UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN DE "PICK AND PLACE" CON EL ROBOT COLABORATIVO UR3

Autor: Mario Sánchez García

Tutor: Alberto Brunete González
Departamento: Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

Madrid, Junio, 2023

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN DE "PICK AND PLACE" CON EL ROBOT COLABORATIVO UR3

Firma Autor

Mario Sánchez

Firma Tutor

TÌtulo: Diseño e implementación de una aplicación de ”Pick and Place” con el robot colaborativo UR3
Autor: Mario Sánchez García
Tutor: Alberto Brunete González

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día ..... de ..... de ... en ....., en la Escuela Técnica Superior de Ingeniería y Diseño Industrial de la Universidad Politécnica de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

A toda mi familia y amigos.

Su apoyo incondicional durante toda mi vida académica y personal, ha hecho de esta, un camino mejor.

Gracias por su cercanía y cariño, por recordarme cada día mi capacidad y potencial para cumplir todos mis objetivos.

A mi tutor.

Agradecerle la orientación y los conocimientos que me ha brindado durante toda su implicación en el proyecto.

A la Universidad Politécnica de Madrid y a la gente que compone la ETSIDI.

Agradecido por haberme prestado recursos excepcionales para desarrollarlo.

Muy agadecido por las personas que ha puesto en mi camino durante estos últimos cuatro años.

Resumen

Este Trabajo de Fin de Grado desarrolla el control del robot colaborativo UR3e combinando el uso del estándar en comunicación para sistemas robóticos, ROS2, con el manejo de visión por computador para la detección de objetos. El resultado es la implementación de aplicaciones de Pick and Place de forma autónoma.

El propósito principal del proyecto es el diseño y desarrollo de un sistema eficiente que permita al robot interactuar con su entorno gracias a la percepción dotada por una cámara tridimensional. Además, su finalidad se centra en que el usuario pueda seleccionar entre diferentes casos de uso a través de una interfaz gráfica de usuario.

En primer lugar, se construye un sistema de comunicaciones programado en *Python* que divide el funcionamiento interno en tres secciones: el control del movimiento del robot, la localización tridimensional de los objetos y la interacción con el usuario. La idea esencial se basa en la integración de todas estas, obteniendo una implementación real y asequible para cualquier tipo de público.

Una vez asentada la familiarización con ROS2, se investiga el funcionamiento interno del UR3e desde la planificación y ejecución de trayectorias hasta el uso de su herramienta que consiste en una pinza paralela de manipulación de objetos. Asimismo, se lleva a cabo su representación en un visualizador tridimensional pudiendo seguir el funcionamiento en tiempo real de forma simulada o en directo. Se estudian los diferentes algoritmos existentes de visión por computador y se aplican para conseguir el reconocimiento y obtención de la posición exacta de los objetos. En consecuencia, se alinean los sistemas de referencia de ambos dispositivos consiguiendo así, una autonomía en los tres casos de uso creados.

Por último, se elabora un medio de interacción con el usuario y un sistema de comunicaciones que unifica el proyecto completo caracterizándose por su usabilidad y eficiencia en términos socioeconómicos, además de destacar por su robustez y escalabilidad.

Palabras clave: Pick and Place, Caso de uso, Robot, Control, Planificación, ROS2, Nodo, Cámara, Imágenes, Interfaz.

Abstract

This project aims to develop the software control of the UR3e collaborative robot. The project is carried out combining the use of a standard on communications for robotic systems called ROS2 and the use of computer vision for the purpose of detecting objects. The result is the implementation of Pick and Place applications in an autonomous way.

The main objective of the project is the design and development of an efficient system which allows the robot to interact with its surroundings due to the visual perception. This object recognition is led by a three-dimensional camera. Moreover, its goal is focused on the possibility for the user to select between three different use cases through a graphical user interface.

In the first place, a communication system programmed in Python is built which separates the intern operation into three different areas: the control of movement, the three-dimensional location of the objects and the user interaction. The final idea is based essentially on the integration of all of these parts, getting a real and achievable implementation for any kind of public.

Once the understanding of the basis of ROS2, the internal operation of the UR3e is investigated from the planning and execution of trajectories up to the use of the tool, which consists of a parallel gripper intended for the manipulation of objects. Furthermore, the visualization of the robot takes place leading to the tracking of the functioning in real time in a simulated or real way. The existing algorithms of computer vision are studied and applied resulting in the recognition and exact positioning of the objects to manipulate. Consequently, the reference systems of both devices are aligned resulting in autonomy for every use case created.

Finally, a user interaction and communication system are elaborated. Both of them merge into the last version of the project, characterized by its usability and efficiency. In addition, the most positive features of it are its robustness and its scalability.

Keywords: Pick and Place, Use case, Robot, Control, Planning, ROS2, Node, Camera, Images, Interface.

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1

Introducción

1.1. Motivación del proyecto

La motivación de este proyecto radica en la continua innovación que las tecnologías de hoy en día presentan, en un mundo en el que la mayoría de procesos han sido automatizados y optimizados con la ambición de mejorar la industria y la calidad de vida de las personas. Además, se destaca la relevancia de soluciones con plataformas como ROS2 y robótica colaborativa con la ayuda de visión artificial teniendo una gran importancia en todos los ámbitos.

El principal interés del proyecto es el aprendizaje que representa dentro del ámbito de la robótica. La creación de aplicaciones con robótica colaborativa aporta un enfoque crítico al autor tanto en su trayectoria académica como profesional. Asimismo, marca el final del grado de Ingeniería Electrónica Industrial y Automática desarrollado durante los últimos cuatro años.

Por un lado, este proyecto supone una gran oportunidad de adquirir habilidades y conocimientos completamente adicionales al grado, proyectando habilidades e intereses enriquecedores a nivel profesional. Y, por otro lado, es una gran opción para desarrollar aspectos técnicos como la gestión de un proyecto real. El pensamiento crítico que este supone, fomenta la creatividad y capacidad de resolución.

1.2. Objetivos

El objetivo principal de este TFG consiste en desarrollar aplicaciones de Pick and Place con el robot colaborativo UR3e de forma autónoma con el fin de manipular ciertos objetos dentro del espacio de trabajo establecido.

Uno de los propósitos más relevantes del proyecto ha sido llevarlo a cabo con una plataforma de *software* considerada como un estándar en robótica denominada ROS2. El lenguaje de programación utilizado ha sido Python, lo que ha acarreado la familiarización y óptimo manejo de este. Asimismo, su alcance abarca el uso de conceptos como la programación orientada a objetos y en tiempo real. La implementación de soluciones con algoritmos de visión artificial y diferentes tipos de comunicaciones también han sido otros de los objetivos de estudio.

La propuesta inicial es la creación de estas aplicaciones mediante el reconocimiento de piezas con el uso de una cámara tridimensional, lo que ha implicado adentrarse en uno de los pilares de innovación dentro de este tipo de tareas. Su meta consiste en automatizar

las aplicaciones de forma que se pudieran ubicar los objetos con visión por computador y, posteriormente, manipularlos con el uso del robot y de su propia herramienta.

En cuanto al contexto del proyecto, se ha pretendido encontrar una solución escalable tanto a nivel de compatibilidad técnica o demográfica como para futuras versiones y herramientas que se presenten. Eston supone la ambición de añadir mejoras a la propuesta implementada. De hecho, su finalidad es poder reutilizarse para otro tipo de robots manipulativos con el ánimo de ayudar a otros desarrolladores y que pueda servir de orientación para otras aplicaciones.

Como objetivo final del proyecto, se lleva a cabo una interacción simple con el usuario, con la intención de compactar el funcionamiento complejo de este en una aplicación de alto nivel que resulte asequible para cualquier persona no especializada en el campo técnico de la robótica colaborativa.

1.3. Importancia

Este proyecto tiene una especial relevancia dentro de la sección de robótica de la universidad, debido a que los robots colaborativos UR3e fueron adquiridos el año pasado, y esta es la primera toma de contacto de su uso con *ROS2*. Por lo que, puede tener una gran utilidad en un futuro a nivel educativo o de investigación dentro de la Universidad. Al igual que ocurre en el ámbito educativo, también puede suponer un buen material de apoyo para otros usuarios.

Este proyecto marca un punto de inflexión en la experiencia del autor, ya que tiene una aplicación práctica de todos los conocimientos teóricos adquiridos hasta la fecha. Este representa una oportunidad de manejar materiales de robótica de gran calidad. Los campos de investigación a los que estos se extrapolan son de elevado valor dentro de la industria actual.

1.4. Estructura del documento

A continuación se detalla el contenido de cada capítulo en la tabla 1.1 para facilitar la lectura del documento. La organización de este corresponde a un documento de carácter científico-técnico.

1.4.1. Recursos utilizados

En cuánto al material complementario utilizado en la memoria, cabe destacar que aquellos recursos que son de medios externos, incluyen un pie de página con el link al sitio original o su correspondiente referencia bibliográfica. Todo recurso que no contenga una referencia ha sido elaboración propia del autor. Este material incluye los siguientes recursos:

- Imágenes descriptivas.
- Tablas explicativas.
- Códigos relevantes.
- Flujogramas de funcionamiento.
- Diagramas técnicos de *software* y auxiliares.

Capítulo	Descripción
1	Realiza una introducción exponiendo el contexto e importancia y objetivos del tema elegido.
2	Analiza el estado del arte del tema con el ánimo de mostrar la situación actual y antecedentes del mismo.
3	Consiste en un marco teórico, donde se explican todas las nociones básicas necesarias para comprender el proyecto.
4	Se expone el diseño completo del proyecto desde los requisitos iniciales hasta su modelado físico y diseño del *software*.
5	Incluye el desarrollo del proyecto donde se explica la implementación de este en profundidad con material de apoyo que lo fundamenta.
6	Ilustra los resultados obtenidos de la implementación final, así como una breve discusión de estos. Además, se analizan el impacto social, económico y sostenible que supone.
7	Describe la gestión del proyecto durante todo su ciclo de vida.
8	Se encuentran las conclusiones finales del proyecto con un añadido de posibles mejoras.
Anexos	En primer lugar, se crea un manual de usuario con la ambición de que este TFG pueda ser usado por otras personas que lo necesiten o quieran ponerlo en práctica. Y, adicionalmente, se añaden materiales de apoyo al desarrollo del proyecto.
Bibliografía	Recopila todas las referencias bibliográficas utilizadas en este documento.

Tabla 1.1: Estructura del documento

Capítulo 2

Estado del arte

En este capítulo se va a estudiar la historia de las aplicaciones de Pick and Place y el estado actual de las mismas, así como las diferentes tecnologías usadas y su contexto industrial.

2.1. Aplicaciones de Pick and Place

Una aplicación de Pick and Place se refiere a la tarea de recoger objetos de una ubicación cualquiera y depositarlos en otra ubicación específica. Este tipo de aplicaciones han estado presente en todo tipo de industrias desde hace muchos años, como puede ser la industria de la logística, alimentaria o farmacéutica.

Estas actividades suelen verse en todas las etapas de las líneas de producción en mayor o menor medida por lo que, a pesar de su poca ergonomía, resultan tener un papel muy importante requiriendo precisión, velocidad y, por supuesto, seguridad.

En el pasado, estas operaciones se llevaban a cabo de forma manual, con el uso de la fuerza y habilidad de muchos trabajadores, los cuales eran expuestos a una carga adicional, mental y, sobre todo, física. Estas actividades son consideradas poco ergonómicas debido a su repetitividad y en muchas ocasiones, forzaban posturas no adecuadas para los trabajadores y obligaban a estos de estar en espacios de trabajo perjudiciales por ruido, vibraciones y muchos otros factores. [11]

En consecuencia, este tipo de tareas se empezaron a automatizar desde el uso de algunos sistemas mecánicas o neumáticos hasta el día de hoy, en el que gracias a la robótica colaborativa estas tareas se han automatizado por completo en aquellas industrias donde es necesaria la manipulación de objetos, aumentando así la productividad y calidad de los productos de forma creciente. Además, con el ritmo al que está avanzando la tecnología hoy en día, es de esperar que, en un futuro cercano, este tipo de aplicaciones robotizadas tenga una mayor presencia dentro de una amplia variedad de sectores como el sector servicios, la educación y salud o medicina.

Este grado de automatización ha logrado conseguir una optimización de los procesos muy superior respecto a las técnicas convencionales descritas. Del mismo modo que se usa la robótica colaborativa, estas técnicas han avanzado exponencialmente gracias a la incorporación de sensores y dispositivos que utilizan visión artificial para recopilar información del entorno constantemente, llevando a las aplicaciones de Pick and Place a otro nivel. Esta mejora surge con el ánimo de lograr la adaptación de este tipo de trabajos ante cualquier cambio inesperado, como trabajadores que entran inesperadamente en el espacio de trabajo o situaciones defectuosas. [31]

2.1.1. Casos prácticos

A continuación, se van a presentar diferentes casos reales en los que se ha utilizado este tipo de aplicaciones de Pick and Place con la ayuda de robots colaborativos y sistemas de visión.

- En primer lugar, la industria automotriz es una de las que más rápido adoptaron este tipo de tecnologías ya que requiere de trabajos laboriosos con grandes cargas en escenarios poco seguros para las personas. Es por esto, que utilizan robots colaborativos para realizar tareas como el ensamblaje de piezas, soldadura, pintado o cualquier transporte de materiales. Ford es una empresa automotriz fundada hace más de cien años en la que implementaron estas tecnologías y así lo afirmaba uno de sus ingenieros:

  [3]
  Según Daniel Llin Terol, ingeniero de líneas de ensamblaje,

  > "Gracias a la estandarización, el coste de los robots y el hecho de no tener que instalar un vallado ni estaciones grandes, hemos conseguido un ROI mucho mayor que con robótica tradicional"

- El segundo escenario está centrado en las tareas de Pick and Place dentro de la industria alimentaria, poniendo como ejemplo a Ocado, uno de los supermercados en línea más grandes del mundo. Esta empresa lleva a cabo tareas de manipulación de alimentos en contenedores con una gran frecuencia, poniendo en práctica sistemas robóticos para dichas tareas. Además, se llevó a cabo un estudio en el que transportaban cinco alimentos distintos de un contenedor a otro, dividiendo estas tareas en diferentes fases: pre-agarre con una pinza como herramienta del brazo robótico, agarre, transporte y depósito de alimentos. A pesar de que experimentaron ciertos problemas con el agarre, en este estudio se descubrieron que dichas tareas resultaron ser eficientes de forma rápida y reconfigurable. [32]

- El tercer caso se basa en la tecnología de montaje en superficies de componentes electrónicos, como *chips* o diferentes componentes delicados fabricados con silicio. En este caso, la línea de producción es el micro ensamblaje, en el que la robótica colaborativa se encarga de manipular y ensamblar componentes con una escala muy reducida. Este tipo de áreas requieren de una gran precisión debido a su tamaño, así como de un especial cuidado, por lo que cuentan con sistemas añadidos de visión para solventarlo. Se llevó a cabo otro estudio en este campo que verificó la mejora en sus resultados dividiendo las tareas en dos pasos: una planificación y control global para su ubicación deseada, y posteriormente, con ayuda de visión artificial, su correcta manipulación. [19]

- Por último, la robótica colaborativa es muy común en aplicaciones de Pick and Place para empresas de logística. Esto se debe a la gran necesidad de cargar y descargar mercancías, transportar objetos, e incluso con la ayuda de visión artificial, hacer clasificación y etiquetado de objetos. Con el propósito de fundamentar la rentabilidad y el rendimiento gracias a este tipo de tecnologías, a continuación se expone una cita de un profesional en el sector:

  Según Brian Tu, director de Ingresos de DCL Logistics,

[0] Surface Mount Technology

«El sistema robótico puede efectuar en dos horas el trabajo que haría un equipo de cinco personas en todo un día»
«Con los cobots de UR ahorramos más de un 50 % en costes salariales».[1]

2.2. Contexto industrial

La industria ha tenido un crecimiento y desarrollo exponencial durante las últimas décadas, adoptando diferentes tendencias en función de los requisitos de la sociedad del momento. Los robots colaborativos han supuesto un gran valor para la Industria 4.0, puesto que esta trataba de involucrar tecnologías emergentes, digitalizando las empresas con materiales y procesos de producción positivos desde un punto de vista sostenible. Por supuesto, la inclusión de tecnologías como estas u otras de alto nivel suponen una gran inversión económica, resultando como un desafío financiero hace algunos años. [20]

Pero actualmente, el efecto que está surgiendo de la integración hombre-máquina, ha llevado a apostar por esta industria. El hecho de que los robots colaborativos puedan trabajar entre ellos o con humanos ha supuesto una optimización de procesos extraordinaria que ha acarreado mayor calidad a un menor coste gracias a efectos como la accesibilidad, seguridad o flexibilidad. [36]

De hecho, la versión siguiente, conocida como Industria 5.0, está tratando de transformar el sector de la industria en espacios inteligentes basados en tecnologías IoT, en los que este tipo de interacciones mencionadas estarán completamente estandarizadas. [34]

El mercado actual de robótica colaborativa es muy competitivo y todas las empresas involucradas buscan la continua innovación para seguir siéndolo. Una de las claves de este impulso ha sido el surgimiento de *startups* y sus colaboraciones con grandes empresas. [22].

Se ha creado un mapa del mundo ilustrando en este las empresas con más importancia e influencia a día de hoy según su zona geográfica:

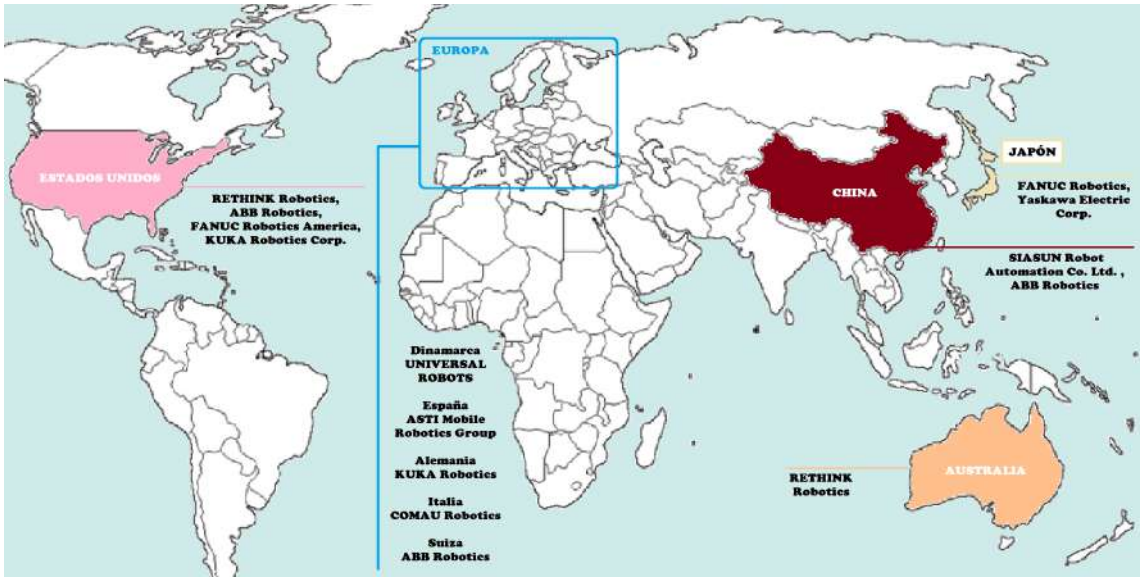


Figura 2.1: Empresas más importantes a nivel mundial

[1] www.universal-robots.com/es/productos/robot-ur3/

Como se puede observar en la figura 2.1, los países que más participación e iniciativa de instalación están teniendo son China, Japón, Corea del Sur, Estados Unidos y Alemania. No obstante, su crecimiento se ve presente por todo el mundo. [1]

2.3. Robótica colaborativa

A día de hoy, la robótica cubre una gran variedad de aplicaciones desde la mayoría de industrias hasta servicios. Gracias a su investigación se han descubierto nuevos áreas de innovación que potencian y mejoran las técnicas que existían en el pasado. Entre todos estas distinguidas áreas surge la robótica colaborativa.

Un robot colaborativo o *cobot* es aquel que ha sido diseñado para interaccionar con las personas en espacios de trabajo compartidos. Esta innovadora tecnología tiene como objetivo llevar a cabo la automatización de procesos de forma eficiente y segura. Su forma de trabajo se define mediante la ayuda de una serie de sensores que dotan al robot de un amplio conocimiento de todos los factores que existen a su alrededor. [16]

La gran mayoría de medianas y grandes empresas usan esta tecnología, ya que a día de hoy, dado sus características se ha convertido en una herramienta asequible e interesante de incluir para mejorar el proceso de producción. La mayoría de estos fueron creados con la idea de poder aumentar su alcance hasta pequeñas industrias e incluso algunos hogares.

2.3.1. Ventajas de los robots colaborativos

Rendimiento. Tienen una gran precisión gracias a toda la retroalimentación que presenta mediante la ayuda de dispositivos externos. Asimismo, destaca su alta eficacia dado que pueden llegar a operar las 24 horas del día, sin sufrir variaciones.

Flexibilidad. Poseen la capacidad de rediseñar las tareas del robot con mínimos ajustes en la propia herramienta. Estos son capaces de llegar a un gran porcentaje de puntos dentro de su espacio de trabajo gracias a un diseño compacto y a todos sus añadidos inteligentes. [16]

Facilidad y rápida adaptación. Otorgan la posibilidad a trabajadores no especializados en el campo de la robótica de aprender a usar esta tecnología de forma intuitiva sin ningún tipo de riesgo. Además, son fáciles de instalar y reprogramar con herramientas propias del fabricante.

Seguridad. Disponen de una serie de protocolos diseñados para tener plena confianza en estos. Sumado a esto, todo robot colaborativo tiene su propia documentación para consultar en caso de ser necesario. [6]

Programación. La mayoría de los *cobots* adquiridos por empresas vienen con interfaces programadas por defecto para desarrollar determinadas tareas. Pero, también, existe la opción de modificar su programación e incluir aspectos adicionales.

2.4. Visión Artificial

La visión artificial es una rama dentro del campo de la inteligencia artificial y del mundo de la computación. Esta ciencia está centrada en la creación de sistemas que puedan

procesar y analizar imágenes o vídeos mediante una serie de algoritmos. El objetivo principal de esta potente tecnología es llevar a cabo tareas específicas o reconocimientos de cualquier tipo. Este concepto de visión artificial o visión por computador surgió de la necesidad que tenían algunas aplicaciones robóticas o de cualquier ámbito tecnológico de conocer la información en tiempo real de su entorno. [25]

A raíz del incremento y presente potencial de los robots colaborativos y muchos otros robots de servicio, muchas de las tareas que antes parecían monótonas y perjudiciales para el ser humano a largo plazo, están siendo sustituidas por la robótica. Gracias a añadidos en los diseños como sensores de visión, estos robots son dotados con un conocimiento del entorno suficiente para poder desarrollar estas tareas mencionadas por sí mismos.
Además, gracias a las tecnologías avanzadas de hoy en día como el *middleware* ROS2, la comunión entre los controladores del robot y una cámara o cualquier tipo de sensor, se han convertido en un trabajo sencillo. [30]

La combinación de la visión artificial con la robótica colaborativa hace de las aplicaciones de Pick and Place tareas totalmente flexibles capaces de adaptarse a cualquier situación u entorno. [31]

Además, existen multitud de técnicas de detección y reconocimiento de objetos con una precisión incomparable a la vista humana que ha sido la tendencia hasta los últimos años.

2.4.1. Evolución de la visión por computador

Las tareas de Pick and Place siempre han existido en el mundo de la industria, es decir, la necesidad de trabajar con sistemas de detección de objetos de cualquier tipo es una parte importante del proceso de producción. El inicio de estos sistemas fue basado en la vista humana, siendo esta una habilidad primaria y útil de los humanos, pero en cuánto llegaron nuevas tecnologías como sensores o cámaras, la visión artificial empezó a simular dicha habilidad. [26]

Hace unos años, el procedimiento seguido se basaba en sensores mecánicos, de proximidad, de luz… hasta que llegaron las cámaras analógicas, de las cuales se tomaba una imágen y posteriormente se analizaba por profesionales. Finalmente, con el avance de las tecnologías estás técnicas se mejoraron con algoritmos de visión por computador y el uso del aprendizaje automático.

A continuación se van a exponer varios casos reales en diferentes campos en los que el uso de la visión por computador fue clave:

- Medicina. La rehabilitación de un paciente es un proceso fundamental en la recuperación de cualquier enfermedad, en este caso se estudió en pacientes con ictus hemiparálisis. Se demostró que en contraste con los sensores mencionados o cámaras, la visión por computador ayudó a mejorar la calidad de vida de los pacientes por las siguientes razones: reconocimiento de la acción y método de entrenamiento, estado físico y de actividad del paciente, y trayectorias de movimiento analizando las mejoras del mismo. [48]

- Ocio. Una de las más recientes tendencias ha sido la inclusión del uso de la visión por computador en videojuegos para lograr una interacción humano-máquina inteligente. Se verificó una mejora de la experiencia de estos juegos inteligentes mediante el análisis automático y la captura del reconocimiento fácil con el entendimiento

de gestos. Además, a día de hoy, muchas de estas aplicaciones interactivas como la realidad virtual tienen como mayor aliado a la visión por computador. [37]

- Robótica móvil. Este es uno de los campos más beneficiados del uso de visión por computador dado que genera grandes mejoras en los sistemas de navegación. Se ha implementado este tipo de cambios para evitar obstáculos, para obtener un sistema de autolocalización e incluso para la navegación ininterrumpida. Todos estos sistemas son útiles en transportes a gran escala como submarinos o aviones. [35]

2.5. Entornos de desarrollo de software en aplicaciones de Pick and Place

Este apartado trata las diferentes opciones de desarrollo de *software* que existen para robótica orientada a este tipo de tareas, tanto para robótica colaborativa como para otras tecnologías como los robots cartesianos o *scara*.

2.5.1. Plataformas de comunicación

Se van a estudiar varias de las herramientas más populares en el desarrollo de aplicaciones robóticas:

- ROS2. A raíz de la continuada demanda de la creación de plataformas de *software* para el uso de la robótica en este tipo de aplicaciones, se desarrolló este sistema operativo de código abierto creando una plataforma accesible y versátil. ROS, resultó ser el producto de priorizaciones hechas durante su ciclo de diseño, en el que su énfasis fue la integración a gran escala. Esto supuso un rápido crecimiento de su comunidad gracias a la colaboración y escalabilidad. Además, con la necesidad de la programación en tiempo real, surgió ROS2, una nueva versión que ha sido uno de los objetos de estudio en este TFG. [9]
- *MATLAB/Simulink*. Es una herramienta muy utilizada en el campo de la robótica, control y automatización dada la amplia variedad de posibilidades que ofrece:
  - Modelado y desarrollo de sistemas.
  - Integración de componentes .
  - Simulación de dichos sistemas y análisis de los mismos.
  - Ejecución de trabajo en paralelo.

  Es por esto que es una de las opciones más usadas ya que además, cuenta con grandes herramientas para algoritmos y control orientadas a aplicaciones de Pick and Place. [29]
- De forma adicional, existen otros *frameworks* orientados al control de robótica como Orocos, que está centrado en el control en tiempo real de robots, y muchos otros que son propietarios de fabricantes. Algunos de estos son:
  - ABB RobotStudio. Fabricado por ABB Robotics de origen suizo.
  - Fanuc ROBOGUIDE. Fabricado por Fanuc de origen japonés.
  - KUKA Sim. Fabricado por KUKA de origen alemán.
  - UR Software. Fabricado por Universal Robots de origen danés.

2.5.2. Plataformas de simulación

Una de las fases más importantes en cualquier aplicación robótica es su simulación previa y en tiempo real de la misma. Algunos de los simuladores más usados a día de hoy:

- *OpenRAVE*. Es un entorno de simulación destinado al desarrollo de algoritmos de planificación de movimiento. Es un entorno de código abierto y ofrece una gran serie de herramientas complementarias a la planificación y control de trayectorias, por lo que es muy usado en la industria en aplicaciones de Pick and Place. Una de sus características más atractivas es su reconfigurabilidad y personalización, así como su posible integración con ROS2, facilitando así las comunicaciones. [40]
- *V-REP*. Es otro entorno de simulación que ofrece la posibilidad de probar algoritmos de diferentes tareas con robots de forma realista. El uso de esta plataforma experimental de robótica virtual es muy usada en fases iniciales de proyectos en los que se desea dar credibilidad a la futura implementación que se vaya a realizar. Asimismo, se suele combinar con el uso de ROS2 para llevar a cabo el control final de la implementación.
- *Gazebo*. Es un entorno de simulación tridimensional en el cual se puede realizar el modelo y simulado de robots. Es muy conocido por la visualización tan realista que ofrece incluyendo interacciones con obstáculos del entorno, así como respuesta de sensores inerciales, sensores de LiDAR[2]o cámaras de cualquier tipo. Al igual que los otros dos simuladores, se suele usar con ROS2 y una de las ventajas que tiene a diferencia del resto, es el amplio soporte para diferentes robots existentes como los UR de *Universal Robots*. [39]

[2] *Light Detection and Ranging*

Capítulo 3

Marco Teórico

En este capítulo se describen todos los conceptos necesarios para entender el trabajo realizado.

3.1. ROS2

ROS es un *middleware* de código abierto y gratuito diseñado para potenciar las capacidades de las aplicaciones robóticas existentes. Sus siglas son la abreviatura de *Robot Operating System* y, ROS2, es la segunda y más reciente generación que existe actualmente. Este meta-sistema operativo de aplicaciones robóticas incluye un conjunto de librerías y múltiples herramientas para desarrollar software convirtiéndose en un estándar en la industria actual. En los últimos años ha crecido de forma exponencial y está siendo implementado por todo tipo de configuraciones robóticas, desde brazos antropomórficos industriales hasta cualquier robot móvil. [39]

Figura 3.1: Logo de ROS2 [1]

- Es extremadamente modular, se puede comunicar entre nodos programados en cualquier lenguaje de programación debido a librerías con soporte por la comunidad como *rclpy* para *Python*, por lo que se puede comunicar como ejemplo, un nodo desarrollado en C++ o C con otro en Python. [39]
- Existen múltiples herramientas para controlar y guiar aplcaciones robóticas como *RViz* o *Gazebo*. Además, la mayoría de plataformas de comunicaciones robóticas actuales son complementadas con este *middleware*.
- Sus características técnicas son descritas en el apartado 3.2.

En relación a su expansión, ROS2 posee una gran comunidad de desarrolladores, adquiriendo un gran potencial a nivel mundial. Cualquier persona o equipo puede subir su

[1] `https://www.ros.org`

propia implementación y así, otras personas que necesiten el mismo recurso, podrán aporyarse en este. [39]

3.1.1. Versiones de ROS

Cada una de las versiones de ROS consiste en una distribución que incluye diferentes paquetes administrados. Estas distribuciones son actualizadas en función de las nuevas distribuciones de *Linux* con el objetivo de proporcionar mejoras y nuevas funciones. Actualmente, existen muchas de ellas, empezaron en 2010 con *ROS Box Turtle*, hasta la más reciente, *Humble Hawsbill*. En este proyecto se usa la versión *Foxy* dado que se lanzó en 2020 y la mayoría de recursos existentes están diseñados para esta.

ROS fue creado para poder ser usado por distintos sistemas operativos como *Linux*, *Windows* o Mac. Pero *Ubuntu* es la mejor opción dado que las versiones de ROS van evolucionando conforme a las distribuciones de esta. [39]

Figura 3.2: Versiones de ROS desde su inicio hasta la actualidad [2]

3.1.2. Elementos de ROS2

La información técnica de este apartado ha sido recopilada fundamentalmente del documento [39]. A continuación, se exponen los elementos necesarios de conocer para seguir el funcionamiento del TFG realizado:

- **Nodo.** Un nodo es una entidad o proceso individual encargado de realizar una cierta tarea. Este se puede comunicar con otros nodos mediante el uso de ROS2, formando así un sistema distribuido. Un nodo tiene asociado un amplio abanico de posibilidades como por ejemplo, publicadores o suscriptores de información. Estos pueden suscribirse o publicar de *topics*, crear o usar servicios u acciones y puede incluir parámetros internos o externos. [28]

- **Topic**:. Es un medio unidireccional de datos en los que se publican y de los que se suscriben los nodos, de forma que son fundamentales para el medio de comunicación

[2] `www.theconstructsim.com`
[3] `https://docs.ros.org/en/foxy/Tutorials`

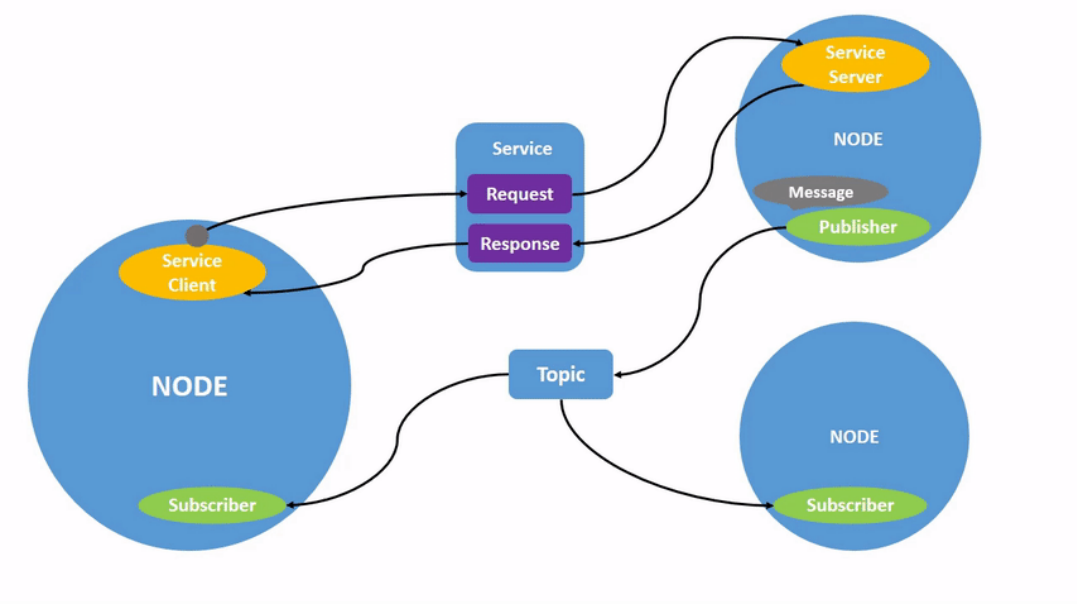


Figura 3.3: Diagrama de nodos con topics y servicio [3]

de ROS2. Se establece una analogía con almacenes de información. Además, poseen una ventaja que consiste en la relación de uno a muchos o de muchos a uno que presentan, pudiendo llegar a cualquier tarea específica.

- **Servicio**. Es un método al que se le introduce una o varias entradas y este se encarga de administrar una o varias respuestas según el algoritmo establecido. Son un recurso muy popular que se divide en dos partes: un servidor que crea el propio servicio y define el propio funcionamiento, y un cliente que será el encargado de hacer uso de este. Tienen una relación de *request-response* como se observa en la figura 3.3.

- **Accion.** Es recurso basado en *topics* y servicios, asemejándose a los servicios con la diferencia de que incluyen un objetivo, un *feedback* y un resultado. Son muy útiles para llevar a cabo tareas elaboradas en robótica que necesite un continuo seguimiento. Por ejemplo, cuando un brazo antropomórfico alcanza cierta posición pedida, existe una retroalimentación del estado de sus articulaciones para comprobar si realmente ha sido alcanzada una determinada posición.

- **Archivos de lanzamiento y parámetros.** Existen archivos encargados de ejecutar varios nodos con sus dependencias específicas a la vez, de forma que la aplicación desarrollada se ponga en funcionamiento con facilidad. De igual modo, tienen la posibilidad de lanzar unos archivos de lanzamiento dentro de otros. Por último, se pueden declarar y utilizar parámetros sobre configuraciones o cualquier información mediante archivos específicos con formato *.yaml* o dentro de los propios nodos.

3.1.3. Configuraciones populares de ROS2

- *Publisher/Subscriber* : Es una de las metodologías más famosas en toda la comunidad de ROS2 debido a sus múltiples usos y ventajas. El publisher publica cualquier tipo de dato en un tiempo determinado y almacena ese dato en un *topic*. Por otro

lado, el suscriptor se subscribe a ese *topic*, recibiendo el dato. De esta forma se logra una comunicación eficaz y rápida.

- Diagrama de nodos: este diagrama muestra todas las relaciones existentes entre nodos y *topics*, obteniendo una idea del funcionamiento genérico del sistema implementado. A modo de ilustración se puede observar la figura 3.3. Se obtienen a través de una herramienta gráfica, denominada *rqt_graph*.

3.1.3.1. Paquetes de ROS2

Se define como una unidad organizativa de *software* que contiene todos los componentes necesarios para implementar cualquier funcionalidad. Es una parte necesaria ya que gracias a esta se puede construir y utilizar dicha funcionalidad, y se pueden construir tantos paquetes como se desee en un espacio de trabajo. [28]

Su creación se puede realizar en *C* o en *Python*, su estructura en *Python* es la siguiente:

- *my_package/*. Es la carpeta donde se crean y definen todos los recursos que necesite la funcionalidad, incluyendo nodos, archivos de lanzamiento o diferentes carpetas.
- *package.xml*. Archivo en el que se guarda su metainformación.
- *setup.cfg*. Identificador del paquete para su ejecución.
- *setup.py*. Archivo en el que se introducen las instrucciones de compilación del paquete.

Además, se pueden crear paquetes destinados a contener uno o varios archivos de lanzamiento, de forma independiente de los demás paquetes. [28]

Por último, el comando del código 3.1 es el utilizado para la compilación y construcción de los paquetes en ROS2, independientemente del lenguaje seleccionado para construirlo.

código 3.1: Comando de compilación y construcción de un paquete en ROS2

```
1 colcon build
```

3.2. Migración de ROS a ROS2

ROS es la versión original del middleware utilizado en este proyecto y la transformación que sufrió en el año 2017 a ROS2 fue el resultado de la continua evolución que ha sufrido la robótica en los últimos años, así como de todos los desafíos que se han presentado con este campo. Dicha transición supuso un cambio notable para toda la comunidad de desarrolladores ya que presentó una serie de mejoras y beneficios a nivel técnico y de accesibilidad para todos ellos. [17]

Entre estas mejoras se encuentran:

- Compatibilidad.

  a) Sistemas operativos. ROS era únicamente compatible para versiones de *Ubuntu*, y, ROS2, en cambio, cuenta con soporte para *Ubuntu*, *OS* y *Windows*. Como consecuencia, mejoró su escalabilidad y sintonía con diferentes soluciones *hardware* o *software*. [17]

b) Lenguajes de programación. ROS2 tiene la capacidad de trabajar con versiones más actualizadas ya que ROS solo soportaba algunas de estas como *Python* 2.7 o *C++* 11. Además, trabaja con muchos otros como *Rust* o *JavaScript*. Además, ROS2 no entiende de idiomas ni lenguajes de programación en cuanto a nomenclaturas o paquetes. [42]

- Metodología. La antigua versión de ROS tenía un nodo denominado *Master* que actuaba de servidor centralizado, limitando así al sistema. En cambio, ROS2 elimina este componente otorgando a cada uno de los nodos la misma responsabilidad, convirtiéndolo así en un sistema distribuido. Se puede observar su evolución en la figura 3.4 [17].

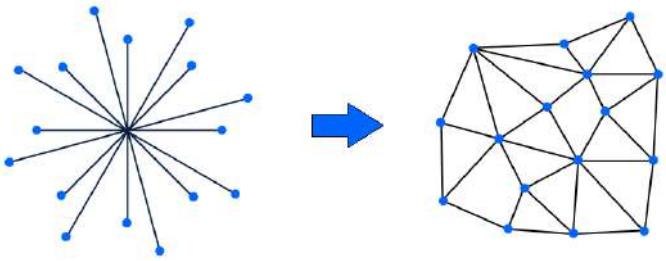


Figura 3.4: Sistema centralizado VS Sistema distribuido

- Comunicaciones internas. ROS utilizaba una capa de transporte apoyada por el protocolo TCP/IP generando ciertas restricciones. ROS2, en su lugar, optó por un sistema basado en DDS gracias a la arquitectura descrita en el apartado 3.1.3, aportando así mayor flexibilidad y la incorporación de programación en tiempo real. [17]
- Compilación. La configuración del entorno, paquetes y compilación del middleware cambió drásticamente:

  a) Paquete de compilación. Originalmente, ROS usaba un paquete denominado *catkin_simple*, el cual no conseguía hacer un completo uso del compilador CMake. En contraste, ROS2, mejoró esta problemática gracias a una reestructuración de la API de *CMake* con *colcon*.

  b) Independencia. ROS2 presenta independencia a la hora de compilar paquetes llevándolo a cabo de forma aislada, ya que en ROS era posible compilar varios en un único espacio de *CMake* dando lugar a colisiones. [42]

3.3. Sistema de referencia

El sistema de referencia de un robot es un sistema de coordenadas que permite posicionar el mismo en una posición tridimensional específica respecto a su origen. Este marco de referencia es utilizado para definir la posición y orientación de los componentes del robot y los objetos con los que interactúa. [24]

La tendencia hasta ahora ha sido trabajar con ángulos de Euler como suele ser definida la orientación de muchos robots por defecto y, en este caso, como la del robot UR3e. Estos consisten en trabajar con grados o radianes respecto de un eje cualquiera para alcanzar una determinada rotación. Pero, tienen un límite, y es que al trabajar con tres ejes simultáneamente, siempre habrá uno de los tres ejes que quede restringido para dar libertad de

rotación a los otros dos ejes. Es por esto, que a día de hoy, en robótica se utiliza un sistema de representación para la orientación, conocido como cuaternio. Los cuaternios se basan en trabajar con dichas rotaciones con una matriz de 4x4 dimensiones, de forma que aún teniendo un eje limitado, se pueden mover los otros tres que son los necesarios. [12]

La forma de un cuaternio es

$$q = q0 + q1i + q2j + q3k$$

Donde:

- q0, q1, q2, q3 corresponden a la parte real.
- i, j, k corresponden a la parte imaginaria.

3.3.1. Sistema de referencia del modelo UR3e

El sistema de referencia predeterminado por el fabricante del robot UR3e viene representado por la posición de la herramienta respecto de la base en milímetros y la orientación mostrada como un vector de rotación en radianes donde el propio vector en sí, facilita dicho eje sobre el que debe rotar. En cambio, se ha hecho una conversión de Euler a cuaternios, usando este último para establecer la orientación de la pinza para coger y depositar los objetos. Y, para la posición, se ha utilizado un punto tridimensional mediante coordenadas absolutas al igual que viene por defecto. [8]

3.4. Robótica manipulativa

La robótica es una rama multidisciplinar de la ingeniería enfocada en el diseño, construcción, programación y puesta en marcha de robots. Abarca conocimientos muy amplios de campos como la electrónica, mecánica, ingeniería de control, programación o diseño industrial. En concreto, la robótica manipulativa está centrada en tareas que incluyen el control de objetos en un entorno determinado. [43]

3.4.1. Conceptos teóricos

Se van a explicar diferentes conceptos del campo de la robótica necesarios para entender este proyecto. En concreto va a orientarse hacia brazos robóticos, ya que el UR3e, objeto de estudio, es uno de estos.

En primer lugar, un brazo robótico es un dispositivo o máquina que cuenta con un conjunto de articulaciones rotatorias que les permite realizar movimientos en cualquier dirección. [44]

Una vez conocida su definición, se van a abordar diferentes conceptos:

- Cada movimiento independiente que una articulación puede llevar a cabo, se considera como grado de libertad. Un robot tendrá tantos grados de libertad como número de movimientos libres pueda realizar. En el caso del robot UR3e, este cuenta con seis grados de libertad como se podrá verificar posteriormente. [41]
- Una vez introducido el concepto de grado de libertad, gracias a estos se pueden realizar dos tipos de movimientos: rotatorios o de traslación. Los movimientos de traslación se refieren a desplazamientos lineales a lo largo de un eje como es el

ejemplo de los movimientos del robot cartesiano mencionado, y los rotatorios son aquellos característicos de los brazos antropomórficos como el UR3e. [41]

- Por otro lado, es importante saber que para adquirir el control de cualquier robot, es necesario conocer su cinemática directa e inversa, ya que resultan fundamentales para la planificación de sus trayectorias. La cinemática directa se centra en la relación que existe entre la posición y orientación de las articulaciones con el extremo del robot. En contraste, la cinemática inversa dicta la configuración de las articulaciones para una posición y orientación determinada del extremo del robot. Además, en la imágen 3.5 se ilustra el objetivo de cada una. [24]

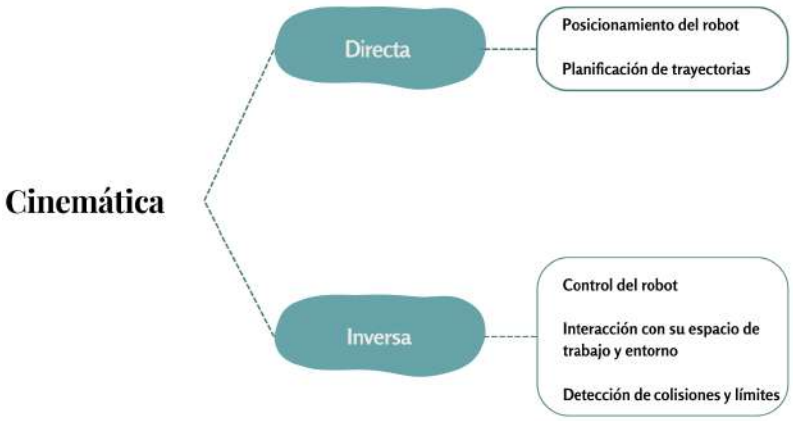


Figura 3.5: Objetivos de la cinemática

3.4.2. Tipos de configuraciones

A continuación, se van a presentar los tipos de configuración robóticas más comunes para la manipulación de objetos:

- Robots cartesianos. Este tipo de configuración se caracteriza por su movimiento lineal a lo largo de los tres ejes cartesianos: x, y, z. La capacidad de moverse a lo largo de los ejes es posible gracias a su diseño modular, que consta de articulaciones y actuadores lineales. Se puede observar en la imágen 3.6.
- Robots SCARA. Son brazos robóticos de cuatro grados de libertad que ofrecen una precisión y velocidad mayor que los cartesianos. Se caracteriza por tener un tamaño reducido diseñado para realizar tareas repetitivas como el ensamblaje o aplicaciones de Pick and Place. Su configuración se puede observar en la imágen 3.7. [15]
- Robots delta. Este es un robot paralelo a diferencia de los anteriores, que cuenta con hasta cinco grados de libertad en su configuración. Se caracteriza por tener comunicado su cuerpo con su herramienta mediante bielas con forma paralelas. Se puede visualizar en la imágen 3.9. [14]
- Robots colaborativos. Esta configuración es la mencionada en el apartado 2.3 y es el objeto de estudio de este proyecto.

[8] https://www.infoplc.net/
[9] https://vistazoalfuturo.com/
[10] https://www.br-automation.com/en/products/machine-centric-robotics/
[11] https://industrial.omron.es/es/products/collaborative-robots

Figura 3.6: Robot cartesiano[8]

Figura 3.7: Robot SCARA de Mitshubishi [9]

Figura 3.8: Robot delta de ABB[10]

Figura 3.9: Robot colaborativo de Omron[11]

3.5. Sistemas de visión en robótica colaborativa

A continuación, se detallan varios tipos de sistemas de visión usados en aplicaciones con robótica manipulativa para dar soporte y autonomía a la misma. En concreto, se van a introducir diferentes tipos de cámaras y sensores orientados a su desarrollo dentro de la robótica, así como ciertos conceptos necesarios para comprender el sistema de visión implementado en este proyecto.

- Cámaras 2D. En primer lugar, las cámaras bidimensionales son dispositivos cuya finalidad es tomar imágenes para poder ser procesadas y analizadas posteriormente, por algún sistema de visión artificial. Tienen ciertas limitaciones debido a la iluminación ya que una falta o exceso de luz puede dañar el reconocimiento de figuras de la imágen. No obstante, su combinación con librerías de *software* de visión hacen que sean muy populares en aplicaciones como reconocimiento, seguimiento y detección de objetos. Además, son útiles en lecturas de códigos de barras y mediciones. [10]
- Cámaras 3D. Este tipo de cámaras son una mejora del caso anterior presentando una tecnología similar con la diferencia de su capacidad para registrar la profundidad. Gracias a ello, son capaces de ubicar un objeto tridimensionalmente siendo muy comunes en aplicaciones donde se necesite una buena gestión del espacio. Es por esto que permiten a los robots poseer una percepción más precisa de la situación espacial de su entorno. Un ejemplo de este tipo de cámaras son las estereoscópicas, explicadas en el apartado 3.5.1. [10]
- Cámaras basadas en tiempo de vuelo. Esta alternativa consiste en el uso de luz infrarroja para obtener la distancia entre la cámara y el objeto. Ofrecen imágenes en

escalas monocromáticas con información tridimensional a altas velocidades en comparación con sistemas tradicionales. Sin embargo, uno de sus inconvenientes es su baja resolución por lo que sus aplicaciones se suelen limitar a mediciones y comprensión de entornos. A pesar de que se consideran cámaras 3D, en muchas ocasiones se suele combinar con imágenes a color en alta resolución para solventar dicha limitación. [46]

- Por último cabe mencionar que existen numerosos sistemas de visión en la actualidad como: sistemas LiDAR, que utilizan luz láser para crear mapas tridimensionales; cámaras térmicas y sensores de todo tipo.

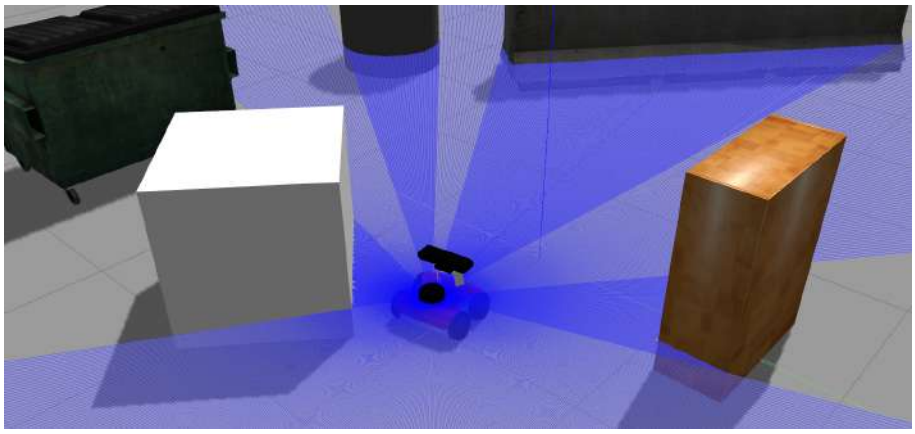


Figura 3.10: Robot móvil con sensor LiDAR incorporado en *Gazebo* [4]

3.5.1. Visión estereoscópica

Se va a explicar el funcionamiento de la visión estéreo debido a que es una de las técnicas utilizadas en el desarrollo de este proyecto, siendo una de las técnicas más interesantes para el reconocimiento y ubicación de objetos. La visión estereoscópica hace referencia a la captura de imágenes estéreo con el uso de dos lentes separadas a una distancia fija entre sí, usando su combinación para recrear información tridimensional. A continuación se va a introducir el concepto de disparidad, el cual es muy popular dentro del mundo de la visión artificial, debido a su importancia para percibir la profundidad. La disparidad se conoce como la distancia fija que hay entre dos puntos vistos desde dos imágenes diferentes generadas por cada una de las cámaras, situadas normalmente en dos extremos. Se puede observar su metodología en la imágen de la figura 3.11. [27]

3.5.2. Visión RGB

Por último, se va a comentar de forma breve el funcionamiento en cuánto a visión con cámaras *RGB*[6]. Estas cámaras son consideradas bidimensionales ya que las imágenes que capturan son en dimensiones de: ancho y alto. Es posible pensar que gracias a su detección de colores se consideran tridimensionales, pero no es así, dado que no es posible estimar su profundidad. En consecuencia, se combinan con cámaras que tengan la capacidad de

[4] https://discourse.ros.org/
[5] https://docs.luxonis.com/

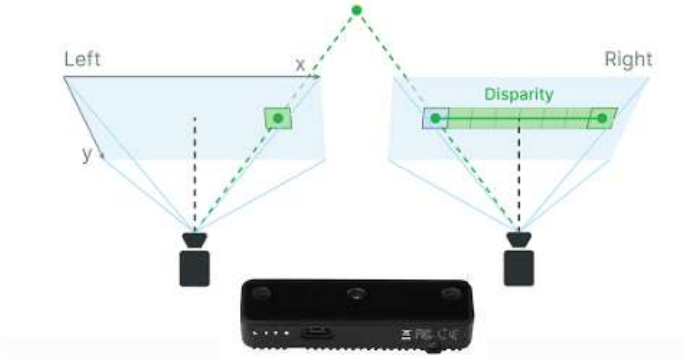


Figura 3.11: Concepto de disparidad[5]

estimar profundidad, como las estereoscópicas, siendo esta, combinación implementada en el proyecto. [10]

[6]red-green-blue

Capítulo 4

Diseño del proyecto

4.1. Recogida e identificación de los requisitos

En cuánto a la visión del proyecto, el propósito fundamental es llevar a cabo el movimiento del robot colaborativo UR3e mediante un sistema de visión artificial y un gripper como efector final, terminando en el desarrollo de diferentes aplicaciones de Pick and Place. La interacción del usuario será de muy alto nivel, administrando este una serie de indicaciones simples, y posteriormente el robot será capaz de desarrollar el procedimiento mencionado de forma autónoma.

En relación a las especificaciones complementarias del proyecto, el enfoque principal es llevar a cabo este propósito mediante el uso de ROS2 como sistema de comunicación para el control de las aplicaciones y el uso de *Python* como lenguaje base de programación. Además, las trayectorias seguidas por el robot serán limitadas a un espacio de trabajo fijo y las aplicaciones serán ejecutadas con objetos previamente definidos ubicados dentro del espacio de trabajo.

Los componentes principales de este proyecto se pueden observar en el glosario de la figura 4.1.

Cámara	Dispositivo de visión que identifica los objetos y le devuelve al robot su posición en tres dimensiones.
Espacio de trabajo	Área cerrada en el cual el robot se mueve para realizar las tareas específicas.
Gripper	Herramienta encargada de la tarea final de las aplicaciones de Pick and Place.
Ordenador	Dispositivo donde se lanza, visualiza el funcionamineto completo. La interacción del mismo se incluye en este.
Robot UR3e	Robot colaborativo movido por el entorno de trabajo.
Usuario	Persona que pone en marcha el funcionamiento e interacciona seleccionando la tarea a realizar.
Objetos	Materiales físicos manipulados por el robot.

Tabla 4.1: Glosario

Con el objetivo de obtener una identificación clara y bien definida de los requisitos, se ilustra la tabla de casos de uso en la figura 4.2.

A continuación, en la figura 4.1, se ilustra el diagrama de casos de uso, y, por lo tanto, las funcionalidades que se van a implementar en este proyecto. Se han creado tres aplicaciones de Pick and Place que validan todos los requisitos recogidos en este apartado.

Actor	Usuario
Descripción	Realizar diferentes tipos de aplicaciones de Pick and Place mediante el movimiento y uso de la herramienta del robot colaborativo UR3e, mediante visión por computador para reconocer y ubicar los objetos dentro del espacio de trabajo.
Precondiciones	El usuario conoce el funcionamiento del robot, cámara e indicaciones a lanzar en la terminal. El usuario entiende el funcionamiento de la interfaz para interaccionar con esta.
Postcondiciones	El usuario termina la aplicación actual manualmente o vuelve a lanzar el funcionamiento de otro caso de uso.
Flujo normal	1.0 El usuario conecta y enciende todos los dispositivos. 2.0 Lanza el archivo de lanzamiento del proyecto en la terminal del ordenador y ejecuta el programa de la tablet del robot. 3.0 El usuario interactúa con la interfaz seleccionando tipo de aplicación y secuencia de orden de objetos. 4.0 La cámara obtiene las posiciones de los objetos seleccionados del espacio de trabajo por orden. 5.0 El robot se mueve a dichas posiciones y recoge los objetos. 6.0 El robot deposita los objetos en ubicaciones determinadas según el caso de uso seleccionado por el usuario. 7.0 El usuario acaba el programa por terminal o por funcionalidad.
Flujo alternativo	5.1 La posición de la cámara no es suficiente precisa y el robot no logra alcanzar los objetos terminando el programa. 5.2 La posición del objeto está fuera del espacio de trabajo y el robot no puede alcanzarla terminando el programa.
Excepciones	1.0.E.1 El usuario introduce erróneamente el archivo de lanzamiento. 2.0.E.1 Se lleva a cabo un uso incorrecto de la interfaz de usuario terminando el programa 2.0.E.2 La conexión de algún dispositivo es incorrecta. 4.0.E.1 El robot planifica de forma incorrecta la trayectoria a ejecutar y colisiona con el espacio de trabajo.

Tabla 4.2: Tabla de casos de uso

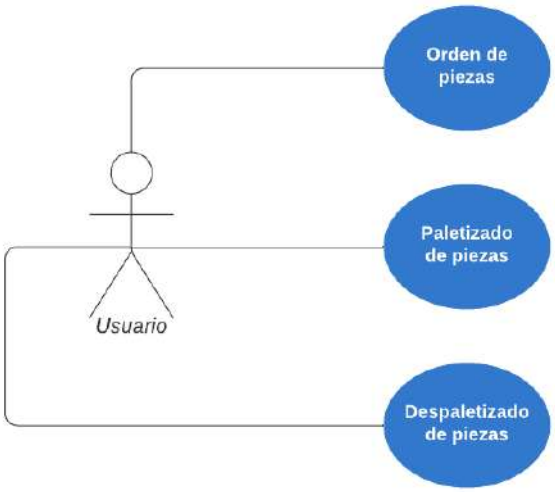


Figura 4.1: Diagrama de casos de uso

4.2. Diseño físico

4.2.1. UR3E

El modelo UR3e es un robot industrial colaborativo que fue lanzado al mercado por Universal Robots en el año 2018. Este robot antropomórfico de seis grados de libertad fue diseñado para trabajar con una carga máxima de 3 kilogramos y realizar diversas aplicaciones como la automatización de procesos industriales o la robótica educativa dentro del campo de la investigación. La información recopilada para la descripción técnica del robot ha sido recogido del manual de usuario proporcionado por Universal Robots. [8]

Figura 4.2: Morfología del UR3e [1]

4.2.1.1. Especificaciones técnicas

El robot UR3e ha sido diseñado para trabajar con pequeñas cargas siendo el más ligero de la gama UR pudiendo realizar movimientos muy rápidos de alta precisión. Es ideal para todo tipo de espacios de trabajo y entornos industriales o de cualquier otro servicio. Se pueden observar sus especificaciones técnicas en las figuras 4.3 y 4.4.

Consumo promedio máximo	300 W	
Consumo típico en régimen de operación moderado	100 W	
Seguridad	17 funciones de seguridad configurables	
Certificaciones	EN ISO 13849-1, PLd Categoría 3, y EN ISO 10218-1	
Sensor de fuerza, brida de la herramienta	Fuerza, x-y-z	Par, x-y-z
Rango	30,0 N	10,0 Nm
Resolución	2,0 N	0,1 Nm
Precisión	3,5 N	0,1 Nm
Carga útil	3 kg (6,6 lbs)	
Alcance	500 mm (19,7 in)	
Grados de libertad	6 articulaciones giratorias	
Programación	Pantalla táctil de 12" con interfaz gráfica de usuario Polyscope	

Figura 4.3: Especificaciones y rendimiento del UR3e[1]

Repetibilidad según la norma ISO 9283	± 0,03 mm
Materiales	Aluminio, plástico, acero
Velocidad TCP estándar	1 m/s (39,4 in/s)
Long. cable del brazo robótico	6 m (236 in) cable incluido. 12 m (472 in) y opciones de alta flexibilidad disponibles.
Peso con cable	11,2 kg (24,7 lbs)
Ruido	Menos de 60 dB(A)
Velocidad TCP estándar	1 m/s (39,4 in/s)

Figura 4.4: Funciones y características físicas [1]

[1] www.universal-robots.com

4.2.1.2. Diseño mecánico

En cuánto a la estructura física del robot, está dividido en tres partes principales: una base, el propio brazo y una extremidad final para la herramienta seleccionada en función de la aplicación a desarrollar:

- **Base.** Es la parte fija del espacio de trabajo, tiene forma cilíndrica y está construido de aluminio fundido. La base contiene tanto controladores como motores, siendo la encargada de tener el controlador principal que coordina los movimientos de los demás ejes del robot.
- **Brazo.** Cuenta con seis eslabones compuestos por una carcasa de plástico conectados entre sí mediante juntas articuladas. Los motores ubicados en la base transmiten el movimiento por medio de los engranajes.
- **Extremo.** Dispone de una placa de montaje para la herramienta a conectar y de un conector para llevar a cabo la conexión eléctrica entre el robot y la herramienta.

4.2.1.3. Hardware

- **Caja de control.** La caja de control del robot es una de las partes fundamentales para el correcto funcionamiento del mismo. Incluye una fuente de alimentación de 100-240 V y un controlador que coordina y controla los movimientos del robot. Por otra parte, dispone adicionalmente de una serie de puertos de entrada y salida utilizados para conectar dispositivos externos o sensores. De igual modo, contiene una serie de puertos y conectores de comunicación para conectar el robot a otros sistemas de control como por ejemplo, un ordenador a través del puerto Ethernet. Se puede visualizar en la figura 4.5.

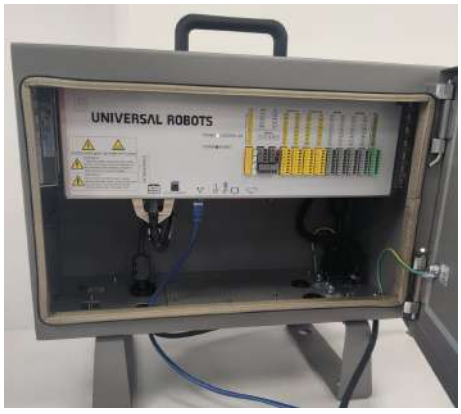


Figura 4.5: Caja de control del UR3e

- **Sensores y actuadores.** Dispone de una gran variedad de sensores incluyendo de proximidad, fuerza y temperatura entre todos estos. En cuánto a actuadores, utiliza motores de corriente continua sin escobillas principalmente controlados mediante los microcontroladores descritos.
- **Microcontroladores.** Tiene un ARM de 64 bits como microcontrolador principal para cubrir las funciones más importantes y está equipado con otros específicos para los motores y sensores mencionados.

4.2.1.4. Software

Los robots de la gama UR cuentan con una consola de programación diseñada para poder llevar a cabo el control y la programación del robot de forma sencilla y eficiente con *PolyScope*, una librería de visulización gráfica. Esta consola observable en la figura 4.6, permite diseñar tareas con cualquier movimiento programado, así como configurar los parámetros necesarios tanto del espacio de trabajo como del robot o restricciones.

Figura 4.6: Consola de programación de UR [1]

4.2.2. Herramienta HRC-03

La herramienta seleccionada para la tarea de Pick and Place de este proyecto ha sido una pinza con modelo HRC-03. El efector final de un brazo antropomórfico tiene un papel importante en la tarea a realizar, ya que se encarga de manipular los objetos dentro del espacio de trabajo. Existen cientos de tipos de pinzas en función de los requisitos del proyecto. [45]

La herramienta HRC-03 es una pinza paralela con accionamiento eléctrico que está compuesta de dos mandíbulas, diseñada para aplicaciones con robots colaborativos. Esta pinza posee una fuerza de agarre de hasta 190 N y de una carrera de 10 mm con una flexibilidad de detección de la pieza durante toda la carrera. Además, dispone de tecnología de transmisiones sin escobillas y diferentes modos de parada o emergencia. [18]

Figura 4.7: Gripper HRC-03 [2]

4.2.3. Cámara OAK-D Lite

El modelo OAK-D Lite-AF de la gama OAK es una versión mejorada de las anteriores con un reducido tamaño y una gran variedad de aplicaciones robóticas para la detección o reconocimiento de cualquier objeto, persona o cosa. La información recogida del modelo ha sido recopilada de la documentación [27].

Además, dado su tamaño y su rápida conexión fue diseñada para trabajar en cualquier espacio de trabajo con una amplia variedad de herramientas disponibles. Asimismo, esta versión incluye enfoque automático lo que hace de esta un modelo preciso y cómodo de usar.

La cámara tiene incorporada una *CPU* capaz de llevar a cabo el procesamiento tanto de la inteligencia artificial como del procesamiento de las imágenes o vídeos. Esto conlleva a que la lógica ocurra dentro de la *CPU* y el ordenador sirve únicamente para mostrar el resultado. Este núcleo se denomina *Robotics Vision Core 2"*.

La cámara OAK-D Lite cuenta con tres cámaras en su configuración: una cámara rgb de alta resolución en el centro y dos cámaras de tipo estéreo en sus extremos.

Figura 4.8: Cámara OAK-D LITE [3]

- **Cámara RGB.** Está destinada a la distinción de colores en dos dimensiones. Cuenta con una resolución de 13 megapíxeles y un sensor específico para proporcionar una alta calidad.
- **Cámaras estéreo.** Se usan para percibir la realidad en tres dimensiones. Dispone de 120 *FPS*, es decir, tiene la capacidad de reproducir 120 imágenes por segundo. Además, son comunes para medir profundidades y distancias concretas.

Se puede observar la tabla de características técnicas de ambos tipos de cámaras en la figura 4.9.

Por último, se puede observar el compacto diseño de la cámara en la figura 4.10.

[2] https://www.directindustry.es/prod/zimmer-group/product-14534-2222425.html
[3] https://docs.luxonis.com/en/latest/

Camera Specs	Color camera	Stereo pair
Sensor	IMX214 (PY014 AF, PY114 FF)	OV7251 (PY013)
DFOV / HFOV / VFOV	81° / 69° / 54°	86° / 73° / 58°
Resolution	13MP (4208x3120)	480P (640x480)
Focus	AF: 8cm - ∞ **OR** FF: 50cm - ∞	Fixed-Focus 6.5cm - ∞
Max Framerate	35 FPS	120 FPS
F-number	2.2 ± 5%	2.0 ± 5%
Lens size	1/3.1 inch	1/7 inch
Effective Focal Length	3.37mm	1.3mm
Distortion	< 1%	< 1.5%
Pixel size	1.12µm x 1.12µm	3µm x 3µm

Figura 4.9: Tabla de características OAK-D LITE [3]

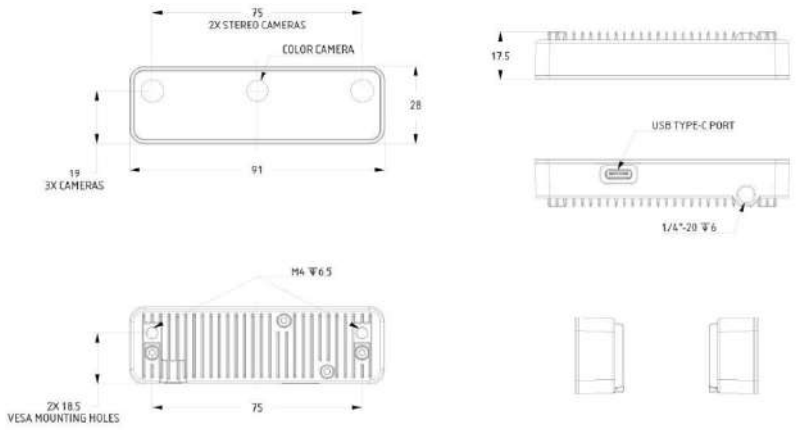


Figura 4.10: Dimensiones OAK-D LITE [3]

4.2.3.1. Estructura de la cámara

El enfoque principal de la visión artificial es identificar la posición de los objetos con una vista panorámica del espacio de trabajo. Es por esto, que se ha colocado la cámara a una distancia considerable por encima del espacio de trabajo, diseñando una estructura que lo permitiese. Además, su ubicación facilita la conversión del sistema de referencia entre la cámara y el robot.

La solución adoptada en este proyecto ha sido la implementación de una estructura física que combina: un trípode, una madera con un perfil en T, que sirve de brazo hasta la cámara, y una pieza que actúa de adaptador entre ambos componentes. La pieza de madera fue diseñada y cortada con una cortadora láser, incorporando un perfil en T, que mejora su robustez y evita su flexión. Adicionalmente, la pieza de conexión se modeló e imprimió con una impresora 3D.

4.2.4. Área de trabajo

Una vez realizada una detallada descripción sobre todos los componentes físicos del proyecto, se va a presentar el área en el cuál se recogerán y depositarán las piezas. El diseño final con todos los elementos integrados está disponible en la figura 4.11, pudiendo observar el gran área de trabajo disponible. Y, finalmente, el área reservado para las

manipulación de piezas es el reflejado en la figura 4.12.

Figura 4.11: Modelo físico completo

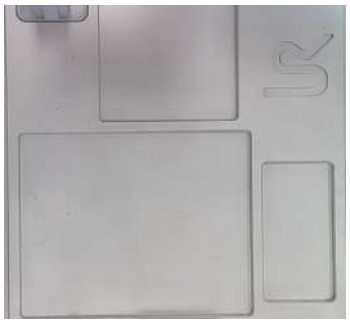


Figura 4.12: Área de objetos

4.2.5. Conexión de componentes

Por último, se va a mencionar como se han llevado a cabo las conexiones físicas entre todos los componentes existentes. El ordenador, cuyo papel consiste en ser el centro de control del proyecto, se conecta de forma material tanto con la caja de control del robot

como con la propia cámara. Por otro lado, la caja de control está conectada al propio robot, siendo su alimentación y controlador principal, así como con la propia tablet de este. En el esquema de la figura 4.13 se ha ilustrado el tipo de conexiones, así como el material utilizado para estas. Por último, cabe destacar que se mencionan sólo materiales tangibles, ya que la mayoría de conexiones del proyecto son en relación al *software* que se menciona en el siguiente apartado.

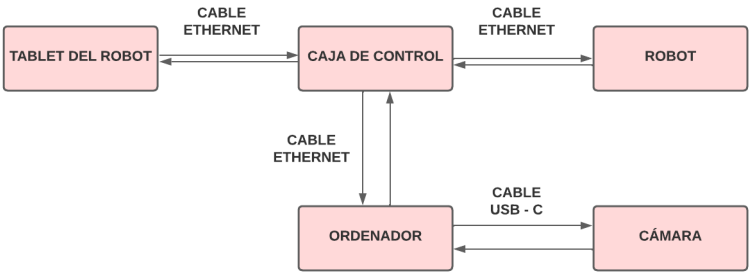


Figura 4.13: Conexiones entre componentes

4.3. Diseño del Software

4.3.1. Tecnologías empleadas

En este proyecto se ha utilizado ROS2, como sistema de comunicación principal, debido a su gran interoperabilidad. Se ha seleccionado este *middleware* por su rendimiento en tiempo real, por su soporte para diferentes lenguajes y por su escalabilidad. El objetivo es poder mejorar la versión creada en un futuro o adaptarla para otro robot con características similares. [38]

Además, a pesar de que ROS2 no hace distinción de lenguajes, el proyecto se ha implementado íntegramente en *Python* debido a que es uno de los requisitos iniciales del mismo por la productividad y facilidad de algoritmia que presenta.

Por último, todo el *software* se ha desarrollado en *Ubuntu* 20.04 como sistema operativo característico, ya que ROS2, fue diseñado específicamente para las versiones de *Ubuntu*, así como por su amplia gama de librerías y herramientas disponibles.

Figura 4.14: Herramientas de software utilizadas [4]

[4] `https://github.com/ros2`, `https://javierin.com`, `www.devacademy.es`

4.3.2. Planteamiento

En este punto del diseño, se va a explicar el planteamiento seguido para integrar todas las tecnologías empleadas mencionadas en el apartado 4.3.1, con el objetivo de comprender la forma de comunicación entre los distintos componentes del proyecto.

Se han creado varios nodos de ROS2, considerados como entidades individuales, de forma que cada uno se encarga de una o varias tareas esenciales. Cada nodo ha sido programado en *Python* como se mencionó inicialmente, y representa una funcionalidad relacionada con una de las tres secciones en las que se ha dividido su implementación: control del robot, visión artificial e interfaz gráfica. La comunicación entre estas entidades se ha realizado mediante diferentes *topics* que hacen posible la integración de todos ellos formando la implementación final.

En dicha implementación se han creado, por un lado, nodos propios orientados a las aplicaciones de Pick and Place, para su funcionamiento y comunicación, y por otro lado, se han usado y modificado ciertos componentes existentes proporcionados por los controladores de UR explicados en el apartado 5.3.2.

A continuación, en la tabla 4.3 se muestran los componentes propios y externos modificados. Y, asimismo, el resumen del rol de cada nodo y *topic* se ha recopilado en la tabla 4.4.

	Nodos	Topics
Propios	*Camera* *ObjectDetector* *Control_robot_master* *interfaz_menu*	*/camera/rgb/image_raw* */camera/depth/image_raw* */object_detector* */sec_color* */resp_aplicacion*
Externos	*move_group_private* *Robot_state_publisher* *joint_state_broadcaster*	*/joint_states /collision_object /planning_scene_world*

Tabla 4.3: Componentes del proyecto

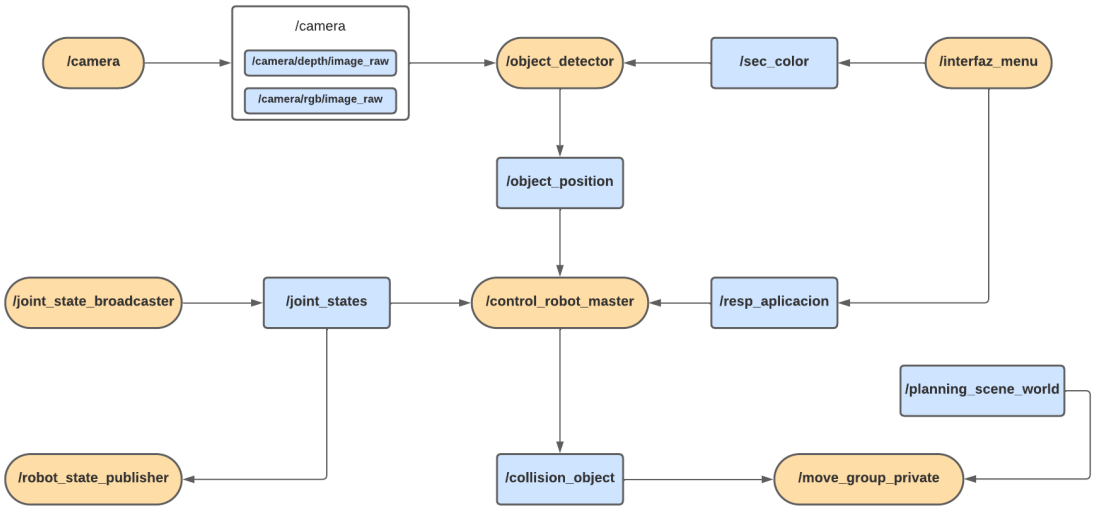


Figura 4.15: Diagrama de nodos

Nodos	Funcionamiento
Camera	Lanzador de las cámaras y encargado de crear y publicar las correspondientes imágenes a tratar por *ObjectDetector*.
ObjectDetector	Nodo encargado de analizar las imágenes en un plano tridimensional y de construir la posición del objeto u objetos a detectar. Esta posición se almacena en el topic denominado */object_position.*
Control_robot_master	Nodo que actúa de gestor del control de la planificación y ejecución de los movimientos del robot, así como del uso de la herramienta. Se suscribe a las posiciones del topic */object_position* y utiliza *Moveit* para su movimiento.
interfaz_menu	Nodo destinado a la creación de una interfaz gráfica de usuario con el objetivo de llevar a cabo la interacción y control del proyecto dentro de esta.
move_group_private	Es el responsable de la planificación y ejecución ordenada por el nodo *Control_robot_master*. Su funcionamiento se observa en el apartado 5.1.1.1 y es propiedad de Moveit.
Robot_state_publisher	Nodo complementario que genera la información del estado del robot en un *topic* llamado */joint_states*. Además, es común utilizar este para publicar las posiciones del robot como coordenadas.
joint_state_broadcaster	Nodo encargado de obtener la información actual del robot, similar y complementario al nodo *Robot_state_publisher*.

Topics	Funcionamiento
/camera/rgb/image_raw */camera/depth/image_raw*	Almacenan las imágenes en bruto en color y en escala de grises con formato ros para ser analizadas posteriormente.
/object_detector	Almacena la posición y orientación de los objetos como una estructura propia de ros denominada *"Pose"*.
/joint_states	Guarda el estado de las articulaciones del robot publicadas por el nodo *robot_state_publisher* mencionado.
/collision_object	Almacena posibles casos de colisiones con objetos que comunica a *move_group* para que planifique y ejecute trayectorias teniéndolos en cuenta.
/sec_color	Contiene la secuencia de colores que determina el orden de los objetos a recoger o depositar en las aplicaciones de Pick and Place.
/resp_aplicacion	Guarda la opción de Pick and Place a realizar introducida por el usuario en la interfaz gráfica.
/planning_scene_world	Contiene información necesaria para la representación virtual del entorno de planificación de *Move_group*.

Tabla 4.4: Resumen del funcionamiento de los nodos del proyecto

A modo de introducción de la idea seguida en el proyecto, estos componentes se van a utilizar según el siguiente procedimiento:

En primer lugar, existe un nodo de control principal encargado del control del robot y su herramienta, el cual se encargará de gestionar toda la información de su estado actual así como las instrucciones para llevar a cabo las aplicaciones desde una perspectiva técnica como a nivel de comunicación. En segundo lugar, se han creado dos nodos proporcionados por la cámara que resultan en la identificación y obtención de la posición de los objetos, con una posterior comunicación al control del robot. Y, por último, hay un nodo encargado de la interfaz gráfica para llevar a cabo la interacción con el usuario y gestionar la comunicación con los demás nodos del proyecto.

Finalmente, se muestra el diagrama de nodos del proyecto en la figura 4.15 con la ambición de poner en situación al lector del plano de funcionamiento descrito.

4.3.3. Diseño del control del robot

El diseño del control del robot se ha llevado a cabo, en primer lugar, haciendo uso de varios recursos que ofrecen los controladores de UR mencionados en el apartado 5.3.2, como la información actual del robot almacenada en el topic /joint_states y en /joint_trajectory_controller. Y, la parte importante, ha sido la creación del nodo de control, el cual coordina toda esta información y se encarga de controlar todo lo relativo al movimiento y uso de la herramienta.

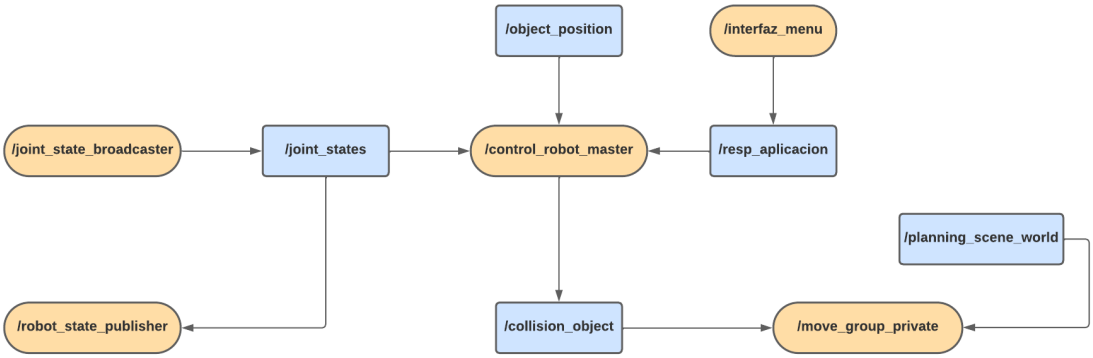


Figura 4.16: Diagrama de nodos de control

4.3.3.1. Nodo de control

El flujograma de la figura 4.17 representa el funcionamiento del nodo encargado de coordinar todos los componentes del proyecto para terminar en realizar las aplicaciones fijadas como objetivo.

4.3.4. Diseño de la visión por computador

En este apartado se mostrará el diseño de la visión por computador implementada en el proyecto con el objetivo de llevar a cabo la detección de objetos en tres dimensiones para poder comunicarsélo al nodo de control. La metodología de detección de objetos genérica incluye los siguientes pasos:

- Adquisición de imágenes.
- Preprocesamiento de las mismas.
- Segmentación de características.
- Detección y análisis de características.
- Salida de resultados.

Esta serie de instrucciones, recogidas del documento [47], es la que se ha seguido para el proceso de detección de objetos gracias a varias librerías que hacen de esto, un trabajo asequible. El diagrama de nodos acotado de la parte de visión se ha plasmado en la figura 4.18.

Además, el objetivo de la visión por computador diseñada es poder observar en tiempo real el comportamiento de la aplicación desde ambas imágenes producidas como se observa en la figura 4.19.

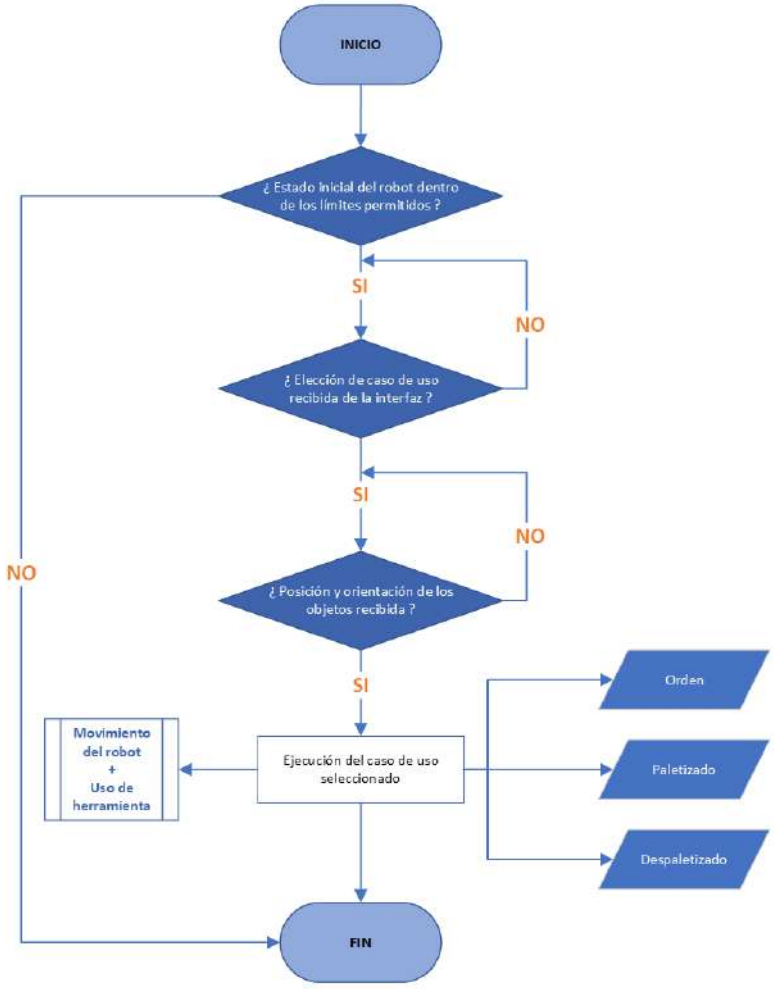


Figura 4.17: Flujograma del nodo de control

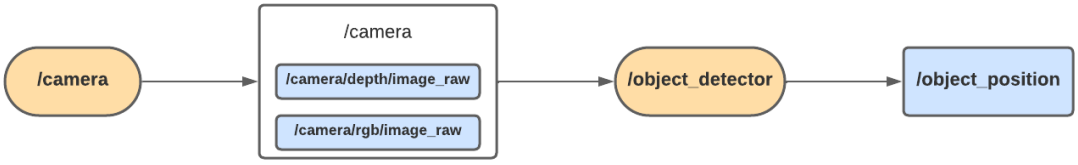


Figura 4.18: Diagrama de nodos de visión por computador

4.3.4.1. Nodo de procesamiento de imágenes

Se ha llevado a cabo el procesamiento de las imágenes tanto a color como en escala de grises utilizando la librería *DepthAI*, mencionada en el apartado 5.1.3.2, y la librería *OpenCV*, mencionada en el apartado 5.1.3.1. Esto se ha realizado dentro del primer nodo introducido en el apartado 4.3.2.

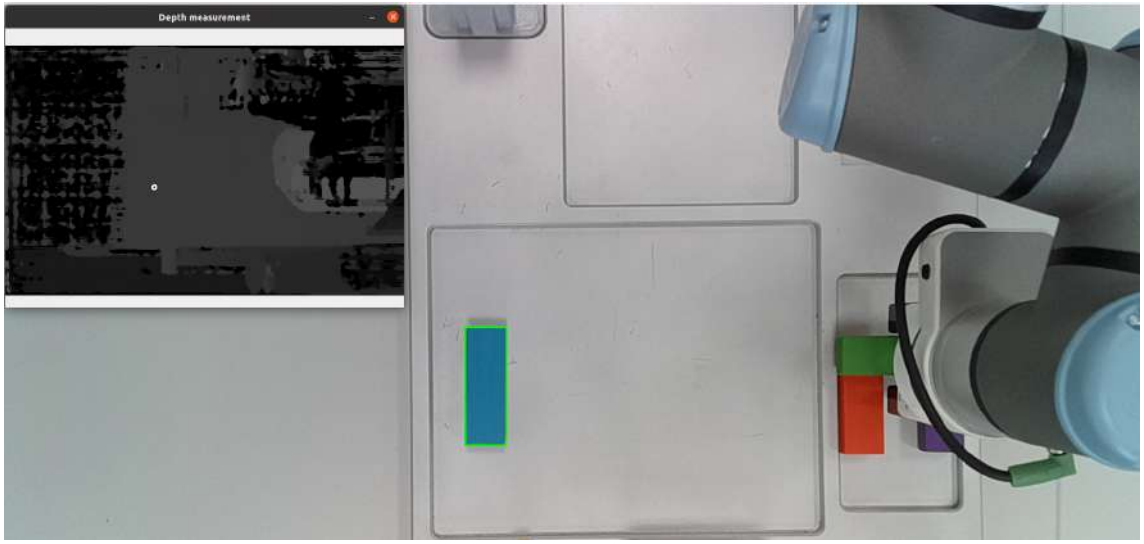


Figura 4.19: Detección completa de objetos mediante imágenes

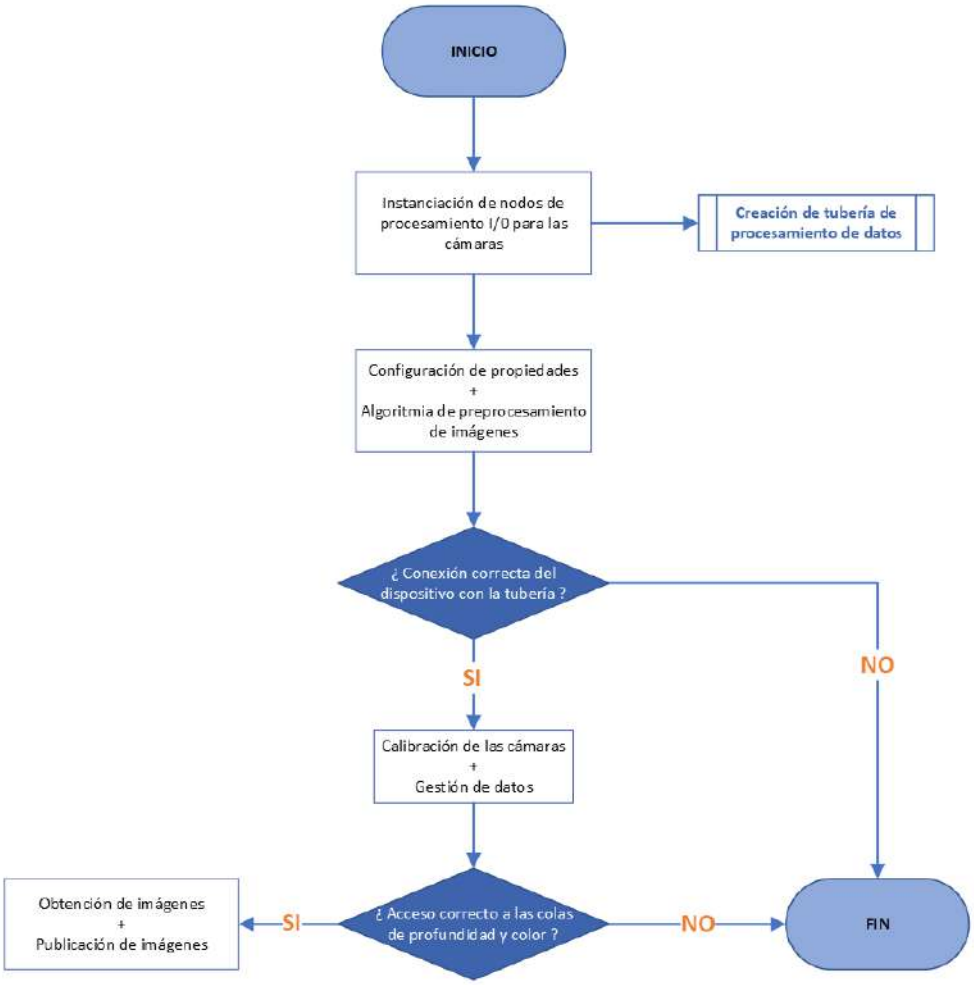


Figura 4.20: Flujograma del nodo de procesamiento de imágenes

4.3.4.2. Nodo de detección de objetos

La detección de formas y posiciones tridimensional, así como la interacción con el nodo de control y la interfaz, se ha llevado a cabo dentro del nodo *ObjectDetector*, cuyo funcionamiento ha sido reflejado en el flujograma de la figura 4.21

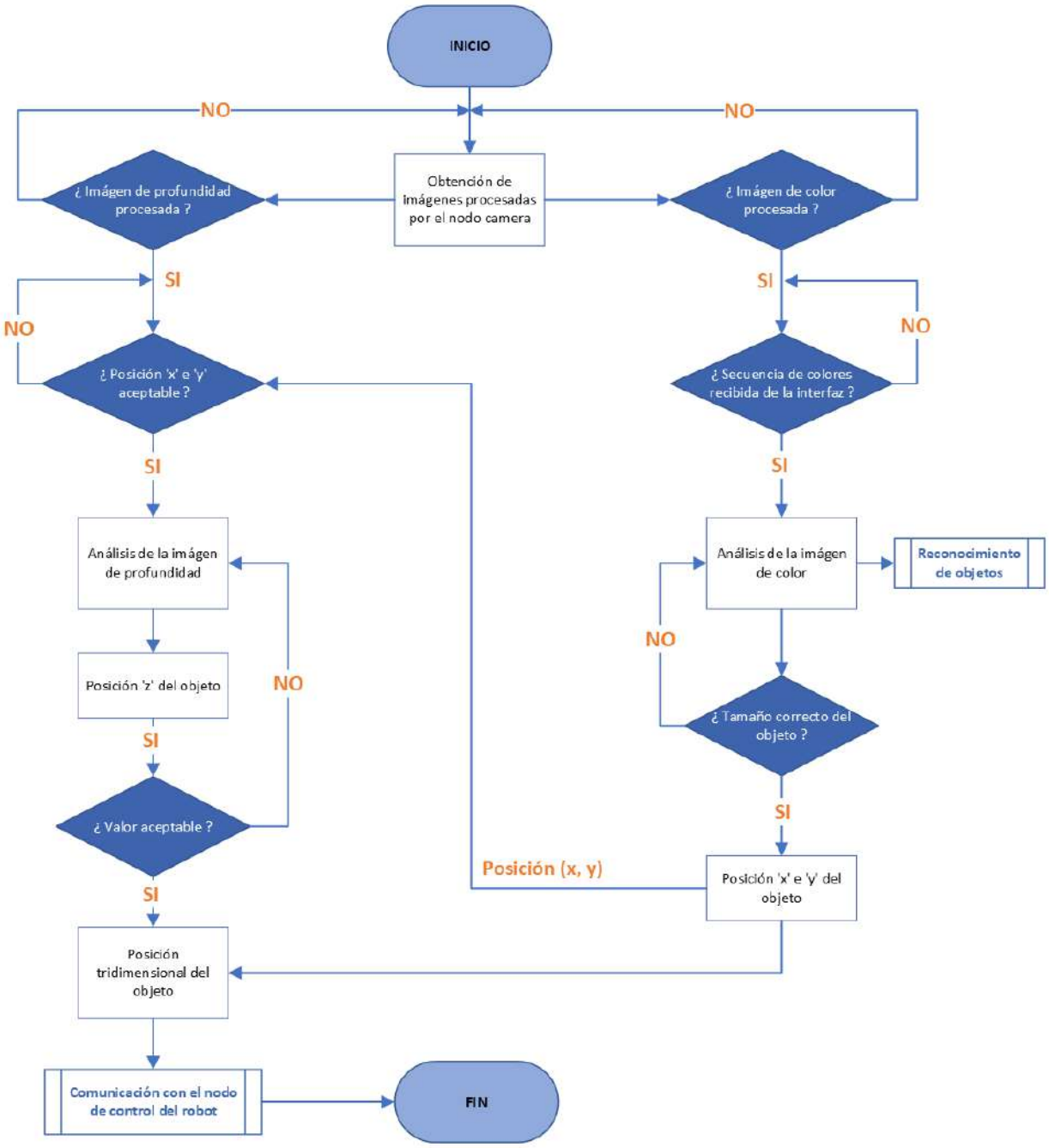


Figura 4.21: Flujograma del nodo de detección de objetos

4.3.5. Diseño de la interfaz gráfica de usuario

Se ha creado una interfaz de usuario con la ambición de conseguir una interacción con el usuario sencilla y atractiva. Su creación se ha llevado a cabo dentro del nodo denominado *interfaz_menu*. Esta ventana emergente observable en la figura 4.23, cuenta con una serie de indicaciones, botones e imágenes a visualizar. Su funcionamiento es el que se observa en el flujograma de la figura 4.22.Las opciones a elegir son:

- Orden de colores de los objetos a recoger. Por ejemplo, rojo, verde, morado, azul.
- Tipo de aplicación de Pick and Place. Por ejemplo, paletizado de piezas.

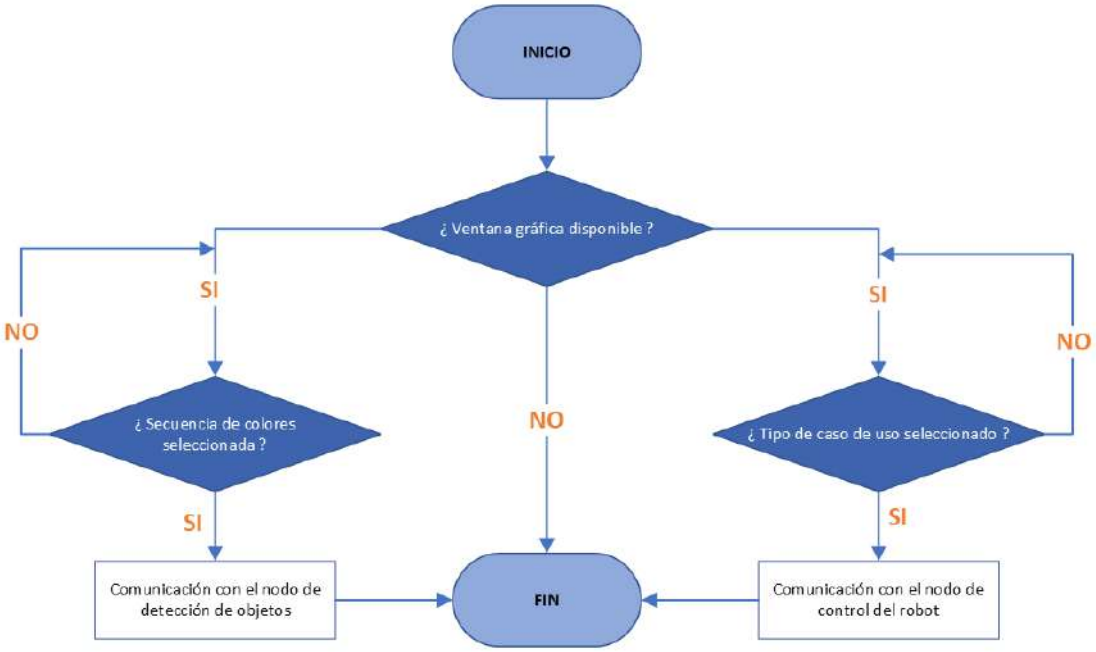


Figura 4.22: Flujo de funcionamiento de la interfaz gráfica de usuario

Figura 4.23: Aspecto visual de la interfaz de usuario

Capítulo 5

Desarrollo del proyecto

5.1. Herramientas utilizadas durante todo el proyecto

El movimiento del robot UR3e con el uso de ROS2 y la visión por computador implementada con la cámara, es un proceso complejo que requiere de diversas herramientas coordinadas entre sí. Su coordinación es necesaria para llevar a cabo su ejecución de forma correcta y eficiente. En esta sección se presentan el conjunto de herramientas más importantes que han sido seleccionadas para llevar a cabo este proyecto.

5.1.1. Moveit2

Moveit2 es una plataforma de planificación en aplicaciones robóticas creada por el equipo de desarrollo de ROS junto con su comunidad en el año 2011. Se creó con la idea de poder planificar de forma asequible el movimiento de estas aplicaciones incorporando grandes variedades de usos en la mayoría de áreas en robótica. La información técnica de este apartado ha sido recogida de la documentación oficial de *Moveit2*: [13].

Algunos de sus usos más frecuentes son:

- Planificación de movimientos: cualquier tipo de trayectoria con distintas limitaciones.
- Manipulación de objetos.
- Control y Navegación de las aplicaciones implementadas.
- Percepción 3D para comprender el entorno de trabajo en tres dimensiones mediante el uso de sensores.

En el año 2020, se lanzó *Moveit2*, una versión actualizada de *Moveit* para las futuras distribuciones de ROS2 convirtiéndose en una de las potentes herramientas más utilizadas por los desarrolladores de esta comunidad.

5.1.1.1. Interfaz propia de Moveit

La forma de trabajo de *Moveit* se realiza mediante una interfaz asociada a diferentes planificadores de movimiento, la cual es dirigida a través de un nodo administrado por ROS2 llamado *move_group*. *Move_group* es un nodo de ROS2 que incluye varias acciones

[1] https://moveit.ros.org/

Figura 5.1: Logo de Moveit [1]

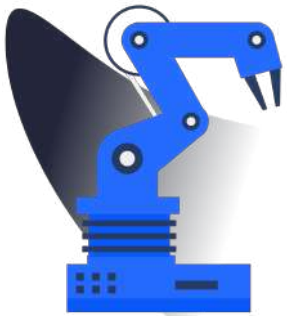


Figura 5.2: Figura representativa de moveit [1]

y servicios a los cuales se pueden acceder mediante *C++*, *Python* o interfaces gráficas. En este caso se ha llevado a cabo en *Python* con la ayuda del paquete *moveit_commander* recientemente actualizado. [13]

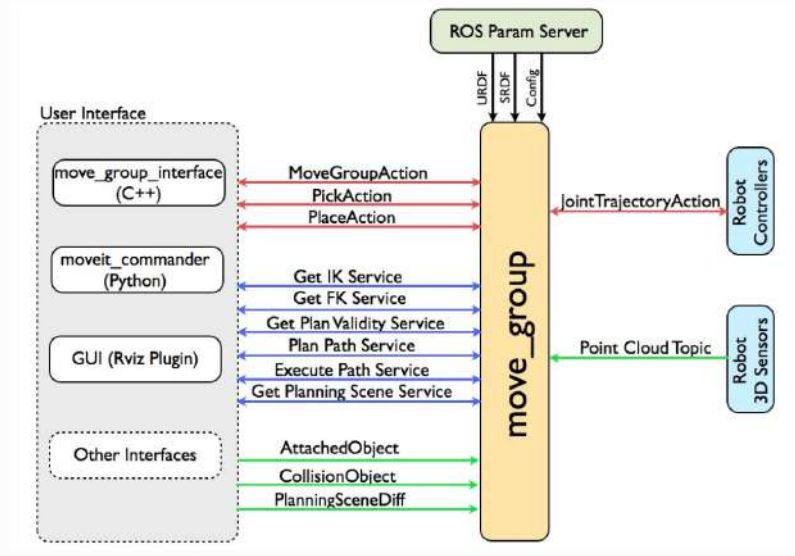


Figura 5.3: Funcionamiento del nodo move_group [1]

La interfaz de este nodo se comunica con el robot mediante *topics* y acciones o servicios recibiendo la información actual del robot, tanto del estado de todas sus articulaciones como lecturas de los sensores que disponga este. Una vez recibidos esta serie de datos, se encarga de comunicarse con los controladores del robot para planificar y ejecutar su movimiento. Además, *move_group* utiliza un servidor de parámetros de ROS como se puede ver en la figura 5.3, que carga la descripción del robot en distintos formatos para fijar su configuración determinada. Y, para completar la configuración, se pone en uso una de las cosas que ha llevado a *Moveit* ser tan potente, que es fijar sus restricciones en el espacio cartesiano o en sus articulaciones. [13]

[1] `https://moveit.ros.org/`

5.1.1.2. Implementación de moveit en el proyecto

Para hacer uso de este planificador, se ha recurrido a una librería, que utiliza el paquete de *moveit*, dónde se han implementado una serie de funciones para planificar, ejecutar trayectorias de diferentes maneras e interaccionar con entornos simulados. La librería fue obtenida y modificada del repositorio [7].

Esta librería usa la ayuda del nodo *move_group* y diferentes recursos como el modelo del robot, creado específicamente para el modelo UR3e, con sus ejes y parámetros característicos. Además, usa mensajes y servicios de *ros2* relacionados con la geometría del mismo y conceptos matemáticos como sistemas de referencia. En el diagrama UML de la figura 5.4 se recogen muchas de estas funciones mencionadas:

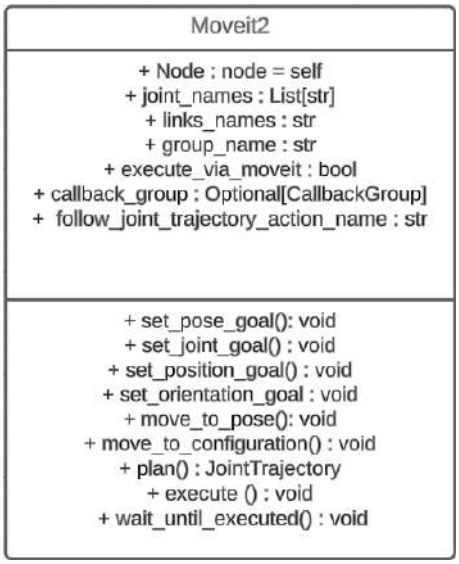


Figura 5.4: Diagrama de clases de la librería creada con *Moveit*

A continuación, en la tabla 5.1 se describen algunas de las funciones más relevantes que se han utilizado para las aplicaciones de Pick and Place.

Nombre	Argumentos obligatorios	Funcionalidad
move_to_pose()	Posición (x, y, z) Orientación (x, y, z, w) Opcional: *Cartesian* (1/0)	Planifica y ejecuta una trayectoria basada en la posición en 3D y la orientación como cuaternio.
move_to_configuration()	Posición de cada articulación en grados Opcional: *Cartesian* (1/0)	Planifica y ejecuta una trayectoria basada en la posición individual de cada articulación
Plan()	Previo *set_pose_goal()* Previo *set_position_goal()*	Planifica la trayectoria con argumentos establecidos o mediante llamadas.
set_pose_goal()	Posición (x, y, z) Orientación (x, y, z, w)	Combinación entre *set_position()* y *set_orientation()*, establece la posición y orientación como paso previo a la planificación.
wait_until_executed()	-	Espera hasta el fin de ejecución de la trayectoria y el robot alcanza su posición esperada.

Tabla 5.1: Funcionalidades importantes librería

5.1.2. Rviz2

Rviz2 es una herramienta de visualización de datos tridimensionales diseñada para aplicaciones de ROS2. Este potente recurso incluye multitud de complementos pudiendo visualizar en tiempo real tanto el modelo del robot como el estado actual de sensores o *drivers* conectados con el ordenador. Es muy frecuente visualizar el movimiento y trayectorias del robot añadiendo además sensores como una cámara o láser, con los que poder añadir información a la situación y restricciones del mismo.

La información técnica de este apartado ha sido recogida del documento [21].

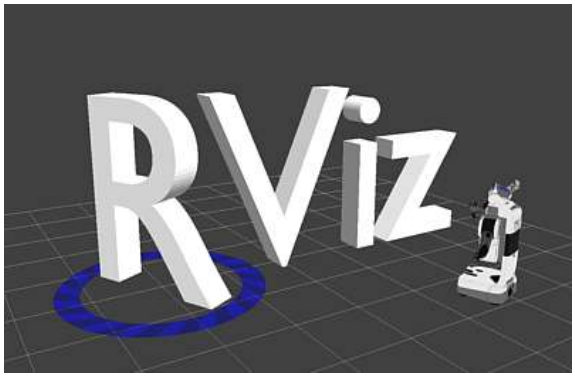


Figura 5.5: Logo de Rviz [2]

Rviz cuenta con una interfaz gráfica en la que visualizar las aplicaciones robóticas pudiendo ver en todo momento la posición, orientación y todos los parámetros del robot respecto de un sistema de referencia a elegir por el usuario. Además, se puede configurar gráficamente. En definitiva, es una forma de tener controlado todos los factores que influyen en la aplicación viendo su estado desde el inicio hasta el final.

Este visualizador es un recurso que se ha utilizado en todo momento durante todo el ciclo de vida del proyecto. En la figura 5.6 se puede visualizar la configuración del robot con la inclusión de la propia cámara como complemento. La adición de la cámara se ha llevado a cabo mediante el *topic* en el que se publica la imágen con formato *ros*.

Las posibilidades más frecuentes del visualizador son: observar el comportamiento de diferentes *topics*, el uso de visualizadores con sus propiedades y estados correspondiente, diferentes vistas y tipos de cámara, así como muchas variedades de proyecciones. Por último, existen opciones para mover el robot como planificar y ejecutar trayectorias con *Moveit* y visualizarlas en vivo dentro del espacio reservado para ello. Estas opciones trae un menú de planificación, denominado *MotionPlanning*, como se puede observar en la figura 5.7.

[2] `https://index.ros.org/r/rviz/`

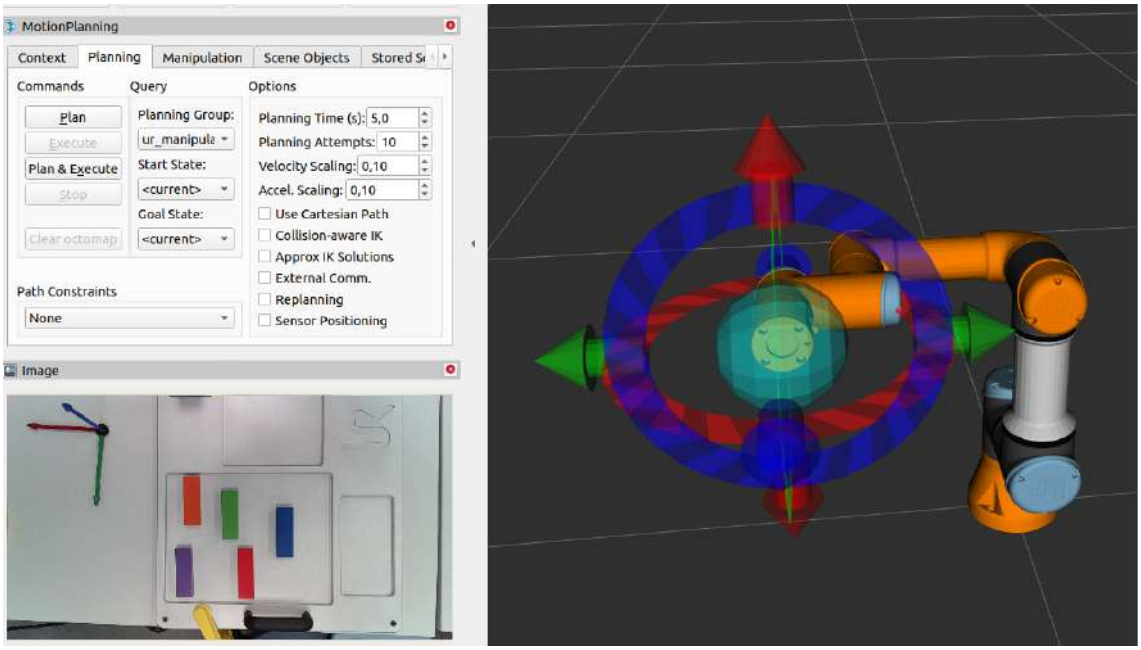


Figura 5.6: Entorno de trabajo de Rviz con imágenes en tiempo real

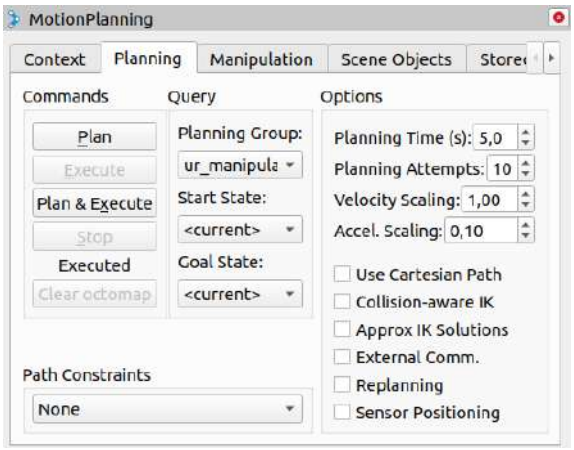


Figura 5.7: Opciones de planificación dentro de Rviz

5.1.3. Librerías utilizadas

5.1.3.1. OpenCV

OpenCV es una librería gráfica de código abierto creada para usos de procesamiento y análisis de imágenes enfocadas a la visión artificial y el aprendizaje automático mediante cientos de algoritmos existentes. Se lanzó originalmente en 1999 con el objetivo de acelerar la percepción visual de los entornos de trabajo donde se utilizaban máquinas de todo tipo. Cuenta con una comunidad de miles personas y actualmente, esta librería gráfica es usada en entornos industriales y de investigación, sobre todo en campos como la robótica, seguridad o medicina. [5]

Es compatible con numerosos lenguajes de programación como *Python*, *C++* o *Java* y está dotada con soporte para diferentes sistemas operativos como *Windows*, *Linux*,*Mac* o *Android*. Los algoritmos implementados y en continuo desarrollo están centrados en una

serie de tareas concretas muy útiles para la visión por computador, algunas de las tareas más populares son:

- Identificación y clasificación de objetos o patrones
- Rastreo de trayectorias o movimientos
- Generación de modelos en tres dimensiones
- Seguimientos y navegación de robots

[5]

Figura 5.8: Logo de OpenCV [3]

5.1.3.2. DepthAI

DepthAI es una plataforma de inteligencia artificial especializada en visión por computador que ofrece una serie de servicios para diferentes cámaras fabricadas por la empresa Luxonis. Dentro de este tipo de cámaras, está incluido el modelo OAK-D Lite, el cual se ha utilizado para desarrollar la aplicación de Pick and Place de este proyecto. La parte de visión por computador del proyecto utilizará nodos, clases y funciones propias de su librería. [27]

Las características principales de la plataforma propia de DepthAI son las siguientes: el uso de la inteligencia artificial, la visión por computador, la percepción de profundidad, su rendimiento y alta resolución, y sus soluciones integradas de bajo consumo. [27]

Figura 5.9: Plataforma DepthAI de Luxonis [4]

5.1.3.3. Tkinter

Tkinter es una de las librerías gráficas estándar incluida de forma predeterminada en *Python* para crear interfaces de usuario. Cuenta con una serie de herramientas *GUI* muy interesantes que permiten añadir widgets personalizados y diseñar las ventanas y los componentes de las mismas a gusto del usuario. [4]

[3] https://docs.opencv.org/4.7.0/
[4] https://docs.luxonis.com/en/latest/

Además, esta librería cuenta con soporte para varios sistemas operativos como *Unix* o *Windows*. Originalmente, fue diseñada para el lenguaje de programación Tcl, pero actualmente es una de las más usadas en *Python* y ha sido la herramienta principal para crear la interfaz del proyecto implementado. [4]

5.2. Descripción del diagrama de secuencia

Se van a detallar, como punto de partida, todos los eventos generados en la ejecución del proyecto para desembocar posteriormente en la implementación de las aplicaciones de Pick and Place, las cuales, son el objetivo principal del proyecto. Dicha implementación ha sido dividida en tres partes principales como se introdujo en el capítulo de diseño 4.3.2, siendo estas las responsables del flujo de funcionamiento de las aplicaciones. No obstante, el control del robot, la visión artificial y la interacción con el usuario, requieren del correcto funcionamiento de los controladores del robot y de otros nodos secundarias que ponen en marcha el proyecto.

Por consiguiente, estas secciones serán las partes del sistema representado en el diagrama de secuencias de la figura 5.10, el cual refleja la interacción de todos los componentes del sistema a lo largo del tiempo de ejecución del proyecto.

A continuación se especifica la secuencia de ejecución presentada en el DSS con el fin de comprender completamente su operación, lo cual resulta crucial para tomar una perspectiva real de la implementación y de todos los componentes involucrados.

1. En primer lugar, el usuario introducice en la terminal de su ordenador el comando que ejecuta el archivo de lanzamiento correspondiente, una vez haya configurado su entorno de trabajo correctamente como se explica en el manual de usuario del apéndice A.1.

2. Seguidamente, en este archivo se ejecutan todos los componentes del proyecto, empezando así su proceso de operación. Por un lado, se llevan a cabo las conexiones con los controladores de todos los dispositivos, añadiendo la ejecución necesaria del programa de control externo de la tablet para completarlo. Por otro lado, en el ordenador aparecen las siguientes ventanas:

   - El visualizador Rviz con el robot y las imágenes de las cámaras en tiempo real para seguir el proceso de operación en todo momento.
   - Las imágenes de las cámaras para visualizar la parte de visión y cómo se realizan las aplicaciones de Pick and Place desde una vista superior.
   - La interfaz gráfica de usuario, que será un componente clave en este diagrama de secuencia, ya que se encarga de comunicarse con los demás componentes para integrar las aplicaciones.

3. Por lo tanto, en este punto, se hará click en la interfaz de usuario para seleccionar que caso de uso se desea poner en marcha y con qué orden de objetos se desea realizar.

   a) Primeramente, se introduce la secuencia de colores de los objetos para establecer el orden de su posterior manipulación. En consecuencia, esta secuencia de colores es enviada al nodo de la cámara, el cual se encargará de gestionar estos datos para detectarlos, ubicarlos y comunicar sus posiciones al control del robot.

b) En segundo lugar, se deberá elegir el tipo de aplicación de Pick and Place que se desea llevar a cabo: orden, paletizado o despaletizado de piezas. Seguidamente, será comunicada su elección al nodo de control otorgando la última información para el inicio de la aplicación.

Finalmente, se lleva a cabo la aplicación seleccionