Axes. Cada uno de estos objetos contiene variables que almacenan información específica sobre cada eje, como la posición actual (*ActualPosition*), velocidad actual (*ActualSpeed*) y torque actual (*ActualTorque*). También contiene referencias a objetos que representan el *PowerTrainType* asociado a cada eje.
    - TCP (*ROB_ UR:Tcp*): Representa el Punto Central de Herramienta (TCP) del brazo robótico y contiene variables que representan información sobre el TCP, como la posición y orientación en radianes (*TcpPosition*), la velocidad lineal (*TcpLinearSpeed*) y la velocidad angular (*TcpAngularSpeed*). Además, contiene objetos que representan los sensores de fuerza (*TcpForceSensor*) y torque (*TcpTorqueSensor*) asociados al TCP.

```
<Object SymbolicName="ROB_UR:RobotUR5" TypeDefinition="ROB_UR:RobotType"
  ModellingRule="Mandatory">
<Description>Root UR Robot object.</Description>
<BrowseName>RobotUR5</BrowseName>
<References>
    <Reference IsInverse="true">
        <ReferenceType>OpcUa:Organizes</ReferenceType>
        <TargetId>DI:DeviceSet</TargetId>
    </Reference>
</References>
</Object>
```

Figura 4.6: Creación de una instancia de brazo robot en el modelo de información.

- TCP Force/Torque Sensor (*ROB_ UR:TcpForceSensor/TcpTorqueSensor*): Son los objetos hijos del objeto TCP, que representan el sensor de fuerza y par asociados al TCP del brazo robótico. Cada uno de ellos contiene un array con las direcciones X, Y, y Z de fuerzas/momentos sobre el TCP.

- Power Trains (*ROB:PowerTrains*): Representa la colección de trenes de potencia del sistema de robótica. Contiene el conjunto de componentes eléctricos y mecánicos que proporcionan la potencia y controlan el movimiento de diferentes partes del robot. Estos componentes son representados con subobjetos que representan trenes de potencia individuales (PowerTrain1, PowerTrain2, PowerTrain3, PowerTrain4, PowerTrain5 y PowerTrain6).
  - Power Train 1, Power Train 2, etc. (*ROB:PowerTrainType*): Representan los trenes de potencia individuales en el sistema de robótica. Cada uno contiene un subobjeto llamado "Motor", que representa un motor eléctrico específico asociado con este tren de potencia, por ejemplo, el Power Train 1 contiene el objeto Motor1 asociado.
  - Motor J1, Motor J2, etc. (*ROB:MotorType*): Este objeto representa un motor eléctrico específico asociado con uno de los Power Train. Contiene las variables que almacenan información específica del motor, como el número de serie, el fabricante, el modelo y el código de producto. También contiene un subobjeto llamado *ParameterSet*, que agrupa variables relacionadas con los parámetros del motor como son la corriente actual del motor (*MotorCurrent*), la temperatura del motor (*Temperature*) y el voltaje actual del motor (*MotorVoltaje*). Cada uno de estos motores está referenciado a uno de los objeto del tipo AxisType (*ROB:AxisType*).

- UR Controller (*ROB_ UR:UrController*): Hijo del objeto Controllers, hijo a su vez de Motion Devices. Este objeto representa el controlador específico del robot de Universal Robots. No contiene hijos asociados por no ser de utilidad en esta aplicación.[1]

- Safety States (*ROB:SafetyStates*): Este objeto contiene los estados de seguridad asociados con el robot, representados como variables bajo un objeto *DI:ParameterSet*. Estas variables son Parada de emergencia (ROB:EmergencyStop), Parada inmediata (ROB:ProtectiveStop). Ambas variables a diferencia de las 'presentadas anteriormente tienen el atributo de *AccessLevel* establecido como *ReadWrite*. Esto permitirá implementar la comunicación bidireccional desde el cliente al servidor.

[1] Tanto a este, como al resto de tipos heredados del RCS pueden añadirse los nodos hijos de manera programática aunque no figuren como *Mandatory* en el modelo de información

Por último se han modelado los sensores de fuerza y par que se localizan en el TCP del brazo. Estos sensores se han modelado a partir de la creación de un tipo de objeto padre denominado *SensorType*. Este objeto hereda los atributos del tipo de objeto *ComponentType* del companion DI. Con ello se han creado los dos tipos de objetos: *ForceSensor* y *TorqueSensor*. Cada uno de ellos contiene una variable de tipo base de OPC UA *ThreeDVector*. Este tipo crea un vector de tres dimensiones (X, Y, Z) y lo gestiona automáticamente como nuevos nodos hijos.

El modelo de información desarrollado es un sistema completo capaz de representar de forma organizada brazos robot colaborativos de Universal Robots. Este modelo ha sido diseñado y construido desde cero en un proceso que ha tomado un mes de trabajo exhaustivo y dedicado. El modelo se basa en estándares como OpcUa y utiliza una estructura jerárquica con objetos y propiedades cuidadosamente definidas para capturar todos los aspectos esenciales del robot y sus componentes. Desde los estados de seguridad hasta los ejes de movimiento y el sensor TCP, cada elemento clave del robot ha sido meticulosamente representado y enlazado a su contexto relevante. Además se ha estructurado de forma que sea posible añadir nuevas funcionalidades al brazo robot o nuevos sensores externos conectados a este. La estructura completa y detallada del modelo de información puede consultarse en el Apéndice A.

4.4 Implementación del servidor OPC UA

Tal y como muestra la Figura 1.1, uno de los componentes principales de la aplicación es el servidor OPC UA y el programa que lo implementa en el sistema empotrado. La implementación del servidor como muestra esta figura, necesita de los archivos NodeSet2 que contienen los distintos espacios de nombres que se utilizarán en el programa del servidor. Además es necesario tener las herramientas del SDK de OPC UA descritas en la sección anterior instaladas, siendo de especial importancia la biblioteca open62451. Aunque no se ha mencionado explícitamente hasta ahora, por ser uno de los requisitos necesarios para la correcta ejecución de la biblioteca open62541, también se ha de disponer de la biblioteca *Boost* para C++.

En las siguientes secciones se presentarán de manera estructurada las distintas partes que componen el programa programa del servidor, en el cual, como ya se ha comentado en anteriores ocasiones, se implementa tanto la extracción de datos del robot, como la creación, configuración y gestión del servidor.

4.4.1 Estructura del programa del servidor OPC UA

La estructura del programa se compone de cuatro componentes fundamentales para su funcionamiento: la función main, la clase ServerManager, la clase RobotManager y las estructuras de datos compartidas. Como es común en los programas escritos en lenguaje C++, la función principal (main) actúa como el punto de entrada al programa. Esta función coordina la inicialización de las variables y estructuras de datos y la creación de los hilos (threads) para la extracción de datos de los robots y la gestión del servidor. Es también responsable de procesar los argumentos de la línea de comandos para configurar el comportamiento del servidor y conectar los robots especificados. Para lanzar el programa del servidor, se espera que el usuario introduzca el numero de robots, un nombre único para cada robot y su dirección ip en los argumentos de la línea de comandos. De esta forma se aprovechan la estructura del modelo de información y las

funcionalidades de la biblioteca open62451, para poder crear tantas instancias de robots como se desee.

Para permitir la comunicación con múltiples robots simultáneamente, el programa implementa multithreading utilizando la librería estándar de C++. Cada robot se maneja en un hilo dedicado, lo que permite una extracción de datos concurrente. Para garantizar el acceso seguro a recursos compartidos, como los datos de los robots, se utilizan semáforos que controlan la sincronización entre los hilos. La clase "RobotManager" es la utilizada en cada uno de los thread de extracción de datos. Esta se encarga de establecer la comunicación con cada robot UR, extraer datos en tiempo real con la interfaz RTDE Receive y manejar situaciones de emergencia con la interfaz RTDE Control. Cada robot se representa mediante una estructura de datos denominada "RobotData", que contiene los datos y mecanismos de sincronización para cada robot. De esta forma, el thread de cada robot, en cada ciclo de lectura, escribe en su instancia de estructura RobotData los datos obtenidos del cada robot.

En la función principal se crea también un único thread para gestionar el servidor, el cual puede acceder a todas las instancias de las estructuras RobotData. La clase "ServerManager" es la utilizada en este thread de control del servidor. Esta clase es la encargada de gestionar la creación y configuración el servidor OPC UA, par a lo cual es necesario cargar los modelos de información y crear los nodos necesarios para cada instancia de robot. Además, la clase gestiona la sincronización del tiempo entre el servidor y los robots, y la actualización de los datos extraídos en el servidor. De forma que la clase lee los datos actualizados en cada una de las estructuras de datos compartidas RobotData y los escribe en los nodos correspondientes de la instancia de robot en el servidor. En el esquema a continuación, se presenta la extracción de los datos y escritura en el servidor mediante los thread tal y como se ha explicado anteriormente.

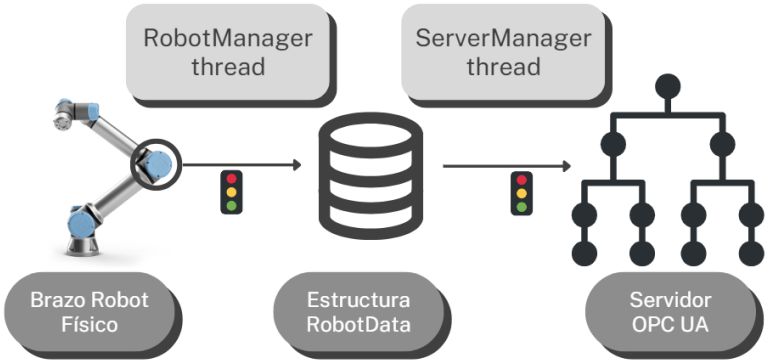


Figura 4.7: Esquema resumen del funcionamiento multithreading del programa del servidor.

4.4.2 Clase RobotManager

La clase RobotManager tiene la responsabilidad de establecer comunicación con cada uno de los robots UR individuales, extraer datos en tiempo real y gestionar señales de emergencia. Para lograr esto, implementa una serie de métodos, cada uno dedicado a la extracción de un tipo específico de variable de los robots. En método principal es el denominado *extractDataFromRobot*. Este método es el punto de entrada de los thread dedicados y recibe un puntero a las estructuras de datos compartidas. Este mismo puntero se pasará al resto de métodos que implementan la extracción de datos. Estos son llamados continuamente en el bucle continuo de extracción de datos. En cada iteración del bucle, se recuperan las posiciones articulares, velocidades articulares, corrientes y voltajes de los motores, posición TCP, velocidad TCP y fuerza generalizada en el TCP. En la Figura 4.8 se presenta un diagrama de flujo resumiendo el funcionamiento de los thread creados con la clase RobotManager.

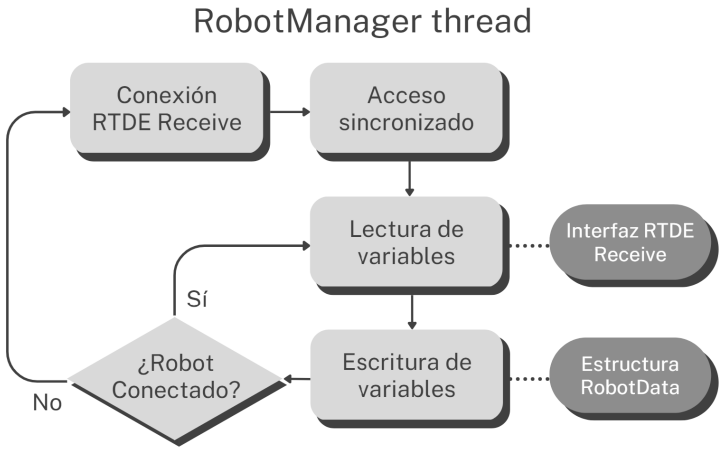


Figura 4.8: Diagrama de flujo resumiendo el funcionamiento de los thread creados con la clase RobotManager.

Además de la extracción de los datos del robot, en esta clase se implementan los métodos para el control del robot. El sentido de esta comunicación es desde el servidor hacia el robot, opuesto a la extracción de datos. Aunque, como se ha comentado en el capítulo de extracción de datos, la interfaz RTDE Control dispone de una amplia gama de comandos para el control del robot, en esta aplicación se he decidido implementar únicamente la parada de emergencia y la puesta a cero del robot. Estos métodos son llamados desde el bucle principal cuando se da la condición de disparo, la cual no es más que flag booleanos miembros de la estructura RobotData. Esta implementación no es óptima en el caso de que se deseen añadir más métodos para el control del robot ya que la comprobación de los flag y la ejecución de los métodos en el bucle de principal puede lastrar la actualización de los datos. En este caso debería implementarse un nuevo thread hijo que utilice flag booleanos atómicos (variables de la librería atomic[2] para C++).

[2] Para implementar variables atómicas se recomienda consultar: blog.devgenius.io

4.4.3 Clase ServerManager

La clase ServerManager (Figura 4.9) es la encargada de gestionar el servidor OPC UA. Esta clase tiene un método principal llamado *run* que sirve como punto de entrada para el thread del servidor. Este método toma un puntero a un vector de estructuras RobotData como argumento para tener así acceso a las estructuras compartidas de todos los robots. Primeramente, el método *run* inicializa el servidor OPC UA con la configuración mínima y crea los nodos OPC UA necesarios basados en los namespace y archivos NodeID proporcionados. Es muy importante que estos archivos figuren como dependencias en el programa del servidor. A continuación, se agregan las instancias de robots para cada robot recibido en el vector de estructuras RobotData. Luego se realiza una búsqueda de los nodos en los que se desea escribir mediante el método iterativo *browseForNodesRecursive*, que implementa las funciones de browsing de open62541. Las IDs de los nodos encontrados se guardan en vectores en una estructura adicional denominada NodesIdsList.

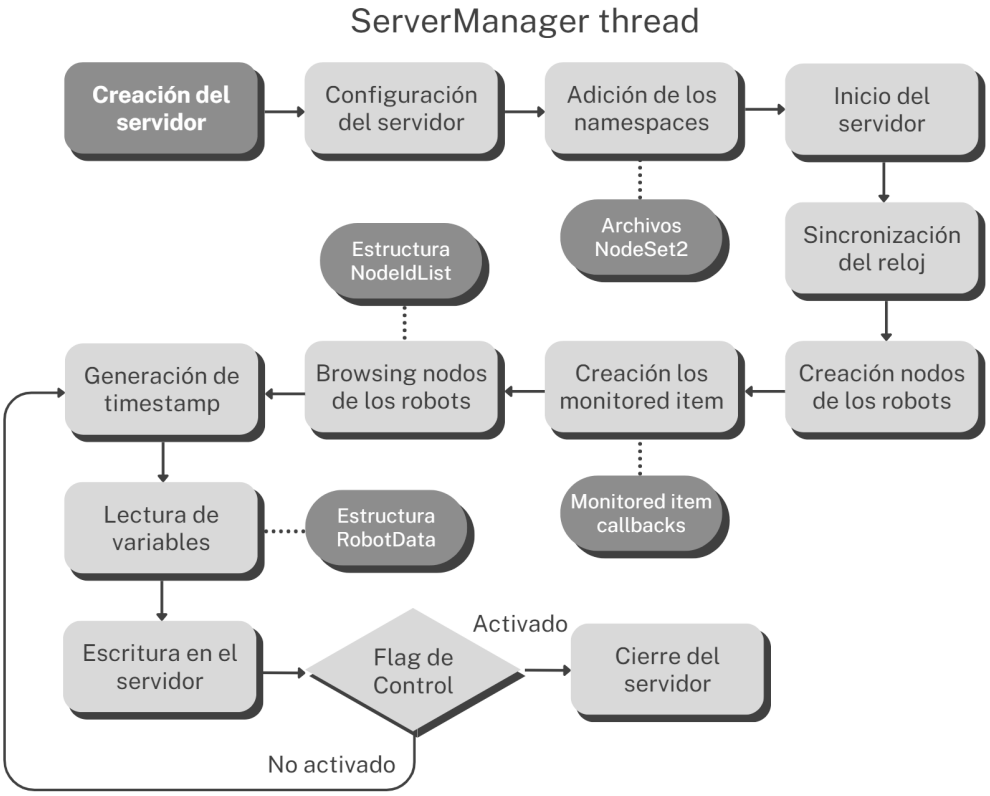


Figura 4.9: Diagrama de flujo del thread para la gestión del servidor con la clase ServerManager.

En este punto se crean dos objetos "Monitored Item" de OPC UA para monitorizar el valor de los nodos *EmergencyStop* y *ProtectiveStop* mediante los métodos *addMonitoredItemToEmergencyState* y *addMonitoredItemToStopState*. El primer método está asociado a la implementación de al parada de emergencia, mientras que el segundo implementa la puesta a cero del robot. Estos métodos crean los Monitored Item que actúan como disparadores (triggers) llamando a una callback que actúa sobre el flag booleano de la estructura RobotData del robot.

Antes de entrar en el bucle principal se sincroniza el tiempo del servidor OPC UA con la marca de tiempo del robot para garantizar una representación precisa del tiempo mediante *mutuex* y

variable conditions. A continuación el thread entra en el bucle principal para leer continuamente los datos del robot en las estructuras compartidas y actualizar los nodos OPC UA con la información más reciente. En este bucle se realizan llamadas al método *addVariable* que recibe los siguientes argumentos:

- Puntero al objeto del servidor.
- Vector con las ID de los nodos de la estructura "NodesIdsList" que contiene los nodos en los que deben escribirse los datos.
- Semáforo de la variable que se está escribiendo. Este semáforo es compartido con el thread RobotManager, de esta forma se garantiza que no haya condiciones de carrera al acceder a la variable en al estructura de datos compartida.
- Vector de la estructura RobotData conteniendo los últimos valores actualizados de los datos del robot.
- Marca temporal del momento en el que se recogieron los datos del robot. Con ella se calculará la marca temporal que escribir en los atributos SourceTimestamp de los nodos del servidor.

En el método de *addVariable*, para escribir los valores en los nodos del servidor se ha utilizado la función *UA_Server_writeDataValue* de la API de open62541. Esta función posibilita la escritura adicional de los atributos de *sourceTimestamp* y *serverTimestamp* de los nodos, en los cuales se escriben las marcas de tiempo del robot (obtenida en el thread RobotManager) y el servidor respectivamente.

4.4.4 Construcción y compilación del programa

En el proceso de desarrollo de software, la construcción y compilación de una aplicación son etapas críticas para obtener un programa funcional y listo para su ejecución. La construcción implica la organización y configuración de los componentes del proyecto, mientras que la compilación traduce el código fuente en un formato ejecutable para la plataforma objetivo. En este contexto, el archivo CMakeLists.txt juega un papel fundamental al establecer las directivas y comandos necesarios para llevar a cabo estas tareas de forma eficiente y estructurada.

En esta sección, se aborda en detalle el contenido y propósito del archivo CMakeLists.txt en el proyecto. Se describen las configuraciones y dependencias necesarias para la construcción del servidor OPC UA, que actúa como un gemelo digital del brazo robot en tiempo real. Mediante la utilización de targets y comandos CMake específicos, se generan los tipos y espacios de nombres esenciales para representar y exponer los datos relevantes del brazo robot. Además, se enfatiza cómo la modularidad y organización proporcionadas por CMake facilitan la adición o eliminación de características del proyecto sin afectar otras partes del código.

En el archivo CMakeLists.txt del servidor, se comienza especificando la versión mínima requerida de CMake, que en este caso es 3.2 (Swidzinski 2022). Esta versión es suficiente para aprovechar las características modernas de CMake, incluidas las facilidades para trabajar con targets. En la última década, CMake ha evolucionado y se ha vuelto más poderoso al introducir la noción de targets. Estos representan elementos individuales del software, como bibliotecas, ejecutables y módulos, y permiten configurar dependencias y opciones específicas para cada uno de ellos.

Ello ofrece un mayor control y flexibilidad para gestionar las diferentes partes del proyecto. En él, se utilizan targets para gestionar las diferentes partes del servidor OPC UA, lo que simplifica y organiza la construcción del proyecto.

En la Figura 4.10 se muestra un ejemplo claro de cómo se gestionan las dependencias y bibliotecas utilizando targets en el archivo CMakeLists.txt del servidor para la búsqueda de la biblioteca open62541. Con el comando *find_package*, CMake localiza y configura automáticamente la biblioteca open62541 en el sistema, lo que permite utilizarla en el proyecto como un target. Este se utiliza posteriormente para enlazar el servidor OPC UA con la biblioteca en las dependencias (*include*) del programa del servidor.

```
cmake_minimum_required(VERSION 3.2)
project(opcua - universal - robots - server)

# open62541 must be installed.
find_package(open62541 1 REQUIRED COMPONENTS FullNamespace PATHS "~/open62541/build")

if (NOT CMAKE_PROJECT_NAME STREQUAL PROJECT_NAME)
# needed or cmake doesn't recognize dependencies of generated files
set(PROJECT_BINARY_DIR ${ CMAKE_BINARY_DIR })
endif()

add_definitions(-DUA_NAMESPACE_ZERO = FULL)
set(UA_NAMESPACE_ZERO "FULL")

# Output directory for Nodeset Compiler @ . / build / src_generated /
set(GENERATE_OUTPUT_DIR "${CMAKE_BINARY_DIR}/src_generated/")

# Creates the directory
file(MAKE_DIRECTORY "${GENERATE_OUTPUT_DIR}")
include_directories("${GENERATE_OUTPUT_DIR}")

if (UA_NAMESPACE_ZERO STREQUAL "FULL")

# Generate types and namespace for DI
ua_generate_nodeset_and_datatypes(
    NAME "di"
    FILE_CSV "${PROJECT_SOURCE_DIR}/UA-Nodeset/DI/Opc.Ua.Di.NodeIds.csv"
    FILE_BSD "${PROJECT_SOURCE_DIR}/UA-Nodeset/DI/Opc.Ua.Di.Types.bsd"
    OUTPUT_DIR "${GENERATE_OUTPUT_DIR}"
    NAMESPACE_MAP "2:http://opcfoundation.org/UA/DI/"
    FILE_NS "${PROJECT_SOURCE_DIR}/UA-Nodeset/DI/Opc.Ua.Di.NodeSet2.xml"
    INTERNAL
```

Figura 4.10: Fragmento del archivo CMakeList.txt del servidor mostrando el uso de targets.

Otra parte esencial del proyecto es la generación de los tipos y namespaces para representar y exponer los datos del brazo robot de acuerdo a la estructura creada en el modelo de información. En esta tarea, se utilizan targets adicionales creados por el comando *ua_generate_nodeset_and_datatypes* disponible en la biblioteca open62541. Estos targets generan el código fuente necesario para los Companion DI (Device Integration) y RCS y el modelo de información propio "universal_robots". Luego son enlazados con el servidor para asegurar la funcionalidad adecuada del gemelo digital.

Además de gestionar las dependencias y bibliotecas, CMake también ofrece facilidades para la creación y ejecución de tests. En el archivo CMakeLists.txt del servidor, se encuentran configuraciones para generar distintos tests dependiendo del argumento proporcionado en la construcción. Por ejemplo, al utilizar el argumento *Debug_asan*, se generan tests utilizando el AddressSanitizer

ASan para detectar y prevenir errores de memoria durante la ejecución del programa. Asimismo, con el argumento *Debug_ def*, se prepara la construcción de tests unitarios en modo de depuración. Estos tests se utilizan para asegurar la calidad y correctitud del código fuente, garantizando que el servidor OPC UA funcione adecuadamente y mantenga un rendimiento óptimo. El uso de tests en CMake permite una validación más rigurosa del servidor y asegura que el gemelo digital del brazo robot esté listo para su implementación en entornos de producción industriales.

4.5 Implementación del cliente OPC UA

Tal como se presentó en la Figura 1.1, el cliente OPC UA se ha implementado directamente en el programa del plugin desarrollado para CoppeliaSim. Esta decisión se basa en las ventajas de integración entre el cliente OPC UA y la simulación en CoppeliaSim, lo cual se explora en detalle en la sección 5.3. Para implementar el cliente ha utilizado la biblioteca open62541, que dispone de varias librerías ofreciendo una amplia gama de funcionalidades, desde bajo nivel para la gestión del cliente, hasta funciones de alto nivel para establecer la comunicación con el servidor. Además, al misma librería que se empleó para implementar el servidor, se garantiza una coherencia y uniformidad en el manejo de la comunicación cliente-servidor a lo largo del proyecto.

La conexión entre un cliente OPC UA y un servidor OPC UA constituye un paso fundamental en el establecimiento de una comunicación efectiva. La Figura 4.11 ilustra el ciclo de vida del cliente. Para establecer la conexión, una alternativa directa y sencilla es emplear la URI del servidor. Esta URI, que engloba tanto la dirección IP del servidor como el puerto dedicado para la comunicación y permite al cliente ubicar y establecer un enlace directo con el servidor OPC UA. Puesto que en este proyecto se ha creado el servidor desde el bajo nivel, es posible modificar la URI de forma sencilla. No obstante, se debe destacar que, en el caso más común de no contar con esta URI, existe la opción de recurrir al servicio denominado *Discovery*. Dicho servicio desempeña el papel de localizar los servidores disponibles en la red junto con sus endpoints, facilitando así la identificación y elección del servidor adecuado para la comunicación.

El proceso de *Discovery* se basa en la utilización del protocolo mDNS (Multicast DNS) en combinación con la tecnología de grupos de multidifusión. Cuando el cliente emite una solicitud de descubrimiento en la red, esta solicitud se propaga a través de la red en busca de servidores que respondan a la petición. Los servidores interesados en ser detectados responden a la solicitud, lo que permite que el cliente recoja información detallada sobre los servidores disponibles y sus respectivos *endpoints*. El protocolo mDNS opera en el nivel de capa de enlace y utiliza direcciones de especiales para anunciar y descubrir dispositivos y servicios en una red local. Esta capacidad de auto-descubrimiento elimina la necesidad de una configuración manual de la tarea de descubrimiento. Una vez que el cliente recopila la información de los servidores disponibles a través del proceso de *Discovery*, puede tomar decisiones informadas sobre con cuál servidor establecerá una conexión.

Tras haberse realizado el intento de conexión, se desencadena una etapa crítica de verificación. Se realiza una evaluación de las variables tipo *UA_ STATUSCODE* proporcionadas por las funciones de la biblioteca open62541. Estas variables, que están intrínsecamente vinculadas a los atributos de estado de los nodos y servicios tanto del cliente como del servidor, permiten una detección

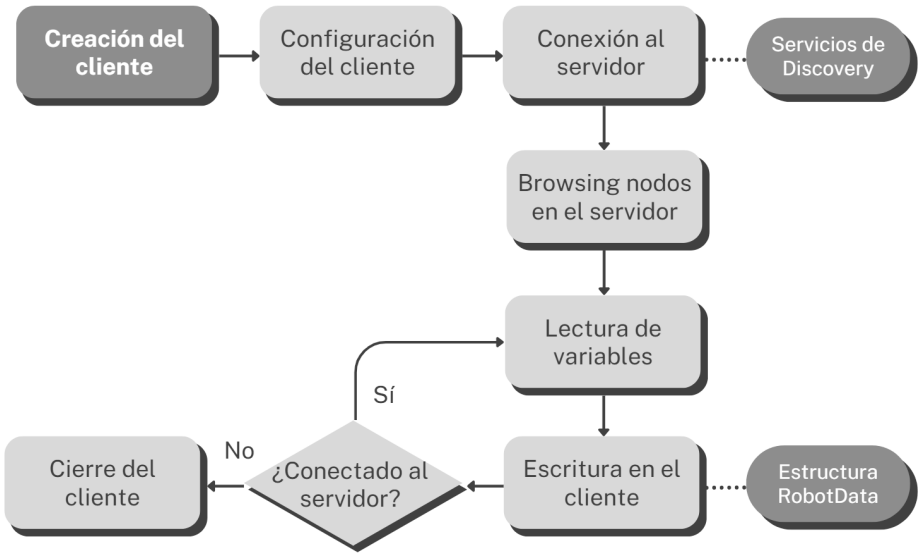


Figura 4.11: Diagrama del funcionamiento del cliente OPC UA.

temprana de posibles problemas de comunicación. Al dar prioridad a la detección de errores, se garantiza un proceso de conexión robusto y confiable. Con la conexión establecida y confirmada, el cliente queda preparado para iniciar interacciones con el servidor OPC UA.

Una de las partes cruciales de la implementación es la búsqueda de nodos en el servidor. Para ello se ha creado la función browseForNodesRecursive, la cual se encarga de explorar la estructura del servidor en busca de nodos que coincidan con el nombre de búsqueda objetivo. La función se inicia con la configuración de los parámetros necesarios, como la definición de la solicitud de navegación ($UA\_BrowseRequest$). Esta solicitud establece los detalles de la búsqueda, como el nodo padre desde el cual comenzar la exploración y el nombre de búsqueda objetivo. Utilizando la función $UA\_Client\_Service\_browse$, se realiza la solicitud de navegación al servidor. Esta función proporciona una vista general de los nodos disponibles en el servidor, incluyendo sus referencias y atributos.

A continuación se itera a través de los resultados de la solicitud de navegación para analizar cada referencia individualmente. Esto se realiza dentro de bucles anidados para recorrer los resultados y las referencias asociadas. En cada iteración, la función compara el nombre de búsqueda objetivo con el nombre de referencia del nodo actual. Si se encuentra una coincidencia, se identifica que el nodo cumple con los criterios de búsqueda. Para asegurarse de que se revisen todas las posibles ramificaciones en la estructura del servidor, la función utiliza una llamada recursiva. Si el nodo actual tiene nodos hijo, se realiza una llamada a browseForNodesRecursive con el nodo actual como punto de partida. Si se encuentra un nodo que coincide con el nombre de búsqueda, se almacena en una estructura de datos local, de forma que el cliente tenga acceso rápido a los nodos relevantes cuando sea necesario. Durante su ejecución, la función considera posibles escenarios en los que no se encuentren coincidencias o en los que ocurran problemas de comunicación con el servidor. Esto se aborda mediante la evaluación de los UA_STATUSCODE y la toma de decisiones adecuadas en consecuencia.

La implementación de la recuperación de datos del servidor OPC UA, se centra en la obtención de datos de posición de las articulaciones del brazo robot, para la simulación del movimiento del brazo. Esta operación se realiza mediante un bucle de recuperación de datos que se ejecuta de manera continua mientras el cliente esté activo y la conexión con el servidor sea estable, una vez se han recuperado los nodos de posición. Entonces, se utiliza la función de open62541 *UA_Client_readValueAttribute* para leer los valores de las variables de los nodos del servidor y se almacenan en una variable tipo *Variant*. De esta forma se capturan también los datos del TCP para su posterior análisis.

Por último, la función EmergencyStop implementa la parte de la comunicación bidireccional desde el cliente al servidor. La ejecución de esta función se muestra en el diagrama de la Figura 4.12. Antes de interactuar con el servidor, la función crea y configura una instancia de cliente OPC UA utilizando la biblioteca open62541 y establece la conexión con el servidor utilizando la dirección y el puerto apropiados. A continuación, se realiza una búsqueda de nodos utilizando la función browseForNodesRecursive que, en este caso, busca el nodo específico que representa el botón de parada de emergencia del robot seleccionado. Puesto que todos los robots han sido creados como instancias del *ObjectType* RobotArm, el nodo del estado de emergencia (al igual que el resto de nodos) tiene el mismo nombre en todos los robots. Por ello, sólo es necesario conocer la id del nodo raíz de la instancia de robot que se desea detener.

Una vez encontrado el nodo correspondiente al estado de parada de emergencia, se configura una estructura de petición de escritura que contiene la información para realizar la escritura en el servidor como el nodo de destino, el atributo en el que se realizará la escritura (como el valor), el tipo de dato y el valor propiamente dicho. A continuación, se utiliza la función de alto nivel de la biblioteca open62541: *UA_Client_Service_write*, para enviar la petición de escritura al servidor. El cliente utiliza entonces el protocolo de comunicación especificado por OPC UA para establecer una conexión con el servidor, a través de la cual, el cliente envía la estructura de petición empaquetada al servidor. Una vez que el servidor recibe la petición de escritura, comienza a procesarla. Primero, verifica la validez de la petición, incluida la autenticidad del cliente y la autorización para realizar la escritura en el nodo de destino. Si el servidor aprueba la petición, procede a actualizar el valor en el nodo de acuerdo con los datos proporcionados por el cliente.

Después de completar la escritura, el servidor genera una respuesta que contiene información sobre el resultado de la operación. Esta respuesta se empaqueta en el formato establecido en la configuración del cliente/servidor y se envía de vuelta al cliente a través de la misma conexión. La respuesta informa al cliente si la escritura se realizó con éxito o si hubo algún problema durante el proceso. De la misma forma se puede implementar una función que lleve el robot a la posición de origen o cualquier otra función de control y monitoreo que involucren la comunicación desde el cliente hasta el servidor.

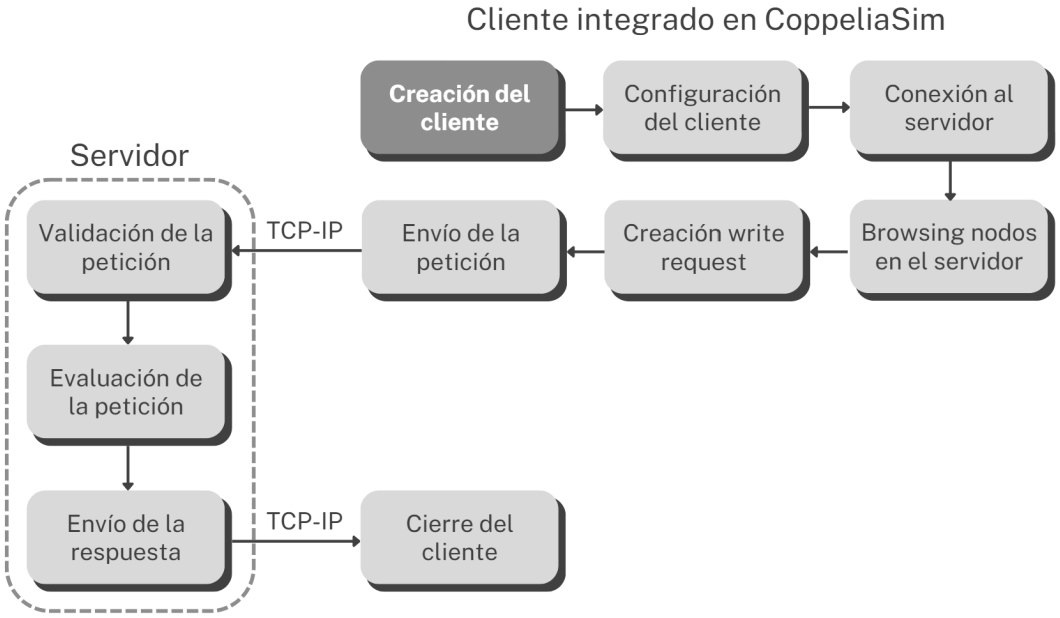


Figura 4.12: Diagrama de la ejecución de la función EmergencyStop del cliente OPC UA.

Capítulo 5

Simulación en CoppeliaSim

En este capítulo, se presenta la simulación del gemelo digital del brazo colaborativo Universal Robots en la plataforma CoppeliaSim. En el capítulo, se comienza examinando las características físicas, cinemáticas y geométricas del modelo del brazo colaborativo, presentando una descripción que sienta las bases de la simulación. Posteriormente, se presentan los parámetros específicos de la escena en CoppeliaSim, esenciales para crear un entorno coherente y realista para el gemelo digital.

Dentro de este contexto, se profundiza en la estructura del plugin desarrollado para CoppeliaSim. Se explican las herramientas utilizadas en el plugin, simPlusPlus y simStubsGen, el diseño y la funcionalidad del plugin, junto con diagramas de la estructura del código. Esta sección busca proporcionar una comprensión clara y directa de cómo el plugin facilita la comunicación y la interacción entre el cliente OPC UA y la simulación.

Finalmente, se examina en detalle la integración y comunicación entre el cliente OPC UA y CoppeliaSim. Se aborda el flujo de datos entre estos dos componentes, resaltando cómo los datos del cliente se incorporan a la simulación.

5.1 Configuración del Modelo y la Escena

En esta sección, se aborda la configuración detallada del modelo del brazo colaborativo Universal Robots en la plataforma de simulación CoppeliaSim, así como la definición de los parámetros esenciales de la escena. La precisión y coherencia en la representación virtual del gemelo digital dependen en gran medida de cómo se establecen y ajustan estas propiedades y parámetros. En ese sentido, esta sección proporciona un análisis en profundidad de los aspectos físicos y geométricos de los modelos, así como de los componentes clave que conforman el entorno de la simulación.

A medida que se exploran las propiedades de los modelos y los parámetros de la escena, se busca proporcionar una comprensión sólida de cómo se ha configurado el gemelo digital para emular de manera efectiva el comportamiento del brazo colaborativo real. Se inicia describiendo las características inherentes al modelo del brazo. Posteriormente, se presentan los parámetros de la

escena en CoppeliaSim, los cuales son fundamentales para establecer las condiciones iniciales y la apariencia visual de la simulación.

5.1.1 Propiedades del Modelo del Brazo Colaborativo

Los modelos de brazos colaborativos utilizados en las simulaciones corresponden al UR5 y el UR3. Estos modelos han sido proporcionados por Universal Robots y están disponibles tanto en su página web como en el repositorio oficial de CoppeliaSim en GitHub. Se trata de réplicas detalladas de los modelos de los cobots reales, adaptados para su simulación en el entorno virtual de CoppeliaSim. A continuación, siguiendo los conceptos de modelado propuestos por Rohmer, Singh y Freese (2013), se exponen las propiedades clave de estos modelos de brazos colaborativos en CoppeliaSim:

- Estructura física: En el contexto de CoppeliaSim, se dispone de diversas opciones para la creación y modelado de las formas geométricas que constituyen los segmentos de los brazos colaborativos. Desde la posibilidad de generar formas básicas en la escena, cuyas propiedades dinámicas son optimizadas por defecto, hasta la importación de geometría CAD desde aplicaciones externas como SolidWorks. Para lograr un equilibrio entre precisión y eficiencia, se recomienda trabajar con archivos CAD que mantengan un nivel de detalle adecuado, evitando el exceso de información no relevante. Una práctica común es la simplificación de las mallas mediante herramientas como "Extract the convex hull" o "Decimate the mesh". Una vez importadas todas las partes de la malla, el modelo se descompone en los segmentos necesarios para su movimiento y simulación obteniendo un resultado similar al mostrado en la Figura 5.1.

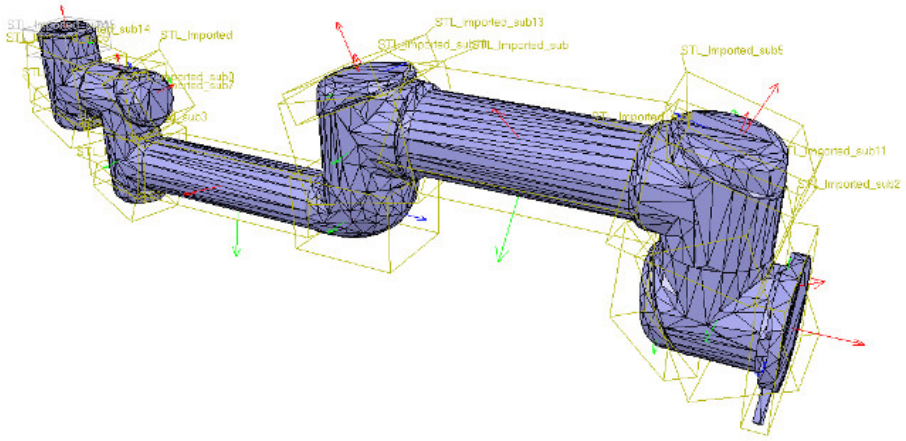


Figura 5.1: Modelo de brazo robot UR5 mallado y dividido en secciones en CoppeliaSim. Fuente: Coppelia Robotics.

- Apariencia: Se pueden ajustar una gran cantidad de propiedades de objetos (creados in-situ o importados) en CoppeliaSim desde el menú "Propiedades de la escena". Desde este cuadro de diálogo se pueden modificar de las propiedades "Comunes" y las propiedades de "Forma". Mientras que las primeras controlan características generales de la interacción del objeto con la simulación y otros objetos, las propiedades de la pestaña "Forma", permiten cambiar el color, la textura y las propiedades dinámicas del objeto, permitiendo una personalización completa de su aspecto en el entorno virtual.

- Cinemática: La movilidad de las articulaciones del modelo se logra a través de la implementación de *joints*. CoppeliaSim ofrece varias herramientas para añadir estos elementos a los modelos. Si se dispone de información sobre la posición y orientación de los ejes, se puede usar la herramienta *Denavit-Hartenberg joint creator.ttm* o los cuadros de posición y orientación para colocar los ejes. En ... otros casos, al disponer únicamente de la geometría CAD, es necesario crear objetos nativos sobre esta geometría que sirvan como referencia para colocar los *joints*. Esta flexibilidad permite replicar los movimientos articulados del brazo colaborativo UR5, como se puede observar en el modelo de la Figura 5.2.

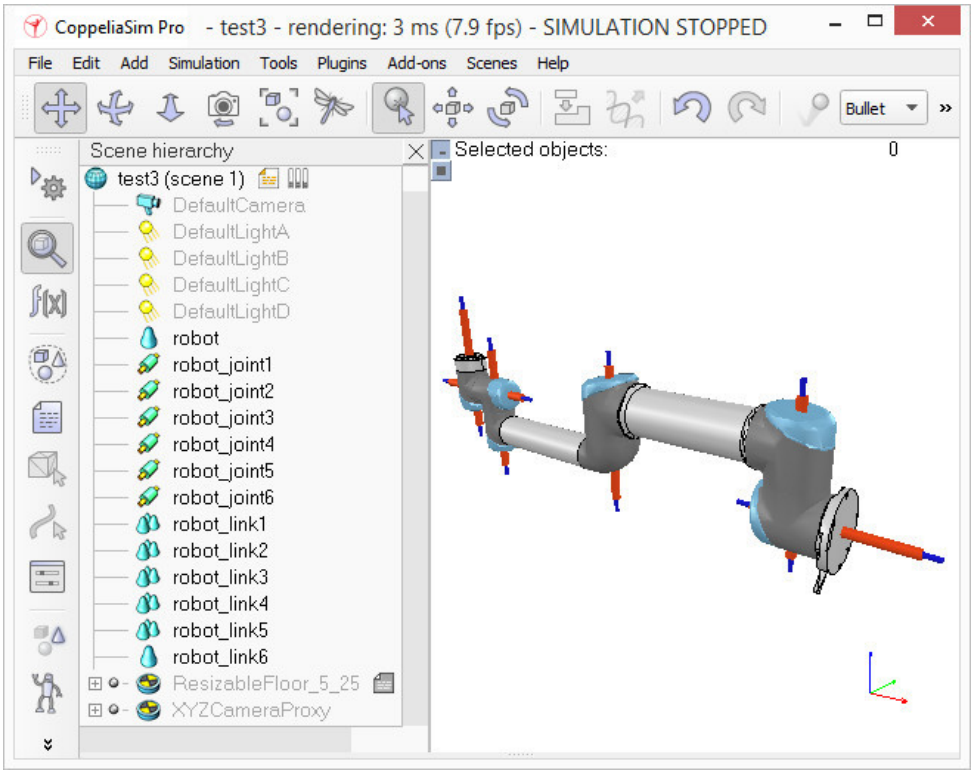


Figura 5.2: Segmentos y ejes del modelo de brazo robot UR5 en CoppeliaSim. Fuente: Coppelia Robotics.

- Modelado dinámico: Aunque el modelado dinámico es un tema complejo, en CoppeliaSim es posible obtener un modelo de forma sencilla que pueda experimentar fuerzas y aceleraciones (objeto *dynamic*) y detectar colisiones (objeto *respondable*). Para lograr esto, normalmente se lleva a cabo la simplificación o parametrización de los segmentos existentes. Una forma técnica rápida para simplificar mallas es el modo "Edición de triángulos" que permite definir formas nativas en función de una malla existente. Otros objetos más complejos pueden ser modelados mediante la descomposición en formas cóncavas o convexas. Una vez obtenidos las nuevas formas dinámicas, es posible acceder a sus características dinámicas desde el cuadro de diálogo de "Propiedades Comunes".
- Definición del modelo: En CoppeliaSim, la estructura jerárquica de los modelos y escenas se presenta en forma de árbol. Esta organización se refleja en la relación entre los objetos y sus niveles de jerarquía, como se ilustra en la Figura 5.3. Para un funcionamiento correcto, los objetos dinámicos deben ser padres de los segmentos de geometría y de los *joints*, estableciendo relaciones jerárquicas que aseguren la coherencia del modelo.

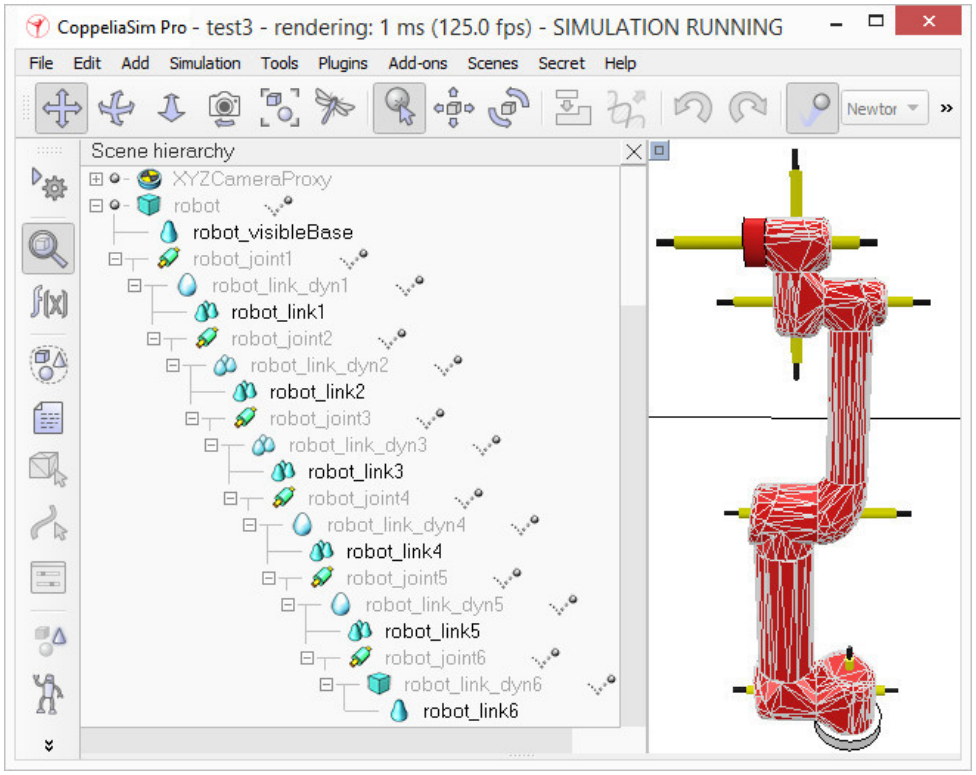


Figura 5.3: Jerarquía completa del modelo de brazo robot UR5 en CoppeliaSim. Fuente: Coppelia Robotics.

- TCP (Tool Center Point): La creación de un TCP en CoppeliaSim, destinado a la conexión de herramientas, puede ser lograda de diversas maneras. La opción más apropiada en el caso de brazos colaborativos es emplear sensores de fuerza, que en CoppeliaSim actúan como conexiones rígidas. Esta elección permite una interacción fluida con las herramientas, facilitando su colocación y cambio en la simulación, al igual que ocurre en la realidad. Además ello permite que el modelo en CoppeliaSim refleje con precisión las capacidades del brazo colaborativo físico.

Es importante destacar que en este trabajo no ha entrado en detalle sobre aspectos específicos de la parametrización y manipulación de objetos en CoppeliaSim. Para obtener información detallada sobre el uso de este software, se recomienda consultar el manual de usuario proporcionado por Coppelia Robotics.[1]

5.1.2 Parámetros de la escena

Las escenas en CoppeliaSim son los elementos básicos para la simulación. Las escenas son realmente archivos (extensión .ttt)que pueden ser guardados o cargados en CoppeliaSim. Estas pueden contener modelos, objetos, entornos, un "main script", páginas y vistas. La mayoría de elementos han sido descritos en la subsección 2.4.2. En este caso se utiliza una escena vacía de la biblioteca de escenas de CoppeliaSim como punto de partida.

El propósito de desarrollar al implementación de la aplicación del gemelo digital para robots UR en un plugin de CoppeliaSim es simplificar y automatizar lo máximo posible la aplicación. Sin embargo, algunos elementos necesarios en la aplicación no pueden ser creados desde el código del

[1] `https://www.coppeliarobotics.com/helpFiles/index.html`

plugin o su creación desde el código no es recomendable. Visto esto, ha sido necesario configurar escenas de forma manual antes de ejecutar el plugin del gemelo digital. Los elementos añadidos a la escena base de CoppeliaSim son los siguientes:

- Gráficos: En las versiones más reciente de CoppeliaSim, los objetos que contienen los gráficos (*Graphs*), sólo pueden ser creados desde la barra del menú de la escena, aunque una vez creados pueden ser accedidos desde la API.
- Interfaces gráficas de usuario (GUI): CoppeliaSim posibilita la creación de GUIs mediante el plugin UI desarrollado por Federico Ferri.[2]. Este plugin se basa en el framework multiplataforma orientado a objetos Qt. La creación de las interfaces gráficas pasa por la programación de su estrcutura en XML, y el uso de la sintaxis específica en LUA. Estos conceptos se desarrollan en la siguiente sección.

5.1.3 Interfaz gráfica de usuario personalizada

En el entorno de simulación de CoppeliaSim, la interacción intuitiva y la presentación de información son elementos cruciales para el desarrollo y la visualización de escenarios complejos. Aunque CoppeliaSim ofrece diversas formas de visualizar y registrar datos, entre las cuales se incluyen la visualización de datos en gráficos y la incorporación de elementos visuales en la escena, una de las alternativas más destacadas es la creación de interfaces personalizadas mediante el plugin simUI. Esta herramienta potente y versátil permite diseñar y desplegar elementos de interfaz adaptados a sus necesidades específicas, facilitando la interacción y la presentación de información relevante de manera eficiente.

El plugin simUI, desarrollado por Federico Ferri, encapsula la funcionalidad del framework Qt y exporta una serie de funciones API que permiten la creación y la interacción con GUI en CoppeliaSim. A través de estas interfaces personalizadas, los usuarios pueden diseñar y desplegar elementos de la interfaz, como cuadros de diálogo con botones, cuadros de edición, deslizadores, etiquetas, imágenes, gráficos y mucho más .

Una de las características mas importantes del plugin simUI es su capacidad para definir y manipular estos elementos de la interfaz utilizando una sintaxis XML como muestra la Figura 5.4. Esta sintaxis permite describir la estructura y el diseño de la interfaz de manera detallada y fácilmente comprensible. Esta estructura puede crearse e importarse desde un archivo .xml o directamente en un script personalizado de CoppeliaSim. Cada acción realizada en la interfaz personalizada, ya sea hacer clic en un botón, editar un campo de texto o mover un deslizador, se comunica al entorno de simulación a través de callbacks desde un script personalizado, lo que permite una interacción dinámica y la respuesta en tiempo real a las acciones del usuario.

El proceso de creación de una interfaz personalizada a través del plugin simUI sigue los siguientes pasos:

Definición de la Interfaz: Se ha de definir la interfaz personalizada mediante un código XML que detalla la estructura y el contenido de la interfaz. Esta código define los elementos de la UI, sus propiedades, su disposición y su funcionalidad asociada. Un aspecto clave en esta parte de la programación es la definición de las id de los elementos de la interfaz. Estos identificadores

[2] Enlace al repositorio de GitHub oficial del plugin UI

```
if (sim_call_type==sim.syscb_init) then
    ui=simUI.create('[
    <ui title = "RobotUR0 Dashboard" size="200,220" closeable="true">
        <tabs>
        <tab title="Control">
            <button id="20" text="Control Robot UR0" checkable="true" on-click="controlUR0" />
            <label id="10" text="Running external script." />
            <group>
                <label text="Robot options:" wordwrap="true" />
                <button id="21" text="Emergency Stop" enabled="false" on-click="emergencyCallback" style="background: #ff0000;" />
                <button id="22" text="Position in cero" enabled="false" checkable="true" on-click="poszeroCallback"/>
            </group>
        </tab>
        </tabs>
    </ui>]])
```

Figura 5.4: Fragmento del código xml que implementa la estructura de la interfaz de usuario personalizada en CoppeliaSim.

únicos definidos al crear los elementos permiten reverenciarlos y acceder a sus propiedades desde otros lugares del código.

Integración de Elementos: Los elementos de la interfaz, como botones, cuadros de edición y gráficos, se integran en el código XML utilizando etiquetas específicas. Los elementos disponibles para usar en al interfaz están limitados a los elementos básicos del framework Qt. En la Figura 5.5 se muestra la creación de textos y botones con distintas propiedades.

```
function controlUR0(ui,id)
    if (controlBool == false)
    then
        simUI.setButtonText(ui, 20, "Controlling Robot UR0")
        simUI.setLabelText(ui, 10, "User RTDE Control", false)
        simUI.setEnabled(ui, 21, true, false)
        simUI.setEnabled(ui, 22, true, false)
        controlBool = true
    else
        simUI.setButtonText(ui, 20, "Control Robot UR0")
        simUI.setLabelText(ui, 10, "Control released", false)
        simUI.setEnabled(ui, 21, false, false)
        simUI.setEnabled(ui, 22, false, false)
        controlBool = false
    end
end

function emergencyCallback(ui,id)
    local result
    result=simUI.msgBox(2, 1, "Emergency Stop Pressed", "Do you want to stop this robot?")
    if (result == 2)
    then
        simUI.setButtonText(ui, 21, "Emergency Stop Pressed")
        simUI.setEnabled(ui, 21, false, false)
        simOpcuaClient.emergencyStop0()
    end
end
```

Figura 5.5: Fragmento del código que implementa distintos callback en la interfaz de usuario personalizada en CoppeliaSim.

Configuración de Acciones: La definición de las acciones que deben realizarse cuando el usuario interactúa con los elementos de la interfaz, por ejemplo, al hacer clic en un botón, se puede especificar mediante una función en el mismo script personalizado de CoppeliaSim. Es posible también especificarla desde una fuente externa como un script de LUA externo compilado junto al plugin, aunque esta alternativa presenta una mayor complicación.

Registro y Visualización: La interfaz debe ir asociada a alguno de los elementos de la escena actual de CoppeliaSim. Además, estos elementos han de poder recibir un script personalizado de CoppeliaSim en caso de programar la interfaz in-situ.

Callbacks: Cada acción del usuario en la interfaz personalizada genera un callback desde a un script correspondiente, que puede programarse para realizar diversas acciones en la simulación mediante funciones de la API de CoppeliaSim. Los callbacks pueden también llamar a funciones en archivos externos u otros plugin como es el caso del ejemplo de la Figura 5.5.

La interfaz desarrollada en el proyecto tiene el propósito de permitir al usuario actuar sobre el estado del robot. Puesto que la actuación sobre el robot no es el objetivo principal de este proyecto, se ha realizado una interfaz sencilla que únicamente implementa la parada del robot. El propósito de esta interfaz es disponer de una platilla que demuestre el uso del plugin para creación de GUI personalizadas en CoppeliaSim y la conexión bidireccional con el robot. La Figura 5.6 muestra el diagrama de navegación de la interfaz.

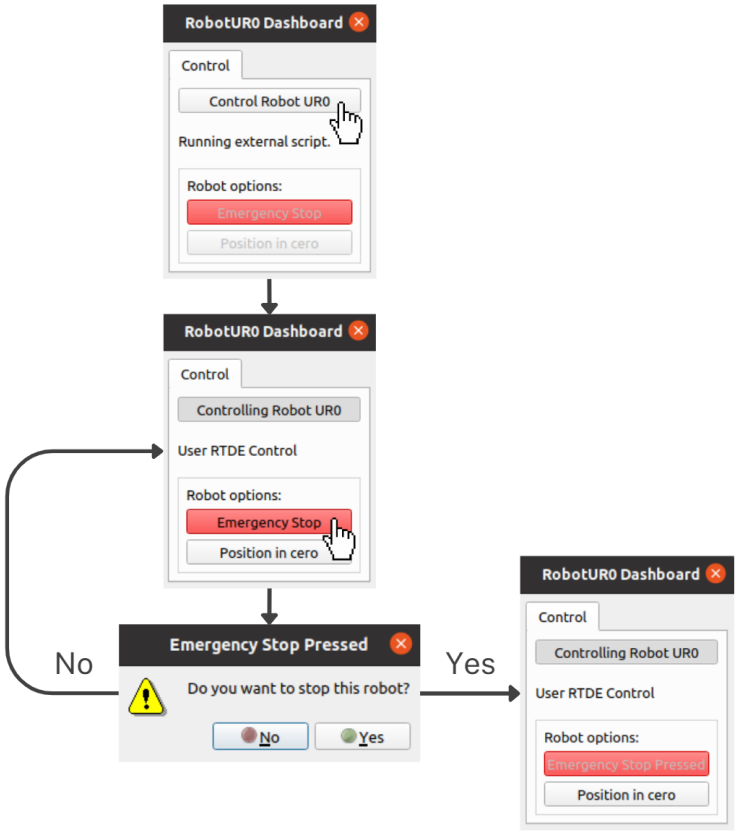


Figura 5.6: Diagrama de la navegación en la GUI personalizada creada en CoppeliaSim.

La interfaz es lanzada desde los child scripts asociados a cada robot al ejecutar la simulación. Cada instancia de robot que se haya creado en la simulación, creará una interfaz individual para su actuación. Entonces, una vez lanzada, se encuentra disponible el botón "Control del robot UR0", haciendo clic sobre este botón se habilitan las acciones de actuación sobre dicha instancia de robot, en este caso el denominado UR0. A continuación, el usuario puede interaccionar con el robot mediante los botones de "Emergency Stop" o "Position in cero" que, tal como su nombre indica, implementan una parada de emergencia y el retorno a la posición inicial del robot respectivamente. En el caso de la parada de emergencia, al pausar el botón se muestra al usuario una ventana emergente adicional preguntando si realmente desea parar el robot.

Tal como se observa en código LUA de la Figura 5.5, en el momento que el usuario acepta la ventana emergente preguntando si realmente desea parar el robot, se modifica la interfaz al estado 3 de la Figura 5.6, y se realiza la callback a la función "emergencyStop0" perteneciente al plugin. En la Figura 4.12 se puede observar el diagrama de flujo de la ejecución de esta callback.

El plugin simUI amplía las capacidades de interacción y personalización en CoppeliaSim, permitiendo a los usuarios diseñar interfaces de usuario adaptadas a sus necesidades específicas. Desde el control y la visualización de datos hasta la implementación de interacciones complejas, simUI brinda a los desarrolladores y usuarios avanzados una herramienta esencial para crear experiencias de simulación más enriquecedoras y eficientes. Con la posibilidad de integrar elementos Qt-style en la interfaz, el plugin simUI abre un mundo de posibilidades para la creación de entornos de simulación altamente personalizados y orientados a objetivos. El repositorio del plugin simUI se encuentra disponible dentro del repositorio oficial de CoppeliaSim[3].

5.2 Entorno de programación de plugins en CoppeliaSim

En esta sección se detalla la implementación del plugin que posibilita la comunicación con el servidor, descrita en la sección 4.5, y la integración en CoppeliaSim.

En los últimos años, CoppeliaSim ha introducido una nueva metodología para la creación de plugins con el objetivo de hacer el proceso más eficiente y accesible para los desarrolladores. Anteriormente, la creación de plugins en CoppeliaSim involucraba la implementación de código específico para las funciones requeridas, como la inicialización del plugin y la gestión de eventos. Sin embargo, este enfoque requería un mayor esfuerzo por parte del desarrollador para establecer la infraestructura del plugin y manejar detalles técnicos, lo que podía ser complicado y consumir tiempo.

La nueva metodología de desarrollo se basa en una arquitectura más modular y orientada a objetos, lo que facilita la creación y el mantenimiento de plugins. El *simSkel repository* (2023) proporciona una estructura y un conjunto de clases predefinidas que cubren aspectos comunes del desarrollo de plugins, como la inicialización, la gestión de eventos y la comunicación con la simulación. Esta nueva metodología ofrece varias ventajas significativas para los desarrolladores que desean crear plugins en CoppeliaSim, entre las que se incluyen:

- Simplificación del Desarrollo: La estructura modular y las clases predefinidas reducen la complejidad y la cantidad de código necesario para crear un nuevo plugin. Esto acelera el proceso de desarrollo y reduce la probabilidad de errores.
- Mayor Abstracción: La nueva metodología oculta gran parte de la complejidad técnica y ofrece una interfaz más abstraída para el desarrollador. Esto permite centrarse en la implementación de la funcionalidad específica del plugin sin preocuparse por detalles internos.
- Mejor Mantenimiento: La arquitectura orientada a objetos y la modularidad facilitan el mantenimiento y la actualización de los plugins en el futuro. Los cambios y mejoras pueden implementarse de manera más eficiente sin afectar el funcionamiento general del plugin.

[3] https://github.com/CoppeliaRobotics/simUI

- Documentación y Ejemplos Mejorados: La metodología viene acompañada de documentación y ejemplos detallados que guían a los desarrolladores a lo largo del proceso de creación del plugin. Esto reduce la curva de aprendizaje y aumenta la accesibilidad.

La nueva metodología moderna de desarrollo de plugins en CoppeliaSim ha ganado tracción en la comunidad de desarrolladores debido a su enfoque modular y su capacidad para abstraer gran parte de la complejidad técnica. Lo cual ha propiciado la aparición de nuevas herramientas basadas en esta metodología A continuación, se presentan dos herramientas clave que se han utilizado para la implementación del plugin del gemelo digital bajo esta nueva metodología: simStubsGen y simPlusPlus. Sin embargo, primero es necesario entender cómo se estructuran los plugin desarrollados mediante esta nueva metodología.

5.2.1 Estructura global del plugin

El plugin del gemelo digital en CoppeliaSim está compuesto por varios archivos que trabajan juntos para establecer la comunicación con un servidor OPC UA y sincronizar los datos con la simulación. Cada archivo desempeña un papel fundamental en el funcionamiento global del plugin:

- CMakeLists.txt: Este archivo mostrado en la Figura 5.7 es esencial para el proceso de compilación. Utilizando el sistema de construcción CMake, configura las opciones de compilación, los directorios de inclusión y las bibliotecas necesarias para vincular el plugin con CoppeliaSim y las bibliotecas de OPC UA. CMake genera archivos de construcción específicos para compilar el plugin en la plataforma y entorno de desarrollo seleccionados.

```
if(NOT COPPELIASIM_INCLUDE_DIR)
    if(DEFINED ENV{COPPELIASIM_ROOT_DIR})
        set(COPPELIASIM_INCLUDE_DIR $ENV{COPPELIASIM_ROOT_DIR}/programming/include)
    else()
        set(COPPELIASIM_INCLUDE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/../include)
    endif()
endif()
list(APPEND CMAKE_MODULE_PATH
    ${CMAKE_CURRENT_SOURCE_DIR}/cmake/modules
    ${COPPELIASIM_INCLUDE_DIR}/cmake)

find_package(CoppeliaSim 4.5.0.0 REQUIRED)

include_directories(${CMAKE_CURRENT_BINARY_DIR}/generated)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/external)
include_directories(${COPPELIASIM_INCLUDE_DIR})

coppeliasim_generate_stubs(
    ${CMAKE_CURRENT_BINARY_DIR}/generated
    XML_FILE ${CMAKE_CURRENT_SOURCE_DIR}/callbacks.xml
    LUA_FILE ${CMAKE_CURRENT_SOURCE_DIR}/simOpcuaClient.lua
)

configure_file(${CMAKE_CURRENT_SOURCE_DIR}/config.h.in ${CMAKE_CURRENT_BINARY_DIR}/config.h ESCAPE_QUOTES)

coppeliasim_add_plugin(
    simExtOpcuaClient
    SOURCES
    plugin.cpp
)

target_link_libraries(simExtOpcuaClient /home/manuel/open62541/build/bin/libopen62541.a)
```

Figura 5.7: Fragmento del archivo CMakeLists.txt para la compilación del plugin.

- callbacks.xml: En este archivo se definen los callbacks personalizados que el plugin utiliza para interactuar con CoppeliaSim. Los callbacks son funciones que se activan en momentos específicos, como la inicialización de la simulación o la carga de una escena. Este archivo establece la conexión entre las funciones definidas en el archivo principal del plugin y dichos eventos en la simulación.
- config.h.in: Contiene las macros y definiciones que afectan la compilación del plugin. CMake utiliza este archivo para generar un archivo de configuración específico para el entorno de desarrollo, permitiendo la adaptación de ciertos aspectos del plugin durante el proceso de compilación. La edición de este archivo es opcional y no es utilizado por muchos plugin.
- plugin.cpp: Este es principal del plugin. Aquí se implementa la clase Plugin, que se deriva de Plugin de la biblioteca simPlusPlus. Dentro de esta clase, se definen las funciones clave onStart y onSimulationAboutToStart, que se activan durante la inicialización y antes de que comience la simulación, respectivamente. Estas funciones coordinan la comunicación entre el servidor OPC UA y la simulación de CoppeliaSim.
- simOpcuaClient.lua: Este archivo LUA contiene un script que se ejecuta dentro de CoppeliaSim y actúa como intermediario entre el plugin y el cliente OPC UA. El script define funciones que interactúan con el cliente OPC UA para establecer la conexión con el servidor y enviar/recibir datos. El archivo simOpcuaClient.lua enlaza las funciones definidas en plugin.cpp con la simulación en sí.

5.2.2 Generación automatizada de stubs con simStubsGen

Una parte fundamental del proceso de desarrollo de plugins es la interacción con las funciones de la API de CoppeliaSim. Sin embargo, lidiar con la API puede resultar complejo y propenso a errores. Para abordar este desafío, se ha introducido simStubsGen, una herramienta que automatiza la generación de "stubs", es decir, interfaces simplificadas que facilitan el acceso a las funciones de la API y que pueden añadir nuevas funcionalidades.

El plugin simStubsGen se ejecuta por separado y examina los archivos .h de la API de CoppeliaSim, analizando las funciones, argumentos y estructuras definidas en ellos. A partir de esta información, simStubsGen genera automáticamente archivos .h y .cpp que definen las funciones y clases de la API en una forma más accesible. Estos archivos generados actúan como interfaces entre el código del plugin y la API de CoppeliaSim. Contienen declaraciones de funciones con argumentos más simples y nombres más claros, eliminando detalles innecesarios y complejidad técnica. Además, simStubsGen genera funciones de inicialización y manejo de eventos, simplificando aún más el proceso de creación de plugins.

Otra característica notable de simStubsGen es su capacidad para generar documentación HTML detallada sobre la API del plugin en desarrollo. Esta documentación se basa en los comentarios y etiquetas XML de los callbacks definidos en el código del plugin. simStubsGen procesa estos comentarios y extrae información relevante para generar una documentación HTML organizada y estructurada. Esta documentación incluye descripciones de las funciones, sus parámetros y valores de retorno, lo que resulta en una referencia completa y coherente de la API del plugin. Esta funcionalidad ahorra tiempo a los desarrolladores al automatizar la generación de documentación,

lo que facilita la comprensión y el uso del plugin por parte de otros miembros del equipo o de la comunidad.

5.2.3 Implementación en C++ con simPlusPlus

Otra herramienta fundamental para el desarrollo de plugins en CoppeliaSim es simPlusPlus, una biblioteca que ofrece una interfaz C++ orientada a objetos para interactuar con la API del software de simulación. Esta herramienta simplifica considerablemente el proceso de creación de plugins al abstraer gran parte de la complejidad técnica y proporcionar una estructura más intuitiva para trabajar con la API de CoppeliaSim.

simPlusPlus permite definir clases derivadas de la clase base CScriptFunction para crear y manipular funciones personalizadas que interactúan con la simulación. Estas funciones pueden ser llamadas desde scripts LUA en la simulación, o desde otras funciones C++ desde el plugin. Para comenzar el desarrollo, se crea una nueva instancia de una clase derivada de la clase *Plugin* de simPlusPlus. Luego, se registran las funciones que se desean asociar con la API de CoppeliaSim. Cada función personalizada se define mediante un método virtual que se sobrescribe en la clase derivada.

Un aspecto clave de simPlusPlus es que abstrae la gestión de argumentos y valores de retorno. Al registrar una función personalizada, se definen mediante punteros los tipos de datos de los argumentos y el valor de retorno, lo que permite que simPlusPlus se encargue automáticamente de la conversión y manipulación de datos entre los scripts LUA y la API de CoppeliaSim. Esto reduce drásticamente la posibilidad de errores y acelera el desarrollo al evitar la necesidad de realizar conversiones manuales de tipos de datos.

Además, simPlusPlus ofrece un enfoque más orientado a objetos para trabajar con objetos de la simulación, lo cual permite acceder y manipular objetos a través de instancias de clases C++, simplificando la interacción con la simulación en comparación con el uso directo de funciones y scripts Lua.

5.2.4 Compilación del plugin

La compilación de plugins en CoppeliaSim normalmente, se realiza mediante las herramientas de Qt, o CMake. En este caso se ha elegido CMake por su uso anterior en varias partes de este proyecto. El archivo CMakeLists.txt es la parte central del proceso de compilación del plugin, proporcionando instrucciones específicas para generar los archivos de construcción y compilar el código fuente. Es de crucial importancia que este archivo se encuentre en la carpeta raíz del plugin ya que la configuración se ha realizado relativa a esta dirección. A continuación, se describen los componentes clave de este archivo y su función dentro del proceso de compilación:

- Configuración de Versión y Nombre del Proyecto:
  - Se define la versión mínima requerida de CMake para el proyecto.
  - Se establece el estándar C++ utilizado en la compilación.
  - Se especifica el nombre del proyecto.
- Configuración de Rutas y Opciones:

  - Se establece la inclusión automática del directorio actual como ruta de búsqueda para archivos de inclusión.
  - Se habilita la opción de búsqueda en tiempo de ejecución en macOS.

- Rutas de Inclusión y Directorios Externos:
  - Configuración del directorio de inclusión de CoppeliaSim, ya sea por medio de la variable COPPELIASIM_INCLUDE_DIR o buscándolo en el entorno o ubicación predeterminada.
  - Se agregan rutas de búsqueda para módulos de CMake y archivos de inclusión de CoppeliaSim.

- Configuración de Paquete CoppeliaSim:
  - Se busca y configura el paquete CoppeliaSim 4.5.0.0 para asegurarse de que las bibliotecas y archivos de inclusión necesarios estén disponibles.

- Rutas de Inclusión Específicas:
  - Se agregan rutas de inclusión de directorios generados
  - Rutas de directorios externos al plugin en CoppeliaSim
  - Agrega todas las rutas de inclusión de los archivos utilizados de la biblioteca open62541.

- Generación de Stubs y Configuración de Archivos:
  - Se generan stubs a partir del archivo XML de callbacks para facilitar la comunicación entre el plugin y CoppeliaSim.
  - Se configura un archivo de encabezado para definir parámetros de configuración específicos del plugin.
  - Vinculación del archivo de llamadas de LUA entre el plugin y CoppeliaSim.

- Agregar el Plugin al Proceso de Compilación:
  - Se agrega el plugin al proceso de compilación mediante la función coppeliasim_add_plugin.
  - Se especifican los archivos que se deben incluir en la compilación.

- Vinculación de Bibliotecas:
  - Se vincula la biblioteca OPC UA libopen62541.a con el plugin, permitiendo la comunicación efectiva entre el plugin y el servidor OPC UA.

Con este archivo se puede construir el programa de forma similar a la construcción del programa del servidor descrita en la subsección 4.4.4, mediante los comandos:

```
cmake -DCMAKE_BUILD_TYPE = Release ..
cmake –build .
```

Ambos comandos han de ser ejecutados desde la carpeta build (o cualquier otro nombre) ubicada en la carpeta raíz del plugin. El resultado de ejecutar estos comandos, son los archivos mostrados en la Figura 5.8. Finalmente, el comando *cmake –install .* copia los archivos compilados, como el plugin compilado y otros recursos necesarios, en las ubicaciones especificadas en las directivas de instalación del archivo CMakeLists.txt.

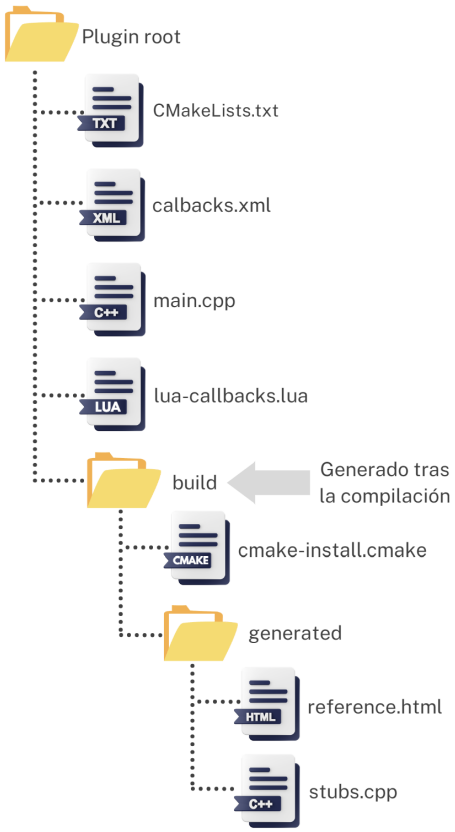


Figura 5.8: Diagrama de árbol con los principales componentes del plugin para CoppeliaSim.

5.3 Programa principal del plugin del gemelo digital

5.3.1 Estructura del programa

Tal y como se ha comentado anteriormente, el plugin implementa el cliente de OPC UA (véase sección 4.5) y la comunicación con CoppeliaSim sobre la cual trata esta sección. En la Figura 5.9 se presenta la estructura del plugin.

La parte central de los plugins en CoppeliaSim es el archivo de código escrito en C++ (o Python), que da lugar a la creación de una instancia derivada de la clase base Plugin, proporcionada por la biblioteca simPlusPlus. Al heredar de esta clase, se obtiene acceso a una serie de funciones y métodos que posibilitan la interacción con CoppeliaSim desde el nuevo plugin en desarrollo.

Una función fundamental en el plugin es onStart, la cual es invocada automáticamente cuando el plugin se carga en el entorno de simulación. En este punto, es común llevar a cabo tareas de inicialización, como el registro de funciones personalizadas y la configuración de parámetros

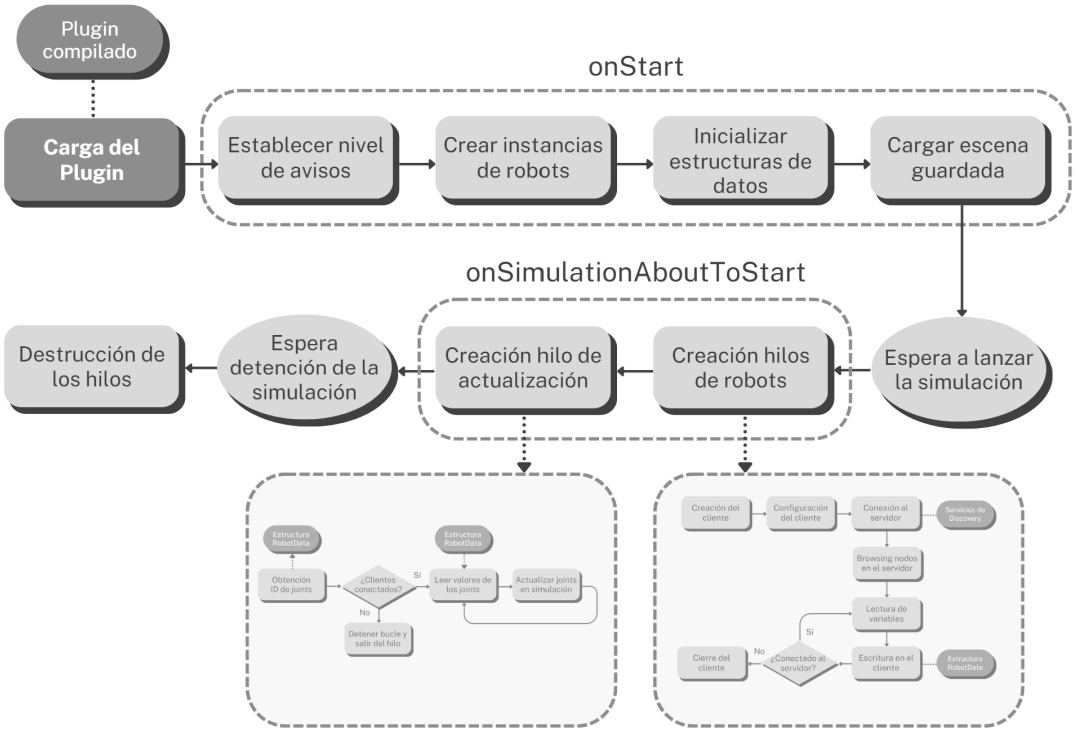


Figura 5.9: Esquema completo de la estructura del plugin para CoppeliaSim.

específicos del plugin. Además de onStart, otro componente crítico es la función onSimulationAboutToStart, la cual se activa justo antes de que la simulación comience. Este punto de ejecución es idóneo para la configuración y el inicio de hilos de ejecución paralela que desempeñarán tareas en segundo plano mientras la simulación está en curso. Es importante tener en cuenta que bloquear la ejecución del plugin en cualquier momento podría afectar el flujo de la simulación en sí.

La función onStart ofrece un espacio propicio para realizar tareas cruciales que preparan el terreno para el funcionamiento del plugin durante la simulación. Esta función inicializa los parámetros de las estructuras de datos compartidas de acuerdo con el número de robots que se van a simular. Además, se carga la escena previamente configurada en la sección anterior mediante *simLoadScene*. Aquí se pueden establecer también los niveles de detalle de los registros y la depuración mediante la función de CoppeliaSim *setModuleInfo*.

Por otra parte, la función onSimulationAboutToStart desempeña un papel esencial permitiendo preparar la ejecución antes de que la simulación se inicie. Al ser invocada justo antes de que comience la ejecución de la simulación, esta función lleva a cabo tareas fundamentales que requieren sincronización con el entorno de la simulación. Su valor estratégico radica en la posibilidad de configurar el entorno, activar hilos de ejecución secundarios y establecer las bases para el funcionamiento sin problemas del gemelo digital durante toda la duración de la simulación.

Una de las primeras acciones realizadas en esta función es la declaración de un vector que contendrá los hilos de los clientes OPC UA. A través de un bucle, se crean tantos hilos como robots se prevean en la simulación. Esto garantiza que cada robot tenga su propio cliente OPC UA conectado al servidor centralizado, lo que habilita la transferencia de datos desde sus homólo-

gos físicos. Cada hilo se inicia mediante la función opcuaThreadFunction, a la que se le pasa una instancia de la estructura de datos compartida denominada "RobotData". Esta estructura almacena información relevante para cada robot, incluyendo identificadores, variables del robot físico, estados de ejecución, etc. La asignación de un hilo individual a cada robot es crucial para la sincronización y el paralelismo, ya que permite la comunicación y el intercambio de datos simultáneos entre los robots y el servidor OPC UA.

Además de los hilos de los clientes OPC UA, se crea un hilo adicional que desempeña un papel fundamental en la actualización continua de la simulación. Este hilo tiene la responsabilidad de integrar los datos recibidos de las estructuras de datos compartidas en la simulación en tiempo real. Para lograr esto, se pasa por referencia un vector que contiene todas las estructuras de datos compartidas de los robots. Esto permite que el hilo acceda eficientemente a los datos actualizados y los aplique a la simulación en curso.

En términos de eficiencia y rendimiento, es esencial que los hilos se gestionen cuidadosamente en términos de sincronización y bloqueo. La creación de múltiples hilos paralelos requiere la coordinación adecuada para evitar conflictos de acceso a los datos compartidos y posibles bloqueos que podrían impactar en el rendimiento general de la simulación. Por ello se utilizan semáforos POSIX individuales para cada uno de los hilos. Estos semáforos son almacenados en las estructuras de datos compartidas.

Antes de que un hilo acceda a los datos compartidos, debe adquirir el semáforo correspondiente utilizando la función sem_wait. Esto bloquea el semáforo si está en uso por otro hilo, asegurando que solo un hilo tenga acceso a los datos en un momento dado. Una vez que el hilo ha terminado de utilizar los datos, libera el semáforo utilizando la función sem_post, permitiendo que otros hilos accedan a los mismos recursos.

La utilización de semáforos de esta manera garantiza una sincronización adecuada entre los hilos. Cuando un hilo adquiere un semáforo, se asegura de que ningún otro hilo pueda modificar los mismos datos simultáneamente, evitando condiciones de carrera y resultados incoherentes. Esto es especialmente crítico en el contexto de la actualización de la simulación, donde la coherencia de los datos es fundamental para una representación precisa del gemelo digital.

5.3.2 Hilo de actualización de datos del programa principal

El hilo de actualización de datos creado en la función onSimulationAboutToStart, está únicamente operado por la función updateSimulation (Véase Figura 5.10), lo que permite que el proceso de actualización se ejecute de manera paralela sin bloquear el hilo principal de la simulación. Esto optimiza el rendimiento general al tiempo que asegura que los datos recopilados sean precisos.

El núcleo de esta función es un bucle continuo, en que que se itera a través de las instancias de robots presentes en la simulación. Previamente, antes de entrar a este bucle se han obtenido los identificadores únicos, denominados "handles", de los joints de la instancia de robot. Estos handles, han sido guardados en la estructura RobotData para poder ser accedidos desde cualquier función del plugin. Dentro del bucle principal, para cada instancia de robot, se recuperan las ID de las articulaciones y los valores de posición almacenados en la estructura de datos compartida. Estos valores de posición han sido escritos en las estructuras compartida propias de cada uno de los hilos que implementan los clientes OPC UA, tal como se explicó en la sección 4.5. Tras haber

accedido a estos datos, se utiliza la función *simSetJointTargetPosition* de CoppeliaSim para actualizar las posiciones objetivo de las articulaciones en el entorno virtual con las posiciones actuales capturadas.

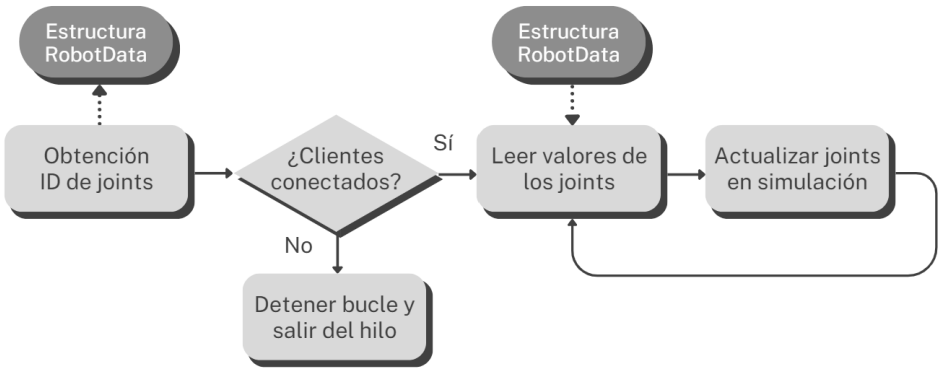


Figura 5.10: Diagrama de flujo del hilo de actualización de la simulación del plugin para CoppeliaSim.

La sincronización de hilos es un aspecto crítico para evitar conflictos de acceso a las estructuras de datos compartidas. Al igual que en la función opcuaThreadFunction, se utilizan semáforos POSIX para asegurar que la actualización de datos y la simulación se ejecuten de manera coordinada y se eviten condiciones de carrera. De esta forma, en el momento previo al acceso de los datos de posición (zona crítica), se bloquea el semáforo propio de esa estructura de datos compartida, restringiendo el acceso a la funciona de escritura opcuaThreadFunction hasta que los datos han sido correctamente leídos en el bucle.

Además, la función updateSimulation está diseñada para detectar el cierre o finalización de la simulación. Cuando esto ocurre, el bucle se detiene y se asegura de liberar cualquier recurso utilizado antes de terminar. Esto contribuye a una terminación ordenada y sin fugas de recursos, manteniendo la integridad del entorno de simulación.

Capítulo 6

Resultados y análisis del Gemelo Digital

Este capítulo se presenta en los resultados obtenidos y el análisis detallado del Gemelo Digital desarrollado. A lo largo de este proyecto, como la planificación del proyecto presentada en el B detalla, se ha trabajado en la mejora continua de la aplicación, así como en su validación en diferentes escenarios. Por ello, se ha dedicado la primera sección, a exponer las técnicas y herramientas utilizadas para perfeccionar el funcionamiento de la aplicación. Esta sección se enfoca principalmente en el análisis y la corrección de posibles errores en el código del servidor, el componente vital que gestiona la recopilación de datos y el servidor OPC UA. AddressSanitizer y Valgrind, son las dos herramientas fundamentales que han desempeñado un papel destacado en este proceso de mejora, proporcionando una visión profunda del comportamiento del código.

En las dos secciones siguientes se presentan el análisis y lso resultados de varias pruebas realizadas sobre la aplicación del Gemelo Digital. Primero se presenta una prueba cuantitativa consistente en una evaluación detallada del Gemelo Digital conectado sincronizado con un robot Universal Robots UR5 en un entorno de laboratorio real. El objetivo es demostrar la capacidad del Gemelo Digital para replicar con precisión el comportamiento del robot físico. Se analizarán los datos recopilados durante esta evaluación y se compararán con los datos del robot real.

A continuación, la tercera sección del capítulo, se centra en los resultados de pruebas realizadas en varios escenarios para evaluar la aplicación del Gemelo Digital en situaciones diversas. Estas pruebas se llevaron a cabo en dos entornos distintos con robots simulados y robots reales. El análisis cualitativo de esta sección, se concentra en la capacidad del Gemelo Digital para gestionar la interacción entre múltiples robots en entornos colaborativos y responder eficazmente a situaciones dinámicas.

6.1 Mejora y Optimización del Funcionamiento de la Aplicación

La mejora constante es una parte esencial del proceso de desarrollo de software. En esta sección, se exploran las técnicas y herramientas utilizadas para perfeccionar el funcionamiento de la aplicación, centrándose en el análisis y la corrección de posibles errores en el código del servidor. Este componente desempeña un papel vital en la aplicación al gestionar la recopilación de datos y el servidor OPC UA.

La sección se centra en las dos herramientas clave que han jugado un papel fundamental en este proceso de mejora: AddressSanitizer y Valgrind. Estas herramientas proporcionan una visión profunda del comportamiento del código y permiten identificar y solucionar errores de memoria, fugas de memoria y otros problemas que podrían afectar negativamente al rendimiento y la estabilidad de la aplicación. A través de su uso, se ha logrado una base sólida para un funcionamiento robusto y confiable del servidor, lo que es esencial para una aplicación crítica en la industria 4.0 como es este gemelo digital.

La principal herramienta utilizada en la depuración y optimización del código de esta aplicación ha sido Visual Studio. Las características de depuración avanzadas de este entorno de desarrollo facilitan la identificación y corrección de problemas en el código de manera efectiva y eficiente.

Una de las características destacadas de Visual Studio es la capacidad para configurar Breakpoints de forma precisa. Estos puntos de interrupción no solo detienen la ejecución del programa, sino que también pueden condicionarse, lo que significa que se activan en situaciones específicas. Esta funcionalidad es especialmente útil para detectar comportamientos específicos y comprender el flujo del programa en detalle.

En paralelo a los Breakpoints, la posibilidad de inspeccionar variables duranet la ejecución del programa, denominado *on runtime*, proporciona una comprensión profunda del estado del programa en diferentes etapas de la ejecución. El nivel de introspección que proporciona esta ejecución detallada es esencial para resolver problemas complejos y optimizar el rendimiento del programa.

Visual Studio también ofrece ventanas especializadas para el seguimiento de variables concretas. La ventana de *Watches* permite monitorear variables, brindando información durante la ejecución del programa sobre sus valores y cambios. Paralelamente, la ventana Autos ofrece un contexto más dinámico al mostrar automáticamente las variables relevantes durante la depuración. Estas características minimizan la necesidad de saltar entre líneas de código y ventanas, agilizando significativamente el proceso de depuración.

En situaciones en las que el desarrollo y la ejecución se realizan en entornos separados, como sistemas empotrados, Visual Studio permite también la depuración remota. Esta característica habilita a los programadores a examinar y solucionar problemas en aplicaciones que se ejecutan en máquinas remotas o en dispositivos físicos. Asimismo, la función *Attach* a Procesos permite conectar el depurador a procesos en ejecución, lo que resulta de gran utilidad a la hora de abordar problemas en tiempo real y realizar ajustes inmediatos.

En las aplicaciones escritas en lenguaje C++, la gestión de la memoria es un concepto fundamental. En las siguientes secciones se introducirán las dos herramientas de depuración de memoria, Valgrind y AddressSanitizer utilizadas en el proyecto. Valgrind, es una herramienta ampliamente utilizada que permite detectar fugas de memoria, uso incorrecto de memoria y otros proble-

mas relacionados con la memoria dinámica. Por otro lado, AddressSanitizer está integrado en compiladores modernos de C++ e identifica eficazmente el uso de memoria no válida y las fugas.

Tanto Valgrind como AddressSanitizer son herramientas extremadamente útiles para detectar errores de memoria y problemas en programas. Sin embargo, ambas herramientas pueden generar falsos positivos en ciertas situaciones. En el proyecto se ha decidido combinar el uso de ambas herramientas para brindar así una mayor confianza en la detección de problemas. En caso de que ambas herramientas señalan un problema en el mismo lugar, es más probable que sea una situación real de error, mientras que, si solo una de las herramientas señala el problema, podría tratarse de un falso positivo. En dicho caso se debería examinar más de cerca la parte del código que ha generado el error y determinar si es un problema real o no

6.1.1 Depuración de memoria con AddressSanitizer

AddressSanitizer, a menudo abreviado como ASan, es una herramienta de depuración y análisis estático diseñada para detectar y mitigar errores de memoria en programas escritos en C++ y otros lenguajes compatibles con LLVM. Conocida por su eficacia en la identificación de problemas de memoria, AddressSanitizer se ha convertido en una herramienta esencial para mejorar la robustez y la estabilidad de una gran variedad de aplicaciones.

Esta herramienta fue desarrollada originalmente por Konstantin Serebryany y otros ingenieros en Google, como una respuesta a la necesidad de abordar los errores de memoria que a menudo plagan los programas escritos en C++ y otros lenguajes de sistema. Fue lanzada por primera vez en 2011 como parte de LLVM (Low Level Virtual Machine), un conjunto de compiladores y herramientas de análisis estático que se utiliza para el desarrollo de diversos lenguajes de programación.

AddressSanitizer destaca por su capacidad para detectar varios tipos de errores de memoria, como fugas de memoria, acceso incorrecto a la memoria (como lecturas y escrituras fuera de límites) y uso de punteros inválidos. Durante la compilación, la herramienta añade una capa de detección de errores que permite identificar problemas de memoria en tiempo de ejecución, facilitando así la depuración y corrección temprana de estos fallos.

Esta herramienta cobra gran relevancia en la identificación y resolución de errores de memoria que, de otra manera, podrían pasar desapercibidos durante el desarrollo. Su uso en aplicaciones industriales es especialmente valioso para garantizar la integridad del código. De hecho, uno de sus usos principales es en servidores, ya que son aplicaciones críticas que requieren alta disponibilidad, estabilidad y seguridad. Los servidores, al gestionar cargas de trabajo pesadas y concurrentes, son susceptibles a los errores de memoria, los cuales pueden impactar significativamente en su rendimiento y seguridad.

- Identificación temprana de errores: Puesto que los servidores son aplicaciones críticas que deben funcionar continuamente sin interrupciones, es importante identificar errores de memoria en etapas tempranas del desarrollo, lo que permite corregirlos antes de que se implemente el servidor en producción.
- Mejora de la estabilidad: Los errores de memoria pueden causar bloqueos y comportamientos inesperados en el servidor. Una estabilidad general del servidor permite garantizar un servicio de confianza para los usuarios.

- Reducción de vulnerabilidades de seguridad: Muchas vulnerabilidades de seguridad, como los desbordamientos de búfer, están relacionadas con errores de memoria, los cuales pueden poner en peligro las aplicaciones y datos del servidor.
- Optimización del rendimiento: Los errores de memoria suelen afectar negativamente el rendimiento del servidor al causar fugas de recursos.
- Ahorro de tiempo y recursos: Identificar y resolver problemas de memoria en un servidor en producción puede ser complicado y costoso en términos de tiempo y recursos.Por ello la detección y solución temprana de estos problemas, ahorra tiempo y esfuerzo a largo plazo.

AddressSanitizer puede ser implementado mediante compilaciones de línea de comandos, sistemas de proyectos de Visual Studio o la creación de tests CMake que incluyan ASan. Para aplicaciones con pocas dependencias, la compilación de línea de comandos permite integrar ASan de manera directa y sencilla. En este proyecto, se ha optado por la opción de CMake, la cual ofrece una estructura más organizada y flexible para la construcción del software. En la Figura 6.1 se muestra un fragmento de código del archivo CMakeLists.txt implementando el uso del AddressSanitizer.

```
# testing binary with ASan
if(CMAKE_BUILD_TYPE MATCHES "Debug_asan")

    set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fsanitize=address")

    add_executable(ASanTest
                                ${UA_NODESET_DI_SOURCES}
                                ${UA_NODESET_ROBOTICS_SOURCES}
                                ${UA_NODESET_UNIVERSAL_ROBOTS_SOURCES}
                                ${UA_TYPES_DI_SOURCES} #types_di_generated.c
                                ${UA_TYPES_ROBOTICS_SOURCES} #namespace_di_generated.c
                                ${UA_TYPES_UNIVERSAL_ROBOTS_SOURCES}
                                src/UR-server.cpp)

    target_compile_options(ASanTest PRIVATE -fsanitize=address)
    target_link_options(ASanTest PRIVATE -fsanitize=address)

    target_link_directories(ASanTest PRIVATE /home/oem/ur_rtde/build/librtde.so)
    target_link_libraries(ASanTest /home/oem/ur_rtde/build/librtde.so)

    add_dependencies(ASanTest
                                        open62541-generator-ns-di
                                        open62541-generator-ns-robotics
                                        open62541-generator-ns-universal_robots
                                        )
    target_link_libraries(ASanTest open62541::open62541)
endif()
```

Figura 6.1: Fragmento del archivo CMake implementando un test con la herramienta AddressSanitizer.

LeakSanitizer, una extensión de AddressSanitizer, está diseñada específicamente para identificar fugas de memoria, un problema común en programas C++. En caso de encontrar un fallo de memoria, la salida de ASan indicará la dirección y detalles del objeto donde se detectó el error, seguido de un rastreo de las asignaciones de memoria relacionadas. Si el fallo está relacionado con otros aspectos de la gestión de memoria, como desbordamientos de búfer, la salida proporcionará información detallada sobre el bloque de memoria afectado y su estado.

En el caso de tener otros fallos relacionados con la gestión de la memoria como desbordamientos de búfer, punteros colgantes u otros, la salida mostrada por ASan cambiará a una similar a la mostrada en al Figura 6.2. En este caso, la primera línea indica que el tipo de fallo, un desbordamiento del búfer. A continuación, se muestra una tabla representando estado de los bloques de memoria (heap) que el programa está utilizando. Los distintos códigos de colores y letras indican los estados de los bloques de memoria. En el caso de la Figura 6.2, se observan

bloques de memoria disponibles (números en blanco), bloques utilizados por el programa, y bloques de memoria liberados. La herramienta indica en esta tabla el bloque de memoria donde ha ocurrido el error.

Siguiendo la tabla de bloques de memoria, la herramienta (dependiendo de la configuración) presenta un breve resumen de los programas y funciones relevantes del rastreo de pila del error (traceback). Luego presenta el rastreo de pila de las funciones y llamadas mucho mas detallado, comenzando con la función en la que ocurrió el error y retrocediendo a través de las llamadas de función hasta llegar al punto en el que se llamó al programa. Cada entrada en el traceback muestra la dirección de memoria de la función y el archivo y línea de código donde se realizó la llamada.

```
SUMMARY: AddressSanitizer: heap-buffer-overflow ../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:827
Shadow bytes around the buggy address:
  0x0c0880010030: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd
  0x0c0880010040: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd
  0x0c0880010050: fa fa fd fd fd fd fd fd fa fa fd fd fd fd fd fd
  0x0c0880010060: fa fa 00 00 00 00 02 fa fa fa 00 00 00 00 00 00
  0x0c0880010070: fa fa 00 00 00 00 00 00 fa fa fd fd fd fd fd fa
=>0x0c0880010080: fa fa 00 00 00 00 00 00[fa]fa fa fa fa fa fa fa
  0x0c0880010090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c08800100a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c08800100b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c08800100c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c08800100d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
```

Figura 6.2: Salida en línea de comandos de la herramienta AddressSanitizer mostrando un desbordamiento de búfer.

La información proporcionada por AddressSanitizer es invaluable para identificar y resolver problemas de memoria en el código, ya que permite ubicar el origen del error y entender cómo se llegó a ese estado en el programa. En este proyecto, la experiencia con el uso de ASan ha sido satisfactoria, especialmente en la detección de errores de memoria, como accesos fuera de límites, desbordamientos de búfer y usos después de liberar. Cualquier falso positivo detectado ha sido verificado utilizando la herramienta Valgrind.

6.1.2 Detectección de errores con Valgrind

Valgrind fue creado por Nicholas Nethercote y Julian Seward en el año 2000 como parte de un proyecto de investigación en la Universidad de California, Berkeley. Su objetivo principal era abordar los desafíos asociados con la detección de problemas de memoria en programas escritos en lenguajes como C y C++. A lo largo de los años, Valgrind ha evolucionado y se ha convertido en una herramienta ampliamente adoptada por la comunidad de desarrollo de software para mejorar la calidad del código y reducir los errores relacionados con la memoria.

La robustez y versatilidad de Valgrind se manifiestan a través de su conjunto de instrumentos especializados diseñados para abordar una amplia gama de problemas relacionados con la memoria y la concurrencia en programas escritos en C y C++. Cada uno de estos instrumentos aporta

una perspectiva única a la depuración, lo que permite a los desarrolladores identificar y resolver problemas específicos con mayor precisión.

Uno de los instrumentos clave de Valgrind es Massif, una herramienta destinada a analizar el consumo de memoria de un programa a lo largo del tiempo. Massif realiza un seguimiento detallado de las asignaciones y liberaciones de memoria, generando un perfil de uso que permite identificar picos en el consumo y áreas en las que la optimización de la memoria puede ser beneficiosa. Esta herramienta es especialmente en proyectos que involucran grandes conjuntos de datos.

Helgrind es otro de los componentes principales de Valgrind que se centra en la detección de problemas de concurrencia, como condiciones de carrera. En entornos multihilo, donde varios hilos de ejecución interactúan, los problemas de concurrencia pueden ser difíciles de identificar y corregir. Helgrind analiza el flujo de ejecución de los hilos y señala situaciones en las que podría ocurrir un comportamiento no deseado. Esto es esencial para asegurar la estabilidad y la coherencia en aplicaciones que emplean concurrencia para mejorar el rendimiento.

Sin embargo, el instrumento más conocido y utilizado de Valgrind es Memcheck, que se concentra en la detección de problemas de memoria, como fugas y referencias a memoria no inicializada. Memcheck realiza un rastreo de la memoria asignada, lo que permite identificar cuando la memoria no se libera adecuadamente o se accede de manera incorrecta. Esta herramienta es esencial para asegurar la estabilidad y seguridad de una aplicación, especialmente en los entornos industriales donde los problemas de memoria pueden causar comportamientos inesperados y potencialmente críticos.

Para utilizar Valgrind se ha de comenzar por instalarlo y configurarlo en el sistema. Una vez configurado, el proceso de ejecución implica simplemente llamar a Valgrind con el ejecutable del programa a analizar. Por ejemplo, para analizar el programa del servidor ür-server", el comando para usar Valgrind sería: *valgrind ./ur-server*.

Una vez que se ha ejecutado el análisis con Memcheck, los resultados proporcionados son esenciales para identificar y abordar problemas de memoria. Las salidas de Memcheck pueden parecer abrumadoras al principio debido a la cantidad de información proporcionada. Sin embargo, es importante comprender que no todos los resultados son necesariamente indicativos de problemas críticos, y algunos mensajes pueden referirse a áreas inofensivas o errores conocidos del sistema operativo.

Debido esta gran cantidad de datos proporcionados, es importante centrar la atención en los resúmenes de los errores presentados al comienzo de cada sección. Estos resúmenes o sumarios ayudan a identificar fragmentos de código susceptibles de ser optimizados o sospechosos de contener fallos de memoria. Una vez que los posibles errores se han identificado se ha de proseguir leyendo la información presentada a continuación del sumario. En estas líneas se incluye información detallada sobre la ubicación en el código donde se produjo el problema y un traceback del código que lo produjo.

La utilización de Valgrind en el proyecto, en particular la herramienta Memcheck, ha permitido solucionar todos problemas de memoria del código y asegurarse de la aplicación crítica del servidor sea robusta y, libre de fugas y accesos incorrectos a la memoria. Interpretar los resultados de Memcheck requiere un enfoque cuidadoso para diferenciar entre problemas críticos y mensajes

inofensivos, tal como ocurría con ASan. No obstante, el uso de las dos herramientas ha resultado complementario y muy beneficioso en el proyecto.

6.2 Comparación Cuantitativa con un Robot UR5 Real

En esta sección, se presenta una evaluación del gemelo digital desarrollado en este proyecto, el cual ha sido conectado y sincronizado con un robot Universal Robots UR5 en un entorno de laboratorio. El propósito de esta evaluación es demostrar la capacidad del gemelo digital para replicar con precisión el comportamiento del robot real.

6.2.1 Configuración experimental de los ensayos

La realización de los ensayos se llevó a cabo en un entorno especialmente configurado que involucra varios componentes clave. Esta configuración se diseñó meticulosamente para permitir un análisis completo del comportamiento del gemelo digital en relación con un robot físico UR5. A continuación, se describe detalladamente la configuración utilizada en los ensayos.

En el núcleo de la configuración, como se discutió previamente en la subsección 4.1.1, se utilizó una placa LattePanda Delta V3, ejecutando el programa del servidor OPC.Este programa, como se describió en detalle a lo largo de la sección de comunicación, se encarga de dos tareas principales: la adquisición de datos mediante la biblioteca RTDE y la creación y actualización del servidor OPC UA. En la Figura 6.3 se muestra de forma resumida las partes de la aplicación y los puntos donde se tomarán los datos.

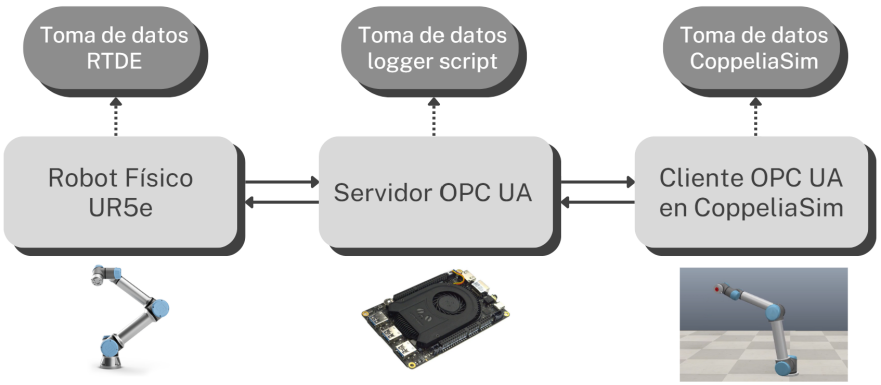


Figura 6.3: Diagrama de los puntos de adquisición de datos del ensayo con el robot UR5.

La simulación del gemelo digital se realizó en un portátil Lenovo Thinkpad E14 con el sistema operativo Ubuntu 22.04. Este portátil ejecutó CoppeliaSim, el entorno de simulación utilizado para crear y analizar el gemelo digital. Dentro de CoppeliaSim, se creó una escena específica llamada "UR5 Digital Twin" que se centró en representar con precisión el robot UR5 y sus variables medidas. Además se adaptaron algunas líneas del código del plugin para graficar los datos leídos por el cliente en la simulación.

Para facilitar la comunicación efectiva entre los componentes, se utilizó una red local de la Universidad Politécnica de Valencia (UPV). Mientras que el robot estaba conectado a una subred de esta red local, la placa LattePanda y el portátil Lenovo Thinkpad se conectaron a esta red

local directamente. Esto permitió que la biblioteca RTDE del programa del servidor OPC UA se comunicara con el robot físico y que los clientes OPC localizaran y accedieran al servidor en la red. El robot UR5 utilizado en los ensayos se encuentra en las instalaciones del Instituto de Automática e Informática Industrial de la UPV. Esta ubicación específica y su identificación en la red en la que se encontraba conectado fueron esenciales para garantizar una conexión estable y confiable durante la ejecución de los ensayos.

Durante las pruebas, se seleccionaron varios puntos de análisis críticos y variables de interés. Estos puntos se centraron en la recopilación de datos relacionados con las coordenadas de posición del TCP, ya que a diferencia de otras variables disponibles, la posición del robot puede interpretarse de forma más sencilla, lo que resulta realmente útil para la evaluación del comportamiento del gemelo digital.

Para la toma de datos el robot físico se programó en un bucle para simular una aplicación de "pick and place". El bucle infinito mueve el robot comenzando en la posición denominada pre-pick, hasta la posición denominada pick mediante una orden *moveL* a baja velocidad (Figura 6.4a). Una vez en la posición de pick, se detiene unos segundos dando tiempo a una posible herramienta montada en el robot a coger el objeto. Después vuelve con el mismo movimiento a la posición de pre-pick. A continuación se mueve a una velocidad baja hasta la posición de pre-place mediante una orden *moveJ*. En este punto realiza el movimiento de pre-place a place tal como muestra la Figura 6.4b con un comando *moveL*, en la posición de place espera un tiempo y vuelve a la posición de pre-place. Desde aquí se mueve rápidamente hasta la posición de pre-pick para repetir el ciclo.

6.2.2 Recopilación de los Datos

La recopilación de datos se lleva a cabo en tres puntos distintos dentro de la estructura de la aplicación. Como se describe en la Figura 6.3, los datos adquiridos en estos puntos se refieren a las coordenadas de posición del punto central de la herramienta en el robot UR5, el servidor OPC UA, y el cliente OPC UA dentro de CoppeliaSim. Cada punto de recopilación de datos desempeña un papel específico en la obtención de información relevante.

El primer punto de recopilación de datos se ubica en el robot UR5. La razón de este punto es poder obtener valores de coordenadas de la posición real del robot en para utilizarlos como referencia en los análisis posteriores. Para lograr esto, se ha desarrollado un programa en C++ que utiliza la biblioteca RTDE. Este programa se conecta a la dirección IP del robot mediante la interfaz Receive, de manera similar a la utilizada en el bucle de extracción de datos del programa servidor. Los datos de posición del TCP se leen del robot y luego se escriben en un archivo de texto mediante el uso de las bibliotecas de manejo de ficheros e C++. Un fragmento de este programa se muestra en la Figura 6.5.

El segundo punto de recopilación de datos se encuentra en el propio servidor OPC UA. En este punto, se ha realizado una modificación en el código del servidor de tal manera que, en cada ciclo de actualización de los datos, el hilo del servidor (encargado de la escritura de los datos en el servidor, como se explicó en la subsección 4.4.2) realiza la escritura de los datos en un archivo de texto. Esto se hace de manera secuencial con la actualización de los datos en el servidor, de esta forma se pretende asegurar que los datos enviados para ser escritos en ele servidor sean los mismos que los extradidos en el fichero de texto.

(a) Movimiento de la posición de Pre-Pick a Pick.

(b) Movimiento de la posición de Pre-Place a Place

Figura 6.4: Posiciones programadas en el robot UR5 del laboratorio del instituto ai2.

El enfoque adoptado aquí permite obtener una visión de los datos recopilados por la RTDE y escritos el servidor OPC UA. Este enfoque es esencial para verificar el correcto comportamiento de la conexión entre el robot físico y servidor alojado en la placa LattePanda, lo que proporciona una comprensión más profunda del comportamiento de la comunicación RTDE.

Para garantizar el correcto comportamiento de todos los agentes involucrados en la comunicación OPC UA y confirmar que los datos se están adquiriendo y almacenando de manera adecuada en el servidor, se ha incorporado un punto extra de recopilación de datos. En este caso, se utilizó

```cpp
#include <ur_rtde/rtde_receive_interface.h>
#include <iostream>
#include <fstream>
#include <thread>   // Include the thread library for sleep
#include <chrono>   // Include the chrono library for time-related functions

int main() {
    ur_rtde::RTDEReceiveInterface rtdeReceive("192.168.1.3");

    std::ofstream outputFile("tcp_positions.txt");

    if (!outputFile.is_open()) {
        std::cerr << "Failed to open the file for writing." << std::endl;
        return 1;
    }

    while (true) {
        std::vector<double> tcpPose = rtdeReceive.getActualTCPPose();

        if (tcpPose.size() != 6) {
            std::cerr << "Received an incomplete TCP pose." << std::endl;
            continue;
        }

        for (int i = 0; i < 3; i++) {
            outputFile << tcpPose[i] << " ";
        }
        outputFile << "\n";

        outputFile.flush();
```

Figura 6.5: Fragmento de código C++ del programa utilizado para la recolección de datos del robot UR5.

un cliente comercial de OPC UA, Prosys Monitor, para conectarse al servidor y recopilar los datos directamente desde él, haciendo uso de las funciones de suscripción y registro de datos del software. Las gráficas de los datos obtenidos con el cliente Prosys se muestran en la Figura 6.6.

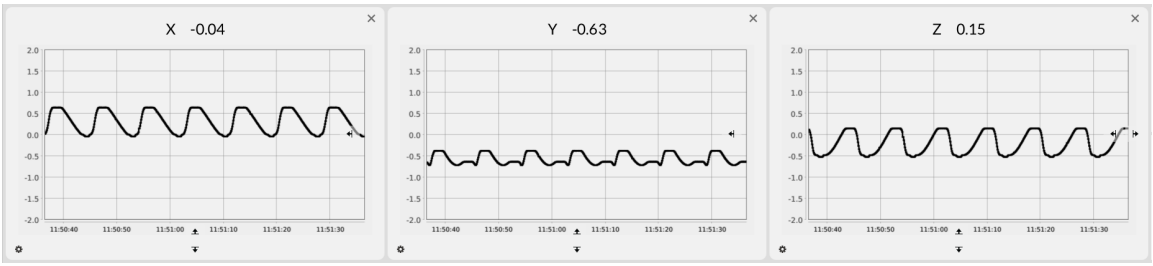


Figura 6.6: Gráficas de las tres coordenadas de posición del TCP del robot UR5 leídas del servidor OPC UA con el cliente Prosys Monitor.

Este enfoque de validación adicional se considera crucial para asegurarse de que se hayan creado las instancias de robot y sus nodos de manera correcta en el servidor, y que los valores se escriban en los nodos correctos. Los datos obtenidos a través de Prosys Monitor proporcionan una verificación independiente de que el servidor está funcionando correctamente y que los datos son accesibles desde fuentes externas al sistema.

La validación de la comunicación OPC UA se completa con la recopilación de datos desde la simulación en CoppeliaSim. El objetivo aquí es confirmar que tanto los clientes OPC UA integrados en el plugin de CoppeliaSim, como el thread que actualiza las variables en la simulación funcionan de manera correcta.

Para lograrlo, se realizaron las siguientes acciones:

- Cliente OPC UA: En este caso, puesto que la escena en CoppeliaSim tiene únicamente un solo robot la actualización de los thread de los clientes afecta sólo a un único thread. En

este thread del cliente OPC UA dentro del plugin de CoppeliaSim se añadió la lectura de los datos de posición del TCP de los correspondientes nodos del servidor OPC UA. Para ello se utilizaron las mismas funciones descritas en la sección 5.3, las cuales permitieron buscar los nodos en el servidor y guardar los datos de estos nodos en la estructurad e datos compartida.

```
unsearchedNodes.clear();
sim::addLog(sim_verbosity_infos,"Browsing opc ua server %s -> CartesianCoordinates nodes ...", robUR.name);
targetBrowseName = "CartesianCoordinates";
std::vector<UA_NodeId> tcpPositionsFatherNode;
std::vector<UA_NodeId> ansPositionsNode;
browseForNodesRecursive(client, &rootRobotsNodes[0], targetBrowseName, tcpPositionsFatherNode);
// Coordinates
unsearchedNodes.clear();
targetBrowseName = "X";
browseForNodesRecursive(client, &tcpPositionsFatherNode[0], targetBrowseName, ansPositionsNode);
robUR.tcpPositionsNodes.push_back(ansPositionsNode[0]);
```

Figura 6.7: Fragmento de código C++ del plugin implementando la obtención del nodo de la coordenada X de la posición del TCP

- Manipulación de Elementos Gráficos: Se añadieron líneas de código al plugin de CoppeliaSim para manipular los elementos gráficos dentro de la escena de simulación. Estos elementos gráficos representan las variables medidas del robot, como coordenadas de posición del TCP, corriente, voltaje y fuerzas generalizadas en las articulaciones. En la Figura 6.8 se muestran las líneas añadidas en el hilo de actualización de datos que permiten obtener la id en este caso del gráfico de las posiciones del TCP y actualizar los valores de estas dentro del bucle principal.

```
// Retrieve graph handle
std::string graphName = "./PositionGraph";
int positionsGraphHandle = simGetObject(graphName.c_str(),-1,-1,0);
std::string unitStr = "rad";
std::vector<std::string> streamNames = {"TCP Pos X","TCP Pos Y","TCP Pos Z"};
std::vector<int> positionsGraphStreamIds;
for (int i = 0; i < 3; ++i)
{
    // Use distinct RGB values for each curve
    const float red = i == 0 ? 1.0f : 0.0f;
    const float green = i == 1 ? 1.0f : 0.0f;
    const float blue = i == 2 ? 1.0f : 0.0f;

    const float color[3] = {red, green, blue};
    positionsGraphStreamIds.push_back(simAddGraphStream(positionsGraphHandle, streamNames[i].c_str(), unitStr.c_str(), 0, color, 0));
}

sem_wait(&serverAccess);
sim::addLog(sim_verbosity_infos,"Starting the simlation update loop ...");
bool allThreadsRunning = true;
while (allThreadsRunning)
{
    // Updates one after each other robot instance
    for (int instance = 0; instance < robotInstances.size(); ++instance)
    {
        RobotData& rob = (robotInstances)[instance];
        sem_wait(&(rob.positions_sem));
        for (int i = 0; i < 6; ++i)
        {   // Retrieve joint's handles and updates the positions
            int handle = rob.handles[i];
            double position = rob.jointPositions[i];
            if (i == 1 || i == 3)
                position = position + (M_PI/2); //Lab UR5
            simSetJointTargetPosition(handle, position);
        }
        for (int i = 0; i < 3; ++i)
        {   // Updates position of the TCP
            double tcpPosition = rob.tcpPositions[i];
            simSetGraphStreamValue(positionsGraphHandle, positionsGraphStreamIds[i], tcpPosition);
        }
```

Figura 6.8: Fragmento de código C++ del plugin implementando

- Escena: Se ha usado una escena creada tal y como se ha descrito en la subsección 5.1.2. La escena incluye un único brazo robot modelo UR5 el cual se ha alienado los ejes de coordenadas del link de la base con el origen de la escena para facilitar la posterior interpretación de los datos.

La validación en la simulación de CoppeliaSim es el paso final que asegura que todos los componentes del sistema funcionan en conjunto de manera adecuada.

6.2.3 Análisis de los Datos

En esta parte, se lleva a cabo un análisis exhaustivo de los datos recopilados con el propósito de evaluar la precisión y la consistencia del gemelo digital en comparación con el robot UR5 real. Los resultados se presentarán considerando diferentes puntos de adquisición de datos, y se examinarán posibles discrepancias significativas, además de buscar explicaciones subyacentes.

Datos del Robot UR5

Se inicia el análisis con los datos obtenidos directamente del robot UR5, que se utilizan como punto de referencia principal para evaluar el gemelo digital. Estos datos comprenden las coordenadas de posición del TCP adquiridas mediante la interfaz RTDE Receive del robot físico.

La Figura 6.9 ilustra las coordenadas de posición del TCP del robot UR5. Se observa que los valores se mantienen dentro de un rango aceptable durante la ejecución. Cualquier desviación significativa será objeto de un análisis más detenido para identificar posibles causas, como latencia en la comunicación o diferencias en la configuración.

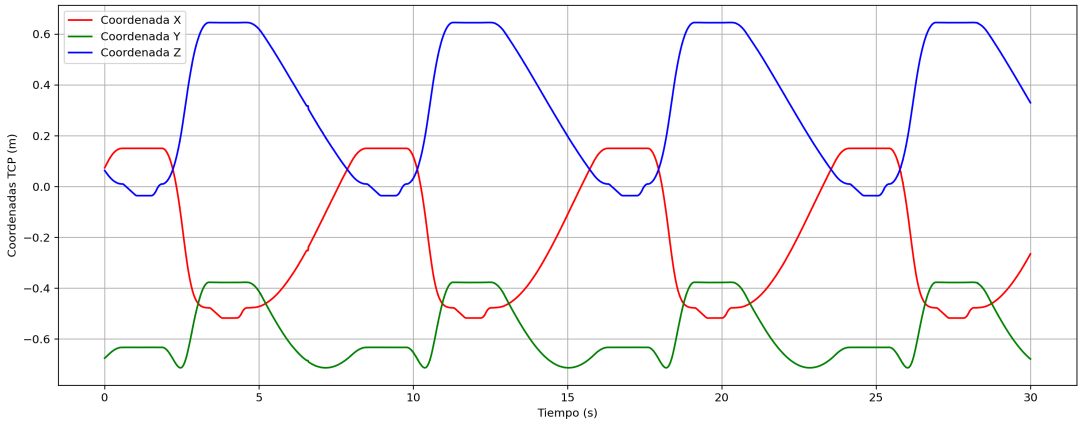


Figura 6.9: Gráfica temporal de las coordenadas de posición del TCP del robot UR5 extraídas con la interfaz RTDE.

Datos del Servidor OPC UA

A continuación, se procede al análisis de los datos almacenados en el servidor OPC UA. Estos datos deben reflejar con precisión las coordenadas de posición del TCP del robot UR5. Se realiza una comparación con las mediciones del robot real.

En la Figura 6.10, se observa que los datos del servidor OPC UA siguen una tendencia similar a las mediciones del robot UR5. Comparando las dos series de datos valor a valor, se ha observado que las mediciones son prácticamente idénticas. La única discrepancia entre esta serie de datos y

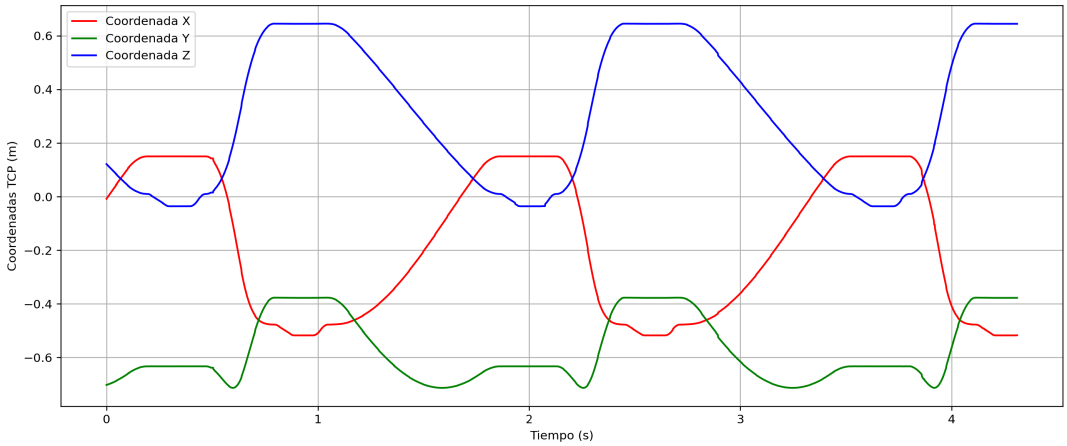


Figura 6.10: Gráfica temporal de las coordenadas de posición del TCP obtenidas del servidor OPC UA.

la recogida en el robot es causada por un ligero ruido de medida, probablemente introducido por los sensores o las etapas de transformación posteriores, no obstante, este ruido es despreciable comparado con el rango de las señales.

Datos del Cliente OPC UA (Prosys Monitor)

A continuación, se consideran los datos recolectados a través del cliente OPC UA, Prosys Monitor. Estos datos proporcionan una validación adicional de la comunicación y el almacenamiento de datos en el servidor. Se comparan las gráficas de Prosys Monitor con las mediciones del robot real.

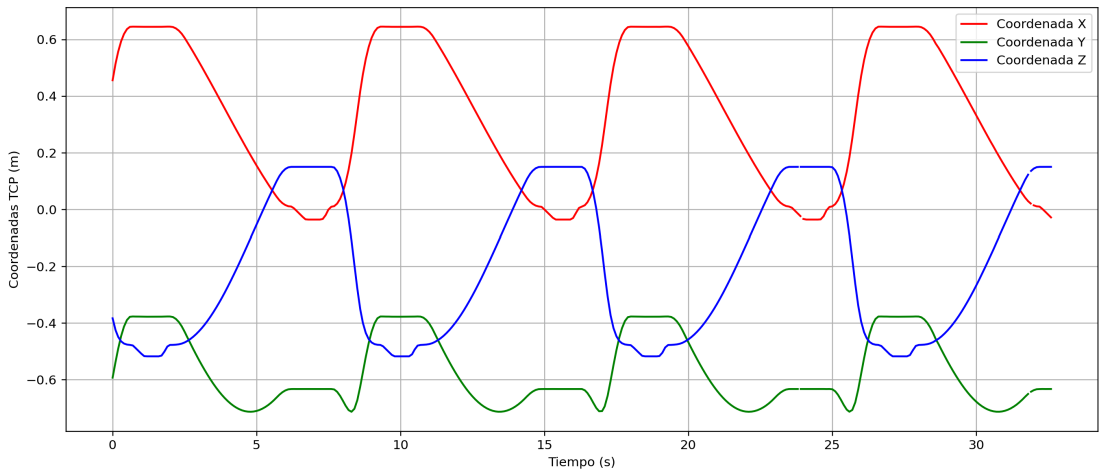


Figura 6.11: Gráfica temporal de las coordenadas de posición del TCP obtenidas con el cliente OPC UA (Prosys Monitor).

En la Figura 6.11, evidencia que los datos del cliente OPC UA siguen una tendencia similar a las mediciones del robot UR5 y los datos del servidor OPC UA. Esto valida aún más el correcto funcionamiento del servidor y la accesibilidad de los datos desde fuentes externas.

Datos de la Simulación en CoppeliaSim

Por último, se analizan los datos obtenidos de la simulación en CoppeliaSim. Estos datos permiten evaluar el rendimiento del gemelo digital en la replicación del comportamiento del robot UR5. Se compara la posición del TCP en la simulación con las mediciones del robot real.

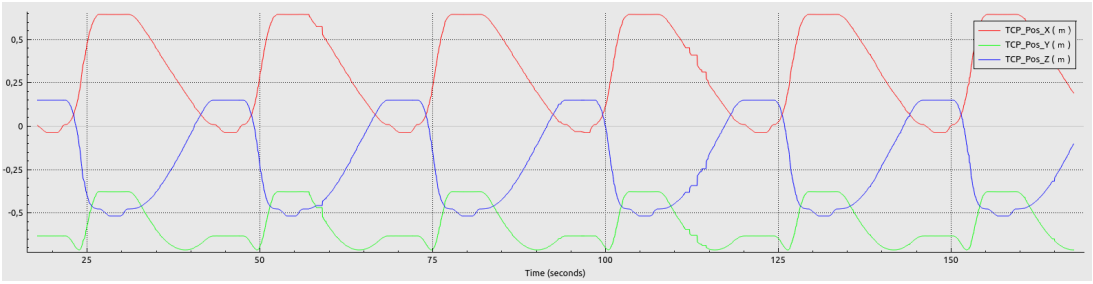


Figura 6.12: Gráfica temporal de las coordenadas de posición del TCP obtenidas de la simulación en CoppeliaSim.

La Figura 6.12 muestra que los datos de la simulación en CoppeliaSim siguen una tendencia similar a las mediciones del robot UR5 y los datos del servidor OPC UA. Este análisis verifica la capacidad del gemelo digital para replicar el comportamiento del robot real.

En todos los puntos de adquisición de datos, se buscan posibles discrepancias o desviaciones significativas y se realiza un análisis minucioso para identificar las razones subyacentes. Este proceso garantiza la precisión y la coherencia del gemelo digital en comparación con el robot UR5 real.

6.2.4 Conclusiones del Análisis

En general, estos patrones en los datos de las coordenadas TCP obtenidos en el apartado anterior, son coherentes con la tarea del robot de "pick and place". Estos coeficientes de correlación entre las coordenadas indican una relación muy fuerte entre ellas como era de esperar. La coordenada X y Z están altamente correlacionadas positivamente, mientras que X y Y, así como Y y Z, están altamente correlacionadas negativamente. Esto sumado a las tendencias temporales de los gráficos da una idea del movimiento del robot.

- La coordenada Z que muestra una tendencia descendente que está relacionada con el movimiento del robot desde una posición baja, donde recogería el objeto de la mesa tal y como muestra la Figura 6.4a, hacia una posición elevada, donde coloca el objeto en alguna estantería en la Figura 6.4b.
- Las coordenadas X y Y, que muestran un aumento gradual en sus valores, lo cual está relacionado con el movimiento lateral y longitudinal del robot mientras se desplaza desde la mesa hasta la estantería. Este movimiento entre las posiciones de pick y place, tal como se muestra en el gráfico tridimensional de la Figura 6.13 describe un arco ascendente.
- La alta correlación positiva entre las coordenadas X y Z indicar que el robot se mueve en una dirección diagonal desde la mesa hasta la estantería como se observa en la figura anterior.

Como cabía esperar, los datos recogidos en el robot UR5 y los recogidos en el servidor son prácticamente iguales. Esto se debe a que ambos datos han sido extraídos mediante la API de la RTDE Receive, que ha demostrado presentar una excelente repetitividad y flexibilidad a la hora de la extracción de los datos.

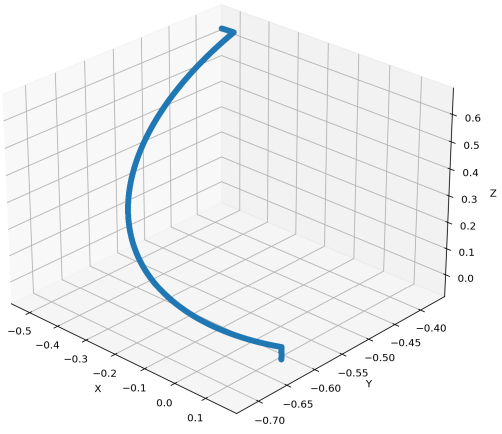


Figura 6.13: Gráfica 3D mostrando la trayectoria del TCP en metros del robot UR5 real.

Para comprender la utilidad de la comparación de estas dos series, es necesario recalcar que los datos recogidos en el servidor son los mismos que los enviados al servidor OPC UA mediante la función de alto nivel *UA_Server_writeDataValue*. Dicho esto, la comparación de estas dos series de datos permite determinar si el proceso de escritura de datos en el servidor, así como la creación de instancias de robots y nodos, se realizan correctamente y sin pérdida de información.

El hecho de que los datos del servidor sean coherentes con los datos del robot real es un indicador positivo de que la comunicación OPC UA funciona de manera efectiva. Esto implica que el programa del servidor es capaz de recibir y procesar los datos correctamente, y que la infraestructura de comunicación entre el robot y el servidor es sólida y confiable.

Al considerar los datos obtenidos del cliente OPC UA (Prosys Monitor), se observa una correspondencia cercana con las mediciones del robot real y los datos del servidor OPC UA. Esto refuerza aún más la confiabilidad del servidor y sugiere que los datos pueden ser accesibles desde fuentes externas, lo que es fundamental en una configuración de gemelo digital en la que se espera que los datos sean accesibles para aplicaciones y sistemas de control externos.

Finalmente, al analizar los datos de la simulación en CoppeliaSim, se observa que estos también siguen una tendencia similar a las mediciones del robot real y los datos del servidor OPC UA. Esto respalda la capacidad del gemelo digital para replicar con precisión el comportamiento del robot real en un entorno de simulación, lo que es esencial para pruebas y desarrollo de controladores sin la necesidad de acceso físico al robot.

En resumen, el análisis de datos demuestra la solidez y la precisión del gemelo digital desarrollado en este proyecto. La coincidencia de los datos en múltiples puntos de adquisición y su correspondencia con las mediciones del robot real validan la efectividad de la comunicación OPC UA y el funcionamiento del servidor. Esto es crucial en aplicaciones de robótica colaborativa en las que la precisión y la confiabilidad son fundamentales para garantizar un comportamiento seguro y coherente del robot.

6.3 Análisis Cualitativo en Diferentes Escenarios

En esta sección se presentan los resultados de las pruebas realizadas en varios escenarios para evaluar la aplicación del gemelo digital en situaciones diversas. Se llevaron a cabo dos tipos de pruebas, cada una en un entorno distinto, con el objetivo de abordar diferentes situaciones y validar la flexibilidad y robustez del gemelo digital. Además, en cada una de las pruebas se utilizaron robot reales y robots simulados en el entorno de URSim en un contenedor Docker, como el mostrado en la Figura 2.1.

6.3.1 Escenario Robot Individual

En este escenario, se simuló un único robot UR5 utilizando el gemelo digital en CoppeliaSim. La primera de las pruebas consistió en crear un robot UR5 simulado en el entorno URSim y conectarlo a la aplicación del gemelo digital. El objetivo principal era evaluar el funcionamiento del gemelo digital en un entorno controlado y seguro antes de realizar pruebas con robots físicos. Se comprobó que las variables obtenidas con la interfaz RTDE del robot simulado en el entorno URSim eran suficientemente buenas para simular el gemelo digital del robot mediante la aplicación. Durante estas pruebas, se supervisaron varios aspectos clave, como la precisión de la simulación, la capacidad de control y la correspondencia con el comportamiento esperado.

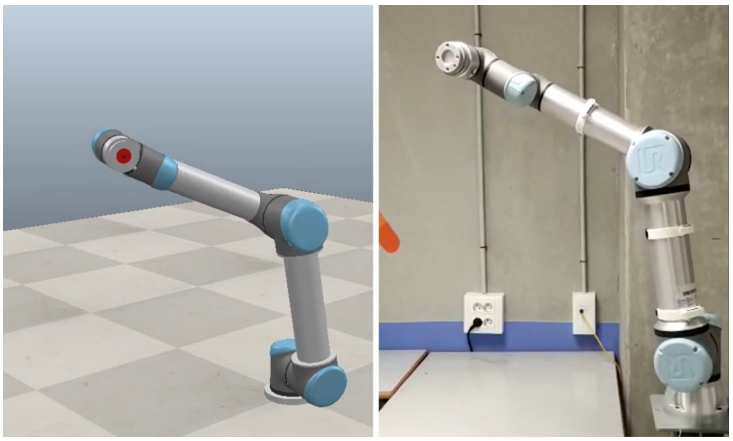


Figura 6.14: Robot UR5 real (derecha) y su gemelo digital simulado en CoppeliaSim (izquierda) moviéndose entre dos posiciones.

Estas pruebas permitieron confirmar que el gemelo digital podía replicar con precisión el comportamiento de un solo robot UR5. Tras ello se realizo la misma prueba utilizando el robot real modelo UR5 del laboratorio del Instituto de Automática y Robótica Industrial de la UPV. En la prueba se programó el robot para moverse en bucle entre dos posiciones como se puede apreciar el la Figura 6.14. Se observó una concordancia perfecta entre los datos simulados y las acciones del robot real. Esto sugiere que el gemelo digital es adecuado para tareas de modelado y simulación en entornos de desarrollo y prueba para un sólo robot.

6.3.2 Escenario con Múltiples Robots

En este escenario, se iniciaron las pruebas utilizando dos robots UR3 simulados, cada uno en su propio entorno URSim en Docker como muestra la Figura 6.15.

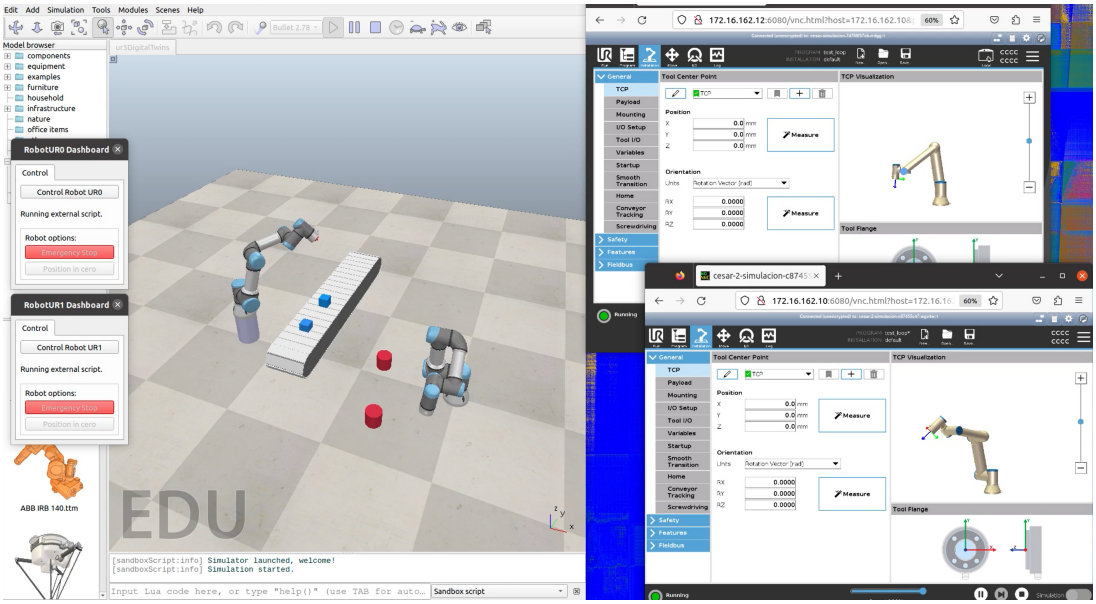


Figura 6.15: Dos robot UR3 simulados (derecha) y sus gemelos digitales en CoppeliaSim (izquierda) moviéndose en una trayectoria.

Tras comprobar que los dos gemelos de los robots replicaban el movimiento de los robots simulados con fidelidad, se replicaron estas pruebas en un entorno controlado con dos robots UR3 reales facilitados por el Dpto. de Ingeniería de Sistemas y Automática (DISA) de la UPV. Para el ensayo se montó realizaron las conexiones mostradas en la Figura 6.16.

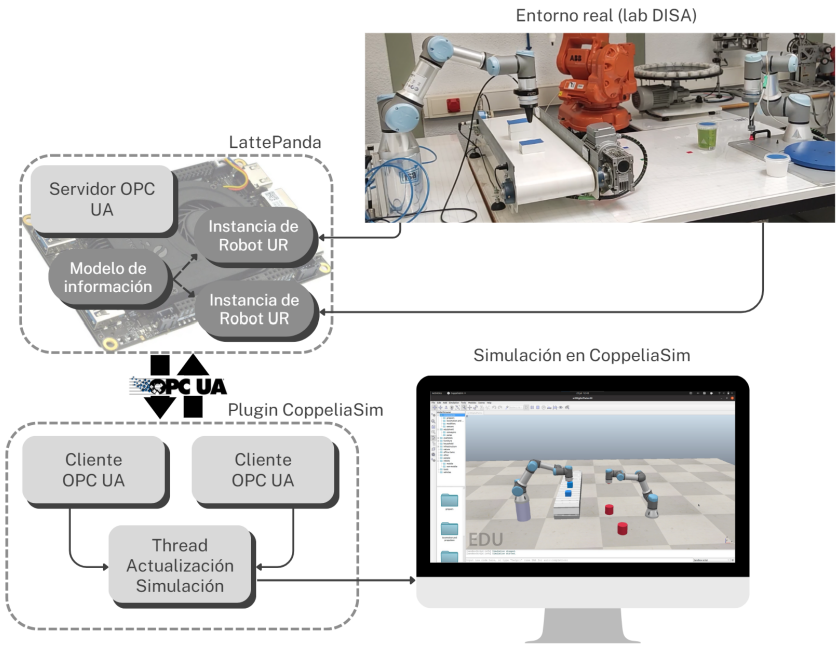
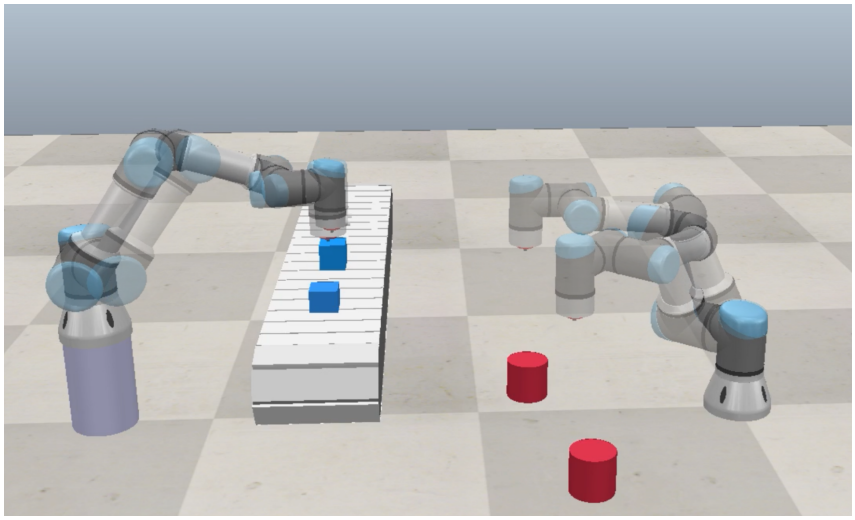
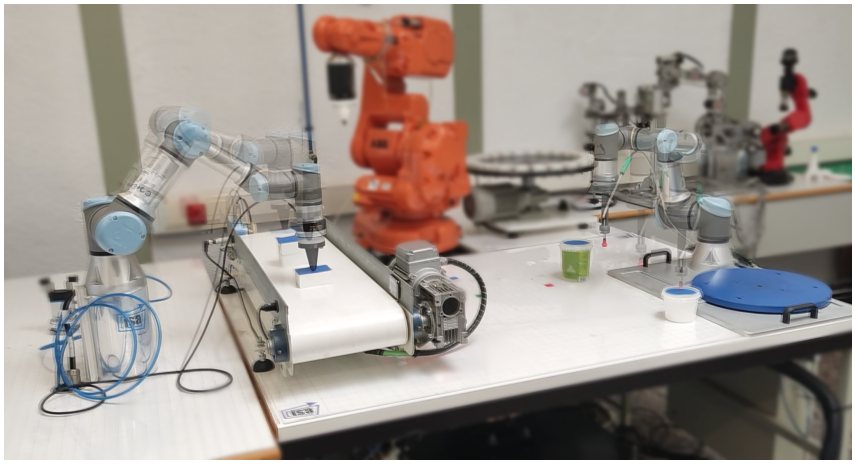


Figura 6.16: Diagrama de los elementos y conexiones de la prueba con dos robots UR3.

Uno de los robots UR3 reales se colocó frente a una cinta transportadora sin movimiento con dos objetos sobre ella y se programó para moverse de un objeto a otro. El segundo robot UR3 real se situó en el suelo con dos objetos cercanos y se programó moviéndose entre ellos. En las se muestra el desarrollo de esta prueba. Estas pruebas avanzadas destacaron la capacidad del gemelo digital para modelar y controlar múltiples robots en un entorno colaborativo y respondiendo eficazmente a situaciones dinámicas.



(a) Gemelos digitales en CoppeliaSim.



(b) Entorno con los robots reales.

Figura 6.17: Prueba de operación con varios robots UR3 interaccionando con el entorno en el DISA

6.3.3 Conclusiones del Análisis

El análisis cualitativo en diferentes escenarios demuestra que el gemelo digital es altamente versátil y preciso. En el escenario de un robot individual, se confirma que el gemelo digital replica con éxito el comportamiento del robot real en un entorno simulado y real, realizando tareas de pick and place. En el escenario con múltiples robots, el gemelo digital demuestra su capacidad para gestionar la interacción entre varios robots en un entorno colaborativo y responder eficazmente a situaciones dinámicas.

Estos resultados respaldan la utilidad del gemelo digital en una variedad de aplicaciones, desde el desarrollo y prueba de algoritmos de control hasta la simulación de escenarios complejos de robótica colaborativa. Además, establecen una base sólida para futuras investigaciones y aplicaciones en las que se requiera una representación precisa y versátil del comportamiento de robots en entornos virtuales y reales.

Capítulo 7

Conclusión

En un mundo en constante evolución, donde la automatización y la colaboración humano-robot están revolucionando la industria, el desarrollo de gemelos digitales se ha vuelto esencial para abordar los desafíos de la Industria 4.0 y más allá. Esta tesis se ha centrado en el diseño, desarrollo y simulación de un gemelo digital funcional para brazos colaborativos Universal Robots, aprovechando las capacidades de las comunicaciones OPC UA y la plataforma de simulación CoppeliaSim. Durante el desarrollo de esta investigación, se han explorado aspectos técnicos y de investigación y desarrollo de la creación de gemelos digitales con el enfoque particular en la monitorización y el mantenimiento predictivo.

Uno de los aspectos que cabe destacar de esta tesis es la convergencia de múltiples disciplinas. En el proyecto, se han integrado la ingeniería automática y la ingeniería de sistemas para diseñar un gemelo digital preciso que imite de manera efectiva el comportamiento del brazo colaborativo físico. Las comunicaciones e informática industriales han desempeñado un papel fundamental en la implementación del protocolo OPC UA, permitiendo la transferencia eficiente de datos entre el sistema físico y su contraparte virtual. Además, se ha aprovechado la potencia de la simulación robótica con CoppeliaSim para visualizar y analizar el gemelo digital en acción. Todo ello, soportado por una base sólida en la planificación y gestión de proyectos.

No obstante, este trabajo no solo se ha centrado en la implementación técnica, sino que también ha con llevado un esfuerzo considerable de investigación y desarrollo (I+D). Se han investigado y evaluado a fondo las tecnologías y enfoques disponibles para la creación de gemelos digitales de brazos colaborativos, intentando llevar la contribución más allá de la simple implementación, ya que se han identificado y abordado desafíos clave relacionados con la precisión, sincronización, convergencia y adaptabilidad del gemelo digital.

7.1 Objetivos Logrados

A lo largo de este estudio, se han alcanzado una serie de objetivos clave que tienen un impacto significativo en la monitorización y el mantenimiento predictivo de los robots UR, así como en la creación de soluciones flexibles y adaptables para la industria:

- Comportamiento Preciso: Uno de los logros más notables de este trabajo es la creación de un gemelo digital funcional capaz de interactuar de manera precisa con sensores y dispositivos físicos. Esta capacidad permite una comunicación fluida para visualizar y registrar datos relevantes, lo que resulta fundamental para el mantenimiento predictivo y la toma de decisiones informadas en la industria. La precisión en la replicación del comportamiento del brazo colaborativo físico en el entorno virtual resulta esencial para garantizar la confiabilidad de las decisiones basadas en los datos recogidos.
- Sincronización Eficiente: Mantener la sincronización entre el gemelo digital y su contraparte física es esencial para una monitorización efectiva del robot. En la aplicación, se ha garantizado que los datos fluyan de manera constante desde el sistema físico hacia el gemelo digital. Esta sincronización eficiente facilita la toma de decisiones y permite una respuesta ágil ante cambios en las condiciones del robot.
- Convergencia: La convergencia continua entre los espacios físicos y digitales es un proceso que se ha establecido y perfeccionado mediante sucesivas pruebas e iteraciones. Identificar y abordar las diferencias entre el comportamiento del brazo colaborativo físico y su representación digital ha sido crucial para mantener la precisión del gemelo digital en todo momento. Esta característica es especialmente valiosa en situaciones donde la contraparte física puede experimentar cambios o desgaste con el tiempo.
- Verificación y Validación Rigurosas: La implementación de mecanismos de verificación y validación ha sido esencial para garantizar que el gemelo digital funcione correctamente. Estas pruebas rigurosas han confirmado la confiabilidad y eficacia del enfoque desarrollado. Esto proporciona tranquilidad a los usuarios y garantiza que los datos recopilados sean precisos y confiables.
- Interoperabilidad y Flexibilidad: La arquitectura independiente de la plataforma ha demostrado ser altamente interoperable y adaptable a diferentes sistemas operativos y lenguajes de programación. Esto proporciona a la industria una solución flexible y de fácil implementación. Además, la capacidad de trabajar en diversos entornos y con una variedad de tecnologías hace que el gemelo digital sea una herramienta versátil que puede integrarse con facilidad en una amplia gama de aplicaciones y configuraciones.
- Deslocalización del Sistema: Facilitar la comunicación entre diferentes gemelos digitales, incluso cuando se encuentran en ubicaciones geográficas distintas, es una característica a destacar del proyecto. Esto amplía las posibilidades de colaboración y comparación de datos en un mundo cada vez más globalizado. La deslocalización del sistema permite a las organizaciones trabajar de manera eficiente y colaborativa en proyectos que involucran gemelos digitales, sin verse limitadas por la distancia física.
- Modificabilidad y Reusabilidad: El gemelo digital se ha diseñado para ser altamente modificable y reutilizable, esto permite adaptarlo a varios tipos de robots UR y la incorporación

de nuevos sensores o actuadores. La capacidad de modificar y reutilizar componentes del gemelo digital es esencial en un entorno donde la tecnología y las necesidades pueden evolucionar con rapidez. Este enfoque permite a las organizaciones mantenerse ágiles y flexibles en un entorno industrial en constante cambio.

7.2 Líneas de mejora

En el transcurso de esta tesis, se han logrado avances notables en la creación de un gemelo digital funcional para brazos colaborativos Universal Robots (UR). Sin embargo, el camino hacia la perfección es un proceso en constante evolución, y hay varias áreas identificadas que presentan oportunidades significativas para mejoras futuras.

Una de las áreas críticas que merece una atención continua es la optimización de recursos y del código subyacente. Si bien se ha logrado una comunicación efectiva entre el gemelo digital y el sistema físico, existe margen para optimizar aún más el rendimiento de la aplicación. Esto puede incluir correcciones y refinamientos en el código fuente, particularmente en lo que respecta a la eficiencia y la velocidad de ejecución. Dado que no todos los usuarios son necesariamente expertos programadores, es importante explorar la implementación de algoritmos y estructuras de datos más eficientes que puedan mejorar la capacidad de cálculo y reducir la carga en los recursos del sistema. Este es un aspecto esencial para garantizar que el gemelo digital pueda funcionar de manera óptima incluso en entornos con recursos limitados.

En cuanto a la adaptabilidad y la interacción con diferentes escenarios, el gemelo digital ha demostrado ser altamente adaptable a una variedad de brazos colaborativos UR. Sin embargo, el enfoque futuro debe centrarse en mejorar aún más la interacción del gemelo digital con su entorno. Esto puede implicar el desarrollo de interfaces de usuario más intuitivas y herramientas de configuración flexibles que permitan a los usuarios personalizar la aplicación para satisfacer sus necesidades específicas. Además, se puede explorar la posibilidad de integrar el gemelo digital con otros sistemas y dispositivos en una línea de producción, lo que ampliaría su utilidad y su capacidad para interactuar de manera más efectiva en entornos industriales complejos.

Un aspecto crítico que se puede mejorar en el gemelo digital es la seguridad y la ciberseguridad. Aunque el gemelo digital ha demostrado ser una herramienta valiosa para la monitorización y el mantenimiento predictivo, la seguridad de las comunicaciones y los datos es un área que requiere una atención continua. Actualmente, la implementación de medidas de seguridad en el protocolo OPC UA no se ha abordado en profundidad en esta tesis, pero es esencial para garantizar la integridad de los datos y proteger la aplicación de posibles amenazas cibernéticas. Las futuras investigaciones pueden enfocarse en fortalecer la seguridad del gemelo digital, implementando protocolos de autenticación robustos y cifrado de datos para salvaguardar la información crítica.

Finalmente, la validación empírica en entornos industriales del mundo real es un paso fundamental para demostrar la eficacia y la utilidad del gemelo digital. A pesar de los resultados prometedores obtenidos en pruebas de laboratorio, es esencial llevar a cabo estudios de caso y pruebas piloto en colaboración con empresas industriales. Esto permitirá validar el rendimiento del gemelo digital en situaciones prácticas y asegurarse de que cumple con los requisitos del mundo real. Además, estas validaciones pueden proporcionar valiosos comentarios que alimenten futuras mejoras y refinamientos en la aplicación.

7.3 Reflexión Final

Al llegar al cierre de este proyecto de investigación y presentar la aplicación del Gemelo digital para brazos UR, es apropiado realizar una reflexión sobre el trayecto llevado a cabo. Este proyecto ha sido un viaje de aprendizaje constante, lleno de retos y descubrimientos que merecen ser destacados.

La rápida evolución de la tecnología y su influencia en la industria son evidentes en cada etapa de esta tesis. Desde los primeros días del proyecto hasta este momento, se ha observado cómo la colaboración entre humanos y robots se ha convertido en un componente esencial de la fabricación moderna.

La creación de este gemelo digital no solo representa un logro técnico, sino también una respuesta a la necesidad de adaptarse y abrazar la innovación en un mundo en constante cambio. La Industria 4.0 y la Industria 5.0 desafían la forma en que se conciben y operan los sistemas de fabricación. La flexibilidad y la eficiencia son clave en este nuevo paradigma, y los gemelos digitales ofrecen una prometedora vía para lograrlo.

Este proyecto subraya que la tecnología es una herramienta poderosa que puede mejorar nuestra forma de trabajar y vivir. También destaca la importancia de la perseverancia y la pasión en la búsqueda de soluciones innovadoras. A medida que se avanza hacia el futuro, se espera que el trabajo realizado aquí siente una sólida base para investigaciones posteriores y desarrollos en el campo de los gemelos digitales y la robótica colaborativa.

En última instancia, esta investigación inspira a seguir explorando los límites de lo que es posible en la intersección de la tecnología y la industria. Los gemelos digitales son solo el comienzo, y se anticipa que las oportunidades futuras en este campo seguirán impulsando la innovación y darán forma a un mundo más eficiente y colaborativo.

Parte II

Presupuesto

Presupuesto general del proyecto

Resumen del presupuesto

Tabla 7.1: Resumen del presupuesto general de ejecución

Recurso	Tipo	Tipo de presupuesto	Trabajo	Costo	Tasa estándar
Presupuesto Mano de obra			**738 h**	**16,395.24 €**	
Ingeniero	Trabajo	Mano de obra	738 h	16,395.24 €	22.19 €/h
Presupuesto Equipamiento			**1542 h**	**1,584.25 €**	
Portátil Lenovo	Material	Equipamiento	738 h	338.25 €	55 €/mes
LattePanda	Material	Equipamiento	348 h	87.00 €	30 €/mes
Robot UR5	Material	Equipamiento	162 h	742.50 €	550 €/mes
Clúster ai2	Material	Equipamiento	294 h	416.50 €	170 €/mes
Presupuesto Total			**2280 h**	**17,979.49 €**	

Presupuesto de ejecución detallado

A continuación se presenta el presupuestos de ejecución detallado y los detalles de los recursos incluidos en el presupuesto.

Tarea	Trabajo	Duración	Costo
1. Investigación y Planificación	**84.86h**	**10.5d**	**1,921.48 €**
Documentación RTDE	**18.86h**	**3d**	**426.69 €**
Ingeniero	18.86h		418.44 €
Portátil Lenovo	18.86h		8.25 €

Formación sobre OPC UA	**54h**	**8d**	**1,223.01 €**
Ingeniero	54h		1,198.26 €
Portátil Lenovo	54h		24.75 €
Fase 2 Planificación	**12h**	**2d**	**271.78 €**
Ingeniero	12h		266.28 €
Portátil Lenovo	12h		5.50 €
2. Programa RTDE	**78h**	**11.83d**	**1,979.07 €**
Desarrollo del programa	**42h**	**7d**	**1,010.73 €**
Ingeniero	42h		931.98 €
Portátil Lenovo	42h		19.25 €
Clúster ai2	42h		59.50 €
Prueba del programa	**24h**	**4d**	**693.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
LattePanda	24h		6.00 €
Robot UR5	24h		110.00 €
Clúster ai2	24h		34.00 €
Fase 3 Planificación	**12h**	**2d**	**274.78 €**
Ingeniero	12h		266.28 €
Portátil Lenovo	12h		5.50 €
LattePanda	12h		3.00 €
3. Comunicación OPC UA	**300h**	**36.5d**	**7,370.00 €**
Instalar herramientas OPC UA	**36h**	**6d**	**824.34 €**
Ingeniero	36h		798.84 €
Portátil Lenovo	36h		16.50 €
LattePanda	36h		9.00 €
Crear Modelo de información	**54h**	**9d**	**1,236.51 €**
Ingeniero	54h		1,198.26 €
Portátil Lenovo	54h		24.75 €
LattePanda	54h		13.50 €
Compilación Modelo de información	**24h**	**4d**	**549.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
LattePanda	24h		6.00 €
Planificación programa servidor	**12h**	**2d**	**274.78 €**
Ingeniero	12h		266.28 €

Portátil Lenovo	12h		5.50 €
LattePanda	12h		3.00 €
Desarrollo thread del servidor	**66h**	**11d**	**1,511.29 €**
Ingeniero	66h		1,464.54 €
Portátil Lenovo	66h		30.25 €
LattePanda	66h		16.50 €
Incluir thread del robot	**24h**	**4d**	**583.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
LattePanda	24h		6.00 €
Clúster ai2	24h		34.00 €
Pruebas comunicación robot-servidor	**24h**	**4d**	**693.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
LattePanda	24h		6.00 €
Robot UR5	24h		110.00 €
Clúster ai2	24h		34.00 €
Extensión para varios robots	**30h**	**5d**	**866.95 €**
Ingeniero	30h		665.70 €
Portátil Lenovo	30h		13.75 €
LattePanda	30h		7.50 €
Robot UR5	30h		137.50 €
Clúster ai2	30h		42.50 €
Optimización del código	**24h**	**4d**	**693.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
LattePanda	24h		6.00 €
Robot UR5	24h		110.00 €
Clúster ai2	24h		34.00 €
Fase 4 Planificación	**6h**	**1d**	**135.89 €**
Ingeniero	6h		133.14 €
Portátil Lenovo	6h		2.75 €
4. Simulación en CoppeliaSim	**192h**	**22,67d**	**4,600.48 €**
Creación de escenas y modelos	**12h**	**2d**	**271.78 €**
Ingeniero	12h		266.28 €
Portátil Lenovo	12h		5.50 €
Planificación del programa del plugin	**12h**	**2d**	**271.78 €**

Ingeniero	12h		266.28 €
Portátil Lenovo	12h		5.50 €
Implementación Cliente OPC UA	**42h**	**7d**	**951.23 €**
Ingeniero	42h		931.98 €
Portátil Lenovo	42h		19.25 €
Hilo Actualización de la simulación	**60h**	**10d**	**1,358.90 €**
Ingeniero	60h		1,331.40 €
Portátil Lenovo	60h		27.50 €
Interfaz Gráfica de Usuario	**24h**	**4d**	**543.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
Pruebas de Funcionamiento	**24h**	**4d**	**687.56 €**
Ingeniero	24h		532.56 €
Portátil Lenovo	24h		11.00 €
Robot UR5	24h		110.00 €
Clúster ai2	24h		34.00 €
Optimización del código	**18h**	**3d**	**515.67 €**
Ingeniero	18h		498.67 €
Portátil Lenovo	18h		17.00 €
Documentación	**30h**	**5d**	**683.95 €**
Ingeniero	30h		665.70 €
Portátil Lenovo	30h		13.75 €
LattePanda	30h		3.00 €
Robot UR5	30h		137.50 €
Clúster ai2	30h		42.50 €
5. Tests y Análisis	**84h**	**14d**	**7,370.00 €**
Depuración del código	**18h**	**3d**	**520.17 €**
Ingeniero	18h		399.42 €
Portátil Lenovo	18h		8.25 €
LattePanda	18h		4.50 €
Robot UR5	18h		82.50 €
Clúster ai2	18h		25.50 €
Tests Fallos de Memoria	**30h**	**5d**	**721.95 €**
Ingeniero	30h		665.70 €
Portátil Lenovo	30h		13.75 €
Clúster ai2	30h		42.50 €
Pruebas un solo robot	**24h**	**4d**	**577.56 €**
Ingeniero	24h		532.56 €

Portátil Lenovo	24h		11.00 €
Clúster ai2	24h		34.00 €
Pruebas multirobot	**12h**	**2d**	**288.78 €**
Ingeniero	12h		266.28 €
Portátil Lenovo	12h		5.50 €
Clúster ai2	12h		17.00 €

Detalle de los recursos

- Renting Portátil Lenovo

  Modelo: Lenovo ThinkPad E15 RAM 16 GB SDD 256 GB

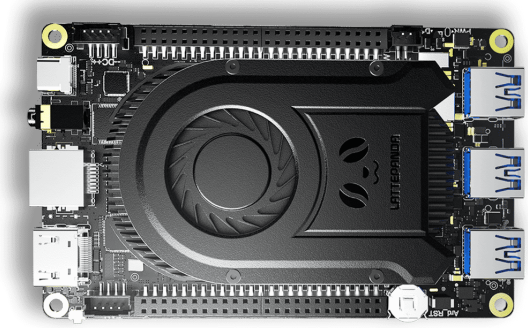
  Precio medio: 55€al mes

Portátil Lenovo ThinkPad E15. Fuente: Lenovo.

- Renting Sistema Empotrado LattePanda

  Modelo: LattePanda Delta V3 864

  Precio medio: 30€al mes



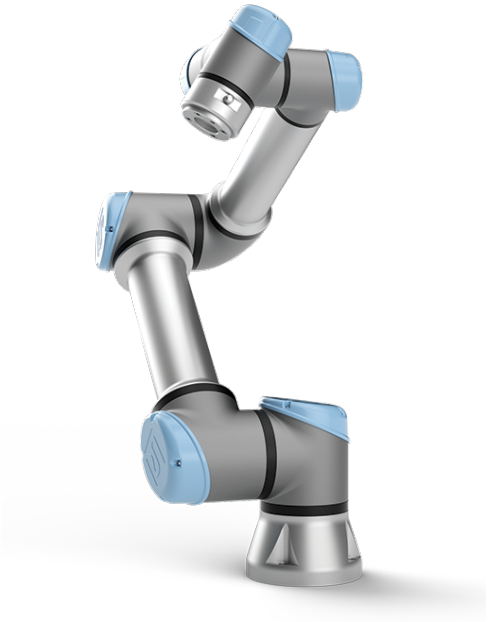
Sistema empotrado LattePanda Delta V3. Fuente: LattePanda.

- Renting Robot UR5

Modelo: UR5e

Ubicación: Instituto de Automática e Informática Industrial

Precio medio: 550€al mes



Robot colaborativo UR5e. Fuente: Universal Robots.

- Uso Clúster ai2

  Modelo: Clúster de cómputo de 20 computadores

  Ubicación: Instituto de Automática e Informática Industrial

  Precio medio: 170€al mes

Clúster de cómputo del instituto universitario ai2. Fuente: Instituto ai2.

Parte III

Apéndices

Apéndice A

Modelo de Información

A.1 Estructura del modelo de información

En esta sección se desarrollará la estructura en detalle del modelo de información creado para la construcción de los gemelos digitales de los brazos colaborativos de Universal Robots. En la primera sección se mostrará un diagrama completo del modelo de información para posteriormente poder hacer referencia a las partes más importantes de este con mayor detalle. Es por ello que en el diagrama completo de la Figura A.3 se han omitido algunos nodos de menor importancia.

A.1.1 Vista general del modelo de información

Antes de comenzar presentando los diagramas del modelo de información se presentan en la Figura A.1 la leyenda de los objetos utilizados en los diagramas de modelos de información de OPC UA. Estos objetos son formas estándar establecidas para todos los modelos de información o companion specification de OPC UA.

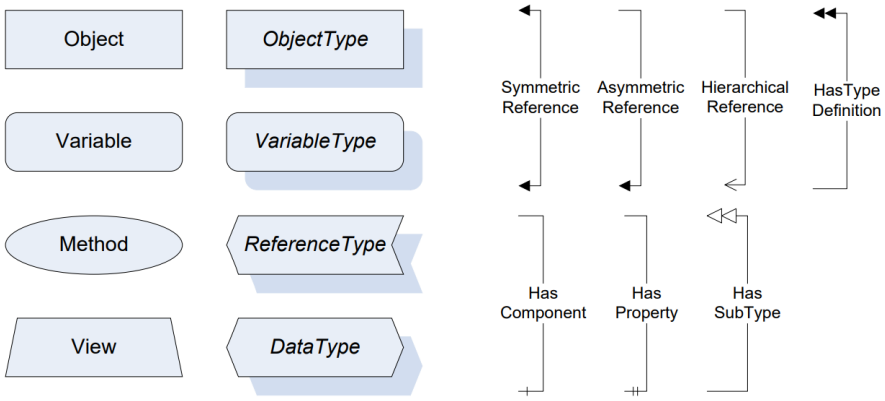


Figura A.1: Símbolos estandarizados para representar los modelos de información de OPC UA.

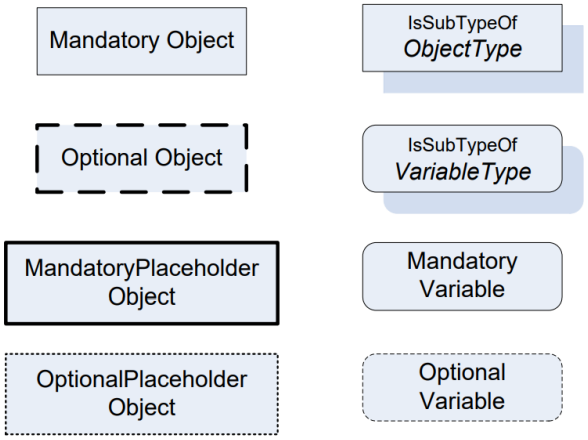


Figura A.2: Símbolos adicionales utilizados en los diagrama de bloques OPC UA.

Figura A.3: Diagrama de bloques OPC UA estandarizado del modelo de información personalizado del gemelo digital.

A.1.2 Principales componentes del modelo

El modelo de información contiene la creación de los tipos denominados *SensorType* y *RobotType*. El tipo de sensor mostrado en el diagrama Figura A.4 se usa para crear las instancias de los sensores de fuerza y par del brazo colaborativo.

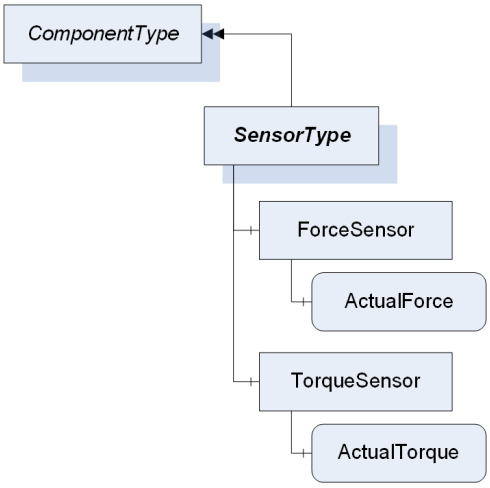


Figura A.4: Diagrama de bloques OPC UA estandarizado del tipo personalizado SensorType.

Por otra parte, en el objeto *RobotType* se compone de los siguientes nodos: MotionDevices, SafetyStates, Controllers. A su vez, MotionDevices da lugar al nodo raíz de las instancias de robots, RobotArm, cuyas propiedades e hijos se presentan en la Figura A.5.

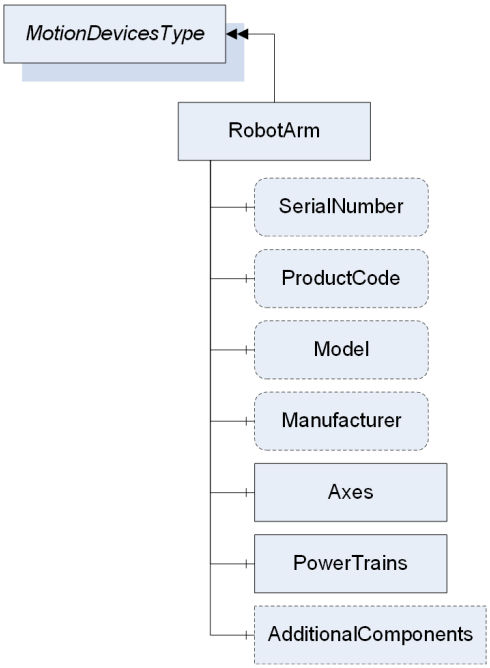


Figura A.5: Diagrama de bloques OPC UA tipo MotionDevices cuyo hijo es RobotArm.

El nodo RobotArm contiene los nodos que representan los ejes del robot cuya estructura se muestra en la Figura A.6, y los sistemas de potencia (Power Trains) cuyo esquema en la Figura A.7 incluye el motor y opcionalmente, las cajas de reducción de velocidad (gears).

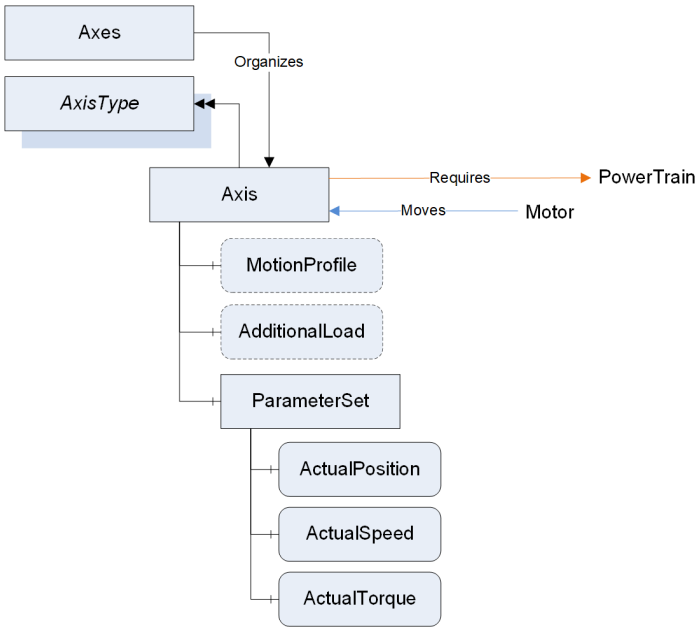


Figura A.6: Diagrama de bloques OPC UA del modelo de los nodos de los ejes (Axis) de las instancias RobotArm.

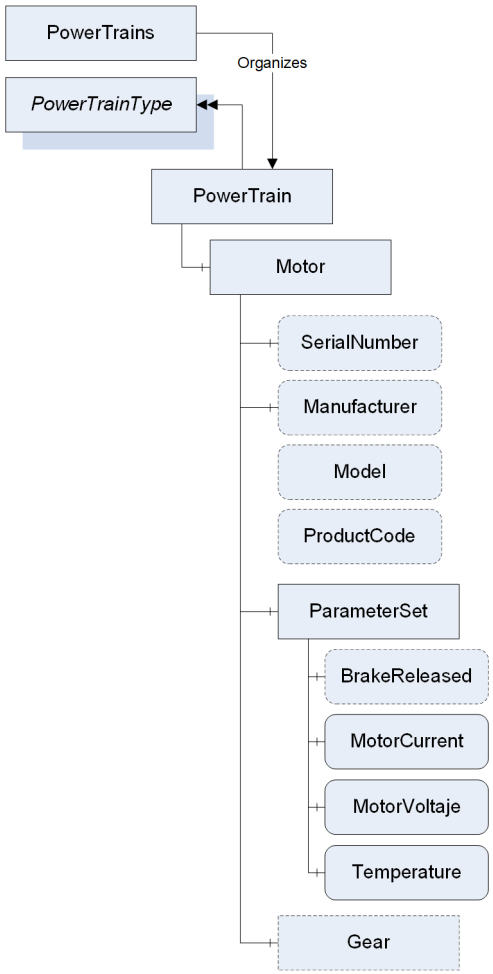


Figura A.7: Diagrama de bloques OPC UA del modelo de los nodos de los sistemas de potencia (PowerTains) de las instancias RobotArm.

Por otro lado, se incorporan del Robotics Companion Specification los nodos de SafetyStates (Figura A.8) y Controllers (Figura A.9). El nodo SafetyStates continene los estados de parada del robot cuyo funcionamiento se ha explicado en la subsección 4.4.3.

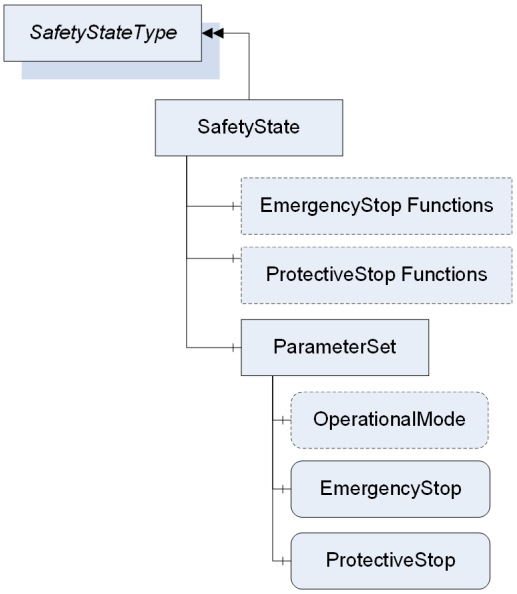


Figura A.8: Diagrama de bloques OPC UA de los nodos de los estados y funciones del robot.

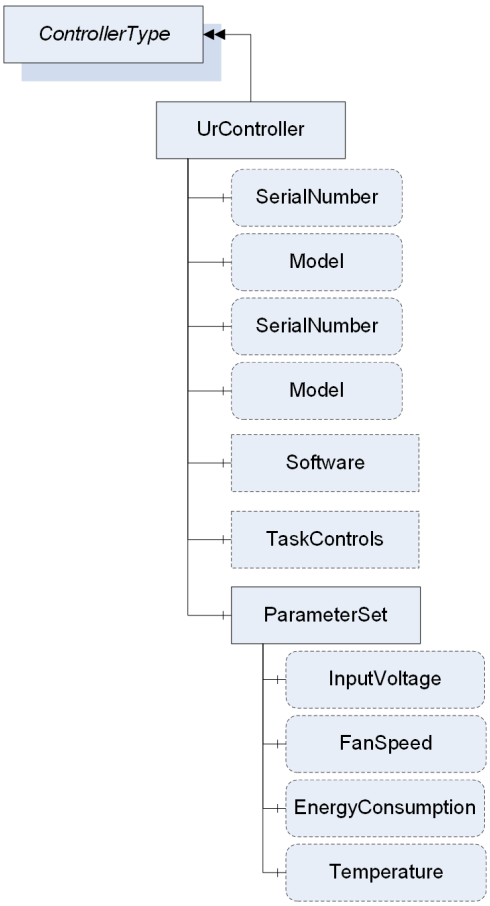


Figura A.9: Diagrama de bloques OPC UA de los nodos del controlador del robot.

A.2 Código del modelo de información

En un esfuerzo por dar claridad a la implementación del modelo de información, en adición a las capturas de código presentadas en la Figura 4.5 y la Figura 4.6 de la sección 4.3, en esta sección se adjuntan las secciones que se han considerado relevantes del código del modelo de información.

En la Figura A.10 se presenta la creación del nodo correspondiente al eje de la base del robot con sus correspondientes variables. En esta imagen se observa también la creación de la referencia unidireccional del tipo *Requires* del eje al Sistema de potencia.

```
<!-- AxisType Objects -->
<Object SymbolicName="ROB:Axis1" TypeDefinition="ROB:AxisType" ModellingRule="Mandatory">
    <BrowseName>Axis J1</BrowseName>
    <Description>Base Axis</Description>
    <Children>
        <Object SymbolicName="DI:ParameterSet" ModellingRule="Mandatory">
            <Children>
                <Variable SymbolicName="ROB:ActualPosition" TypeDefinition="OpcUa:AnalogUnitType" DataType="OpcUa:Double" ModellingRule="Mandatory">
                    <Description>Position of the corresponding joint in radians.</Description>
                     <DefaultValue>
                        <Value>0.0</Value>
                        <EngineeringUnits>
                            <DisplayName>radians</DisplayName>
                            <NamespaceUri>http://opcfoundation.org/UA/units/un/cefact</NamespaceUri>
                            <UnitId>256</UnitId>
                        </EngineeringUnits>
                    </DefaultValue>
                </Variable>
                <Variable SymbolicName="ROB:ActualSpeed" TypeDefinition="OpcUa:AnalogUni" DataType="OpcUa:Double" ModellingRule="Mandatory">...</Variable>
                <Variable SymbolicName="ROB:ActualTorqu" TypeDefinition="OpcUa:AnalogUni" DataType="OpcUa:Double" ModellingRule="Mandatory">...</Variable>
            </Children>
            <References>
                <Reference IsInverse="false">
                    <ReferenceType>ROB:Requires</ReferenceType>
                    <TargetId>ROB:PowerTrain1</TargetId>
                </Reference>
            </References>
        </Object>
    </Children>
</Object>
```

Figura A.10: Sección del código xml de la implementación del primer eje del robot en el modelo de información.

En la siguiente sección del código, mostrada en la Figura A.11, se muestra el uso de los tipos de sensores *ForceSensor* y *TorqueSensor* creados en el propio modelo de información.

```
<Object SymbolicName="ROB_UR:TcpGeneralizedForces" ModellingRule="Mandatory">
    <Children>
        <Object SymbolicName="ROB_UR:TcpForceSensor" TypeDefinition="ROB_UR:ForceSensor" ModellingRule="Mandatory">
            <Description> Force experienced by the tcp of the robot</Description>
            <Children>
            <Variable SymbolicName="ROB_UR:ActualForce" TypeDefinition="OpcUa:AnalogItemType" DataType="OpcUa:Double"
 ModellingRule="Mandatory" ValueRank="Array" AccessLevel="ReadWrite">
                <Description>Current forces in X Y and Z in N in an array</Description>
            </Variable>
            </Children>
        </Object>
        <Object SymbolicName="ROB_UR:TcpTorqueSensor" TypeDefinition="ROB_UR:TorqueSensor" ModellingRule="Mandatory">
            <Description> Torque experienced by the tcp of the robot</Description>
            <Children>
            <Variable SymbolicName="ROB_UR:ActualTorque" TypeDefinition="OpcUa:AnalogItemType" DataType="OpcUa:Double"
 ModellingRule="Mandatory" ValueRank="Array" AccessLevel="ReadWrite">
                <Description>Current Torques in X Y and Z in Nm in an array</Description>
            </Variable>
            </Children>
        </Object>
    </Children>
</Object>
```

Figura A.11: Sección del código xml de la implementación de los sensores de fuerza del robot en el modelo de información.

Por último, en la Figura A.12 se presenta la creación de los nodos *EmergencyStop* y *ProtectiveStop* que soportan la escritura y lectura de sus atributos.

```
<Object SymbolicName="ROB:SafetyStates" TypeDefinition="OpcUa:BaseObjectType" ModellingRule="Mandatory">
    <Description>Contains Safety states.</Description>
    <Children>
        <Object SymbolicName="ROB_UR:RobotSafetyStates" TypeDefinition="ROB:SafetyStateType" ModellingRule="Mandatory">
            <Description>Robot safety states.</Description>
            <Children>
                <Object SymbolicName="DI:ParameterSet" ModellingRule="Mandatory">
                    <Children>
                        <Variable SymbolicName="ROB:EmergencyStop" TypeDefinition="OpcUa:BaseDataVariableType" DataType="OpcUa:Boolean"
                          ModellingRule="Mandatory" AccessLevel="ReadWrite">
                            <Description>The EmergencyStop variable. </Description>
                        </Variable>
                        <Variable SymbolicName="ROB:ProtectiveStop" TypeDefinition="OpcUa:BaseDataVariableType" DataType="OpcUa:Boolean"
                          ModellingRule="Mandatory" AccessLevel="ReadWrite">
                            <Description>The ProtectiveStop variable. </Description>
                        </Variable>
                    </Children>
                </Object>
            </Children>
        </Object>
    </Children>
</Object>
```

Figura A.12: Sección del código xml de la implementación de las funciones de parada del robot en el modelo de información.

Apéndice B

Planificación del proyecto

B.1 Planificación de tareas del proyecto

Para desarrollar con éxito el proyecto se ha realizado una planificación temporal de las tareas a desarrollar. La para desarrollar la planificación y el seguimiento de las tareas se ha utilizado el software de dirección de proyectos de la suite de Microsoft, *MS Project*. En la Tabla B.1 se presentan los principales hitos del proyecto. Cabe destacar que la planificación, no ha sufrido cambios considerables durante el desarrollo del proyecto. En la Tabla B.2 se puede observar la planificación detallada del proyecto.

Tabla B.1: Hitos del proyecto

Hito	Fecha conseguido
Fase 1	
Fase 1 Completada	vie 31/03/23
Fase 2	
Programa RTDE	mar 11/04/23
Fase 2 Completada	mié 19/04/23
Fase 3	
Modelo de información	mar 10/05/23
Servidor OPC UA	mié 31/05/23
Fase 3 Completada	vie 16/06/23
Fase 4	
Cliente OPC UA	mar 29/06/23
Plugin CoppeliaSim	mar 12/07/23
Fase 4 Completada	vie 17/07/23
Fase 5	
Fase 5 Completada	vie 31/07/23

Tabla B.2: Planificación detallada del proyecto por tareas

Tarea	Trabajo	Duración	Comienzo	Fin
1. Investigación y Planificación	**84.86h**	**10.5d**	**lun 20/03/23**	**lun 03/04/23**
Documentación RTDE	18.86h	3d	lun 20/03/23	mié 22/03/23
Formación sobre OPC UA	54h	8d	jue 23/03/23	vie 31/03/23
Fase 2 Planificación	12h	2d	vie 31/03/23	lun 03/04/23
2. Programa RTDE	**78h**	**11.83d**	**mar 04/04/23**	**mié 19/04/23**
Desarrollo del programa	42h	7d	mar 04/04/23	mar 11/04/23
Prueba del programa	24h	4d	mié 12/04/23	lun 17/04/23
Fase 3 Planificación	12h	2d	mar 18/04/23	mié 19/04/23
3. Comunicación OPC UA	**300h**	**36.5d**	**jue 20/04/23**	**vie 16/06/23**
Instalar herramientas OPC UA	36h	6d	jue 20/04/23	mié 26/04/23

Crear Modelo de información	54h	9d	mié 26/04/23	vie 05/05/23
Compilación Modelo de información	24h	4d	vie 05/05/23	mié 10/05/23
Planificación programa servidor	12h	2d	mié 10/05/23	jue 11/05/23
Desarrollo thread del servidor	66h	11d	vie 12/05/23	mié 31/05/23
Incluir thread del robot	24h	4d	jue 01/06/23	lun 05/06/23
Pruebas comunicación robot-servidor	24h	4d	mar 06/06/23	jue 08/06/23
Extensión para varios robots	30h	5d	jue 08/06/23	mar 13/06/23
Optimización del código	24h	4d	mié 14/06/23	vie 16/06/23
4. Simulación en CoppeliaSim	**192h**	**22,67d**	**lun 19/06/23**	**mié 19/07/23**
Creación de escenas y modelos	12h	2d	lun 19/06/23	mar 20/06/23
Planificación del programa del plugin	12h	2d	mié 21/06/23	vie 23/06/23
Implementación Cliente OPC UA	42h	7d	jue 22/06/23	jue 29/06/23
Hilo Actualización de la simulación	60h	10d	jue 29/06/23	lun 10/07/23
Interfaz Gráfica de Usuario	24h	4d	lun 10/07/23	mié 12/07/23
Pruebas de Funcionamiento	24h	4d	jue 13/07/23	lun 17/07/23
Optimización del código	18h	3d	lun 17/07/23	mié 19/07/23
Documentación	30h	5d	jue 20/07/23	mar 25/07/23
5. Tests y Análisis	**84h**	**14d**	**jue 20/04/23**	**vie 16/06/23**
Depuración del código	18h	3d	mar 18/07/23	vie 21/07/23
Tests Fallos de Memoria	30h	5d	jue 20/07/23	mié 26/07/23
Pruebas un solo robot	24h	4d	mar 25/07/23	vie 28/07/23
Pruebas multirobot	12h	2d	vie 28/07/23	lun 31/07/23

En la figura presentada en la página a continuación se ilustra el diagrama de Gantt de seguimiento de la planificación del proyecto. El diagrama contiene la siguiente información:

- Duración real de las tareas presentadas en barras de color azul oscuro.
- Duración planificada de la tarea en azul claro no es visible puesto que es ocultada por el punto anterior.
- Duración de las tareas resumen en barras grises.
- Porcentaje de complexión de la tarea en la fecha establecida. Un porcentaje menor del 100 % indicaría una finalización de la tarea previa a la fecha establecida, mientras que un porcentaje mayor indicaría un retraso.
- Fechas de complexión de las fases.

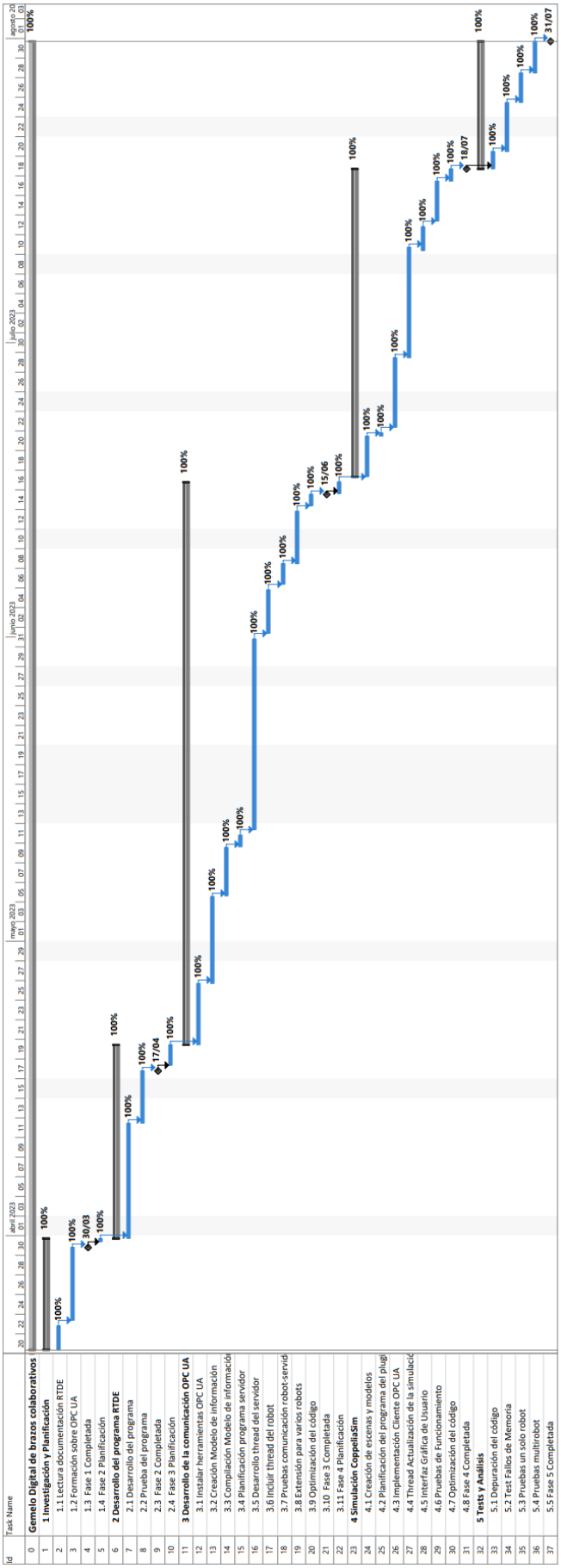


Figura B.1: Diagrama de Gantt de la planificación de las tareas del proyecto.

Apéndice C

Despliegue de la Solución

C.1 Compilación del Modelo de Información

Una de las grandes ventajas que se han mencionado a lo largo de este documento es la interoperabilidad y flexibilidad de la solución. Estas características son debidas en gran parte al uso de un modelo de información personalizado. El modelo de información puede ser fácilmente adaptado a cambios en la aplicación y compilado de nuevo para cambiar la estructura de nodos del servidor. Como se ha explicado en la 4.2, la herramienta utilizada para compilar el modelo de información y los Companion Specification es el UA-ModelCompiler. A continuación se detallan los principales pasos de su instalación y el uso de la herramienta para compilar el RCS y el modelo de información personalizado utilizado en este proyecto.

C.1.1 Instalación de Dependencias

Como ya se ha dicho es necesario instalar el UA-ModelCompiler, que se puede encontrar en el GitHub oficial de la OPC Foundation `https://github.com/OPCFoundation/UA-ModelCompiler`. La aplicaicón está disponible para Windows y Linux, aunque se recomienda utilizar la versión de Linux (Ubuntu). A continuación se presentan las instrucciones para la compilación de la herramienta UA-ModelCompiler:

```
sudo apt-get install mono-complete

git clone https://github.com/OPCFoundation/UA-ModelCompiler.git

git submodule update -init -recursive

cd UA-ModelCompiler

wget https://dist.nuget.org/win-x86-commandline/latest/nuget.exe

mono nuget.exe install -OutputDirectory packages ModelCompiler/packages.config

msbuild "ModelCompiler Solution.sln/p:TargetFrameworkVersion="v4.5"
```

Destacar que para la correcta compilación de la herramienta es necesario disponer de la herramienta *mono* de Linux, y de la herramienta *nugget* para disponer de la compilación con el comando *msbuild*. Aunque el uso de este comando es el más recomendado, la compilación de modelos de información que utilicen algunos tipos de OPC UA complejos puede dar problemas, en cuyo caso se recomienda sustituir el comando por:

```
# Instala dotnet para Ubuntu 22.04

sudo apt-get install -y dotnet-sdk-7.0

dotnet msbuild "ModelCompiler Solution.sln/p:TargetFrameworkVersion="v4.5"
```

C.1.2 Compilación de Modelos de Información

Tal como indica la ayuda de la herramienta accesible mediante el comando: `ModelCompiler compile -h`, se han de proporcionar el fichero del modelo de información a compilar y las dependencias de este. En este caso, primero se proporciona el fichero XML del RCS, Opc.Ua.Robotics.NodeSet2.xml, disponible en el repositorio UA-Nodeset. A continuación se han de proporcionar las dependencias de este modelo, en este caso es únicamente el Device Companion Specification, Opc.Ua.Di.NodeSet2.xml. Tras las dependencias se proporciona el fichero cvs indicando los nuevos tipos creados por el modelo de información copilado, Opc.Ua.Robotics.NodeIds.csv. Por último, se puede indicar el lugar de destino de la compilación en una nueva carpeta.

El siguiente comando muestra cómo compilar el modelo de información personalizado creado para el proyecto utilizando el UA-ModelCompiler:

```
# El comando se ha d ejecutar en la ruta: cd ruta/al/directorio/UA-ModelCompiler/build/

./Opc.Ua.ModelCompiler compile -d2  /UR_Digital_Twin/model/URDigitalTwinModel.xml
-d2  /UR_Digital_Twin/UA-Nodeset/DI/Opc.Ua.Di.NodeSet2.xml -d2  /UR_Digital_Twin/deps/Rob
-cg  /UR_Digital_Twin/model/URDigitalTwinModel.csv -o2  /UR_Digital_Twin/Published
```

C.2 Compilación de la Aplicación

En esta sección se detalla el proceso compilación de la aplicación del Gemelo Digital, el cual comprende los siguientes pasos:

1. Instalación de dependencias.
2. Compilación del programa del servidor.
3. Instalación de CoppeliaSim versión Edu 4.0.5 o 4.2.
4. Compilación del plugin.
5. Instalación del plugin.

Para comenzar con la compilación se ha de tener el proyecto estructurado de la forma mostrada en la figura. En caso de obtener el proyecto vía *git clone[...]* no será necesario modificar la estructura.

C.2.1 Instalación de Dependencias

La primera dependencia necesaria que posibilita además la realización de test y pruebas unitarias del hilo RobotManager es la librearía RTDE. Para el uso de las capacidades en tiempo real de la libraría se ha de disponer de un kernel con capacidades de tiempo real. Mientras que el kernel de Windows NT dispone de capacidades de tiempo real por defecto, Ubuntu no dispone de estas capacidades. Para habilitar las capacidades de tiempo real en Ubuntu es necesaria la instalación de un parche. Este parche depende de la versión de Ubuntu utilizada y puede encontrarse en la documentación oficial de Ubuntu.

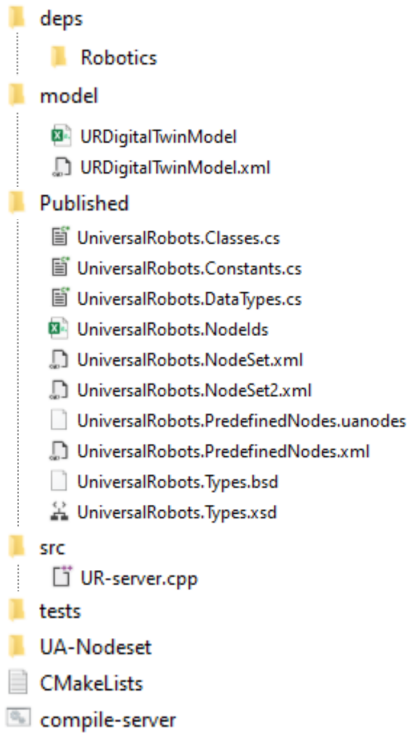


Figura C.1: Estructura mínima necesaria de los ficheros y carpetas del proyecto para su correcto funcionamiento.

La instalación de la libraría RTDE en Ubuntu, es realmente sencilla y puede realizarse mediante los siguientes comandos:

```
   sudo add-apt-repository ppa:sdurobotics/ur-rtde
sudo apt-get update
sudo apt install librtde librtde-dev
```

La segunda dependencia y la más importante, es la librería open62541. En el proyecto se ha utilizado la librearía principalmente en Ubuntu, cuyo proceso de instalación se explica a continuación. Para consultar el proceso de instalación en Windows puede consultarse en la documentación de open62541.

Antes de instalación de open62541 en Ubuntu, han de instalarse las dependencias y construir (build) la librería. Para ello se han se ejecutar los siguientes comandos:

```
   sudo apt-get install git build-essential gcc pkg-config cmake python
# Configuración adicional
sudo apt-get install cmake-curses-gui # ccmake GUI
sudo apt-get install libmbedtls-dev # encriptado
sudo apt-get install check libsubunit-dev # tests
sudo apt-get install python-sphinx graphviz # documentación
sudo apt-get install python-sphinx-rtd-theme # documentación
# Construcción de la biblioteca
```

```
cd open62541
mkdir build
cd build
cmake ..
make
```

Después de esta construcción, se procede con la instalación:

```
   git clone https://github.com/open62541/open62541.git git submodule update
-init -recursive

   mkdir build && cd build

   cmake -DBUILD_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=RelWithDebInfo -DUA_NAMESPACE_ZERO=FUL
..

   make

   make install
```

C.2.2 Compilación del Programa del Servidor

En el caso de que se disponga del fichero bash mostrado en la Figura 4.2 para automatizar la compilación, la ejecución del fichero con el modo de compilación deseado genera automáticamente los archivos ejecutables del servidor o los test.

```
./compile-server.sh [Release, Debug_asan, Debug_def]
```

En caso contrario la compilación manual del servidor se puede realizar mediante los siguientes comandos:

```
cd ur-digital-twin

mkdir build

cd build

cmake -DCMAKE_BUILD_TYPE=Release ..

make -j

# Muestra el uso del servidor
```

```
./opcua-universal-robots-server -help
```

Tras realizar cualquier cambio en el código del servidor se debe volver a realizar esta compilación eliminando los archivos de la anterior compilación en la carpeta *build*.

C.2.3 Compilación e Instalación del Plugin de CoppeliaSim

El despliegue del plugin para CoppeliaSim necesita tener una instalación de CoppeliaSim 4.2.0 con los plugin simStubsGen, simPlusPlus y simUI para funcionar. Estos plugin vienen instalados por defecto en la versión oficial binaria de CoppeliaSim 4.2.0 que se puede encontrar en: `https://www.coppeliarobotics.com/previousVersions`. También se ha utilizado la versión educativa de CoppeliaSim 4.0.5.

Para instalar el plugin se han de disponer de los ficheros de la Figura 5.8. Estos ficheros han de estar en una carpeta denominada simOpcuaClient (el nombre del plugin) ubicada en la capeta de *Programming* dentro del directorio principal de CoppeliaSim. Entonces la ejecución de los siguientes comandos construye e instala el plugin en CoppeliaSim:

```
mkdir -p build && cd build

cmake -DCMAKE_BUILD_TYPE=Release ..

cmake -build .

cmake -install .
```

Apéndice D

Alineación ODS

D.1 Desarrollo de la Alineación con los ODS

A continuación se presenta una tabla indicando la alineación del trabajo con los distintos ODS de la Agenda 2030.

Tabla D.1: Alineación del trabajo con los Objetivos de Desarrollo Sostenible

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Proceede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**			X	
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**			X	
ODS 11. **Ciudades y comunidades sostenibles.**		X		
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

La implementación de gemelos digitales en la industria de la robótica, como se presenta en esta tesis, contribuye directamente al ODS 8. Este objetivo se centra en promover el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo, y el trabajo decente para todas las personas. Al permitir el mantenimiento predictivo de robots colaborativos como los brazos UR, se minimizan los costos de tiempo de inactividad y se optimizan los recursos, lo que fomenta un ambiente propicio para el crecimiento económico. Además, al mejorar la eficiencia de los procesos de producción, se generan oportunidades de empleo cualificado y se fomenta un entorno laboral más seguro y colaborativo, en línea con los principios de trabajo digno.

Esta tesis también se alinea también con el ODS 9, que tiene como objetivo construir infraestructuras resilientes, promover la industrialización inclusiva y fomentar la innovación. La implemen-

tación de gemelos digitales en la industria es una clara manifestación de la innovación tecnológica en el contexto de la Industria 5.0. Estos modelos digitales mejoran la eficiencia y la adaptabilidad de los procesos de fabricación, lo que a su vez promueve la productividad industrial. Además, al desarrollar una aplicación que integra comunicaciones OPC UA y simulación en CoppeliaSim, este trabajo se suma a la promoción de infraestructuras tecnológicas de vanguardia y la adopción de prácticas industriales avanzadas.

La tesis también se relaciona con el ODS 12, que busca garantizar patrones de producción y consumo sostenibles. Los gemelos digitales en la robótica colaborativa permiten un uso más eficiente de los recursos y una reducción significativa del desperdicio. Al prevenir fallos inesperados y optimizar la programación de mantenimiento a través del monitoreo constante, se evita la sustitución innecesaria de componentes y se prolonga la vida útil de los activos físicos. Esto se traduce en una producción más responsable y en un menor impacto ambiental, en línea con el enfoque de producción y consumo responsables.

Además, la tesis no solo contribuye a los aspectos técnicos y científicos relacionados con los gemelos digitales y la robótica colaborativa, sino que también resalta el valor de las alianzas estratégicas en la búsqueda de soluciones innovadoras y sostenibles en la industria al formar parte de un proyecto europeo. En el contexto del proyecto europeo del que forma parte, colaboran diversas empresas e instituciones, tanto públicas como privadas, lo que es un claro ejemplo de cómo las alianzas estratégicas pueden impulsar el progreso hacia un desarrollo sostenible. Estas colaboraciones fomentan la transferencia de conocimientos y tecnologías, promueven la innovación y permiten abordar desafíos complejos a través de un enfoque conjunto.

Bibliografía

BRC (2023). *Collaborative Robots Global Market Report 2023*. Rep. de inv. (Accessed on 08/01/2023). The Business Research Company (vid. pág. 3).

CMake Documentation (2023). CMake. URL: `https://cmake.org/cmake/help/v3.20/index.html` (vid. pág. 40).

CoppeliaRobotics (2023). *Manual de usuario de CoppeliaSim*. 4.5. CoppeliaRobotics (vid. pág. 24).

Lippman, Stanley B. (2013). *C++ primer*. Addison-Wesley. ISBN: 0321714113.

Mahnke, Wolfgang, Stefan-Helmut Leitner y Matthias Damm (2009). *OPC Unified Architecture*. Springer Berlin Heidelberg. ISBN: 9783642088421. DOI: `10.1007/978-3-540-68899-0` (vid. pág. 12).

Nath, Shyam Varan y Pieter Van Schalkwyk (2021). *Building Industrial Digital Twins. Design, Develop, and Deploy Digital Twin Solutions for Real-World Industries Using Azure Digital Twin*. Packt Publishing, Limited, pág. 195. ISBN: 9781839219078 (vid. pág. 8).

OPC Foundation (2018). "Diapositivas del programa OPC UAcademics". En: *OPC UAcademics*. Unidades 1 a 4. OPC Foundation (vid. pág. 12).

— (2022a). "OPC 10000-1. Part 1: Overview and Concepts". En: *OPC UA Specification*. OPC Foundation (vid. pág. 14).

— (2022b). "OPC 10000-3. Part 3: Address Space Model". En: *OPC UA Specification*. OPC Foundation (vid. pág. 15).

open62541 (2021). *open62541 Documentation*. 1.3. open62541 (vid. pág. 41).

open62541 repository (2023). Último commit: ff73268829359639531ff02905c889f73a77b408. open62541. URL: `https://github.com/open62541/open62541` (vid. pág. 42).

Øvern, Aksel (2018). "Industry 4.0 - Digital Twins and OPC UA". En: NTNU (vid. pág. 8).

Profanter, Stefan (2023). *How to create custom OPC UA Information Models*. URL: https://profanter.medium.com/how-to-create-custom-opc-ua-information-models-1e9a461f5b58 (vid. pág. 15).

Rohmer, E., S. P. N. Singh y M. Freese (2013). "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework". En: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com (vid. pág. 62).

simSkel repository (2023). Último commit: ed52889b93c72977143a6a0290dc0ffb50085eee. CoppeliaRobotics. URL: https://github.com/CoppeliaRobotics/simSkel (vid. pág. 68).

simUI repository (2023). Último commit: c88a0d235ce5539ccdda8a4bf1ea7c0584b85e4d. Coppelia Robotics. URL: https://github.com/CoppeliaRobotics/simUI/tree/master.

Swidzinski, Rafal (2022). *Modern CMake for C++. Discover a Better Approach to Building, Testing, and Packaging Your Software*. Packt Publishing, Limited. ISBN: 9781801070058 (vid. pág. 54).

UA Nodeset repository (2023). Último commit: 78954a2a1e31a6624828771ddf031648554f43df. OPC Foundation. URL: https://github.com/OPCFoundation/UA-Nodeset (vid. pág. 44).

UA-ModelCompiler repository (2023). Último commit: c7a9be18743761ca79a7b68f94486a5f475ddc5e. OPCFoundation. URL: https://github.com/OPCFoundation/UA-ModelCompiler (vid. pág. 43).

Universal Robots (2023). *Real-Time Data Exchange (RTDE) Guide*. https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/. Accessed: 09/08/2023 (vid. pág. 31).

Wang, Yulin y Dr James Edmondson (2022). *Collaborative Robots (Cobots) 2023-2043: Technologies, Players and Markets*. Rep. de inv. (Accessed on 08/01/2023). IDTechEx (vid. pág. 3).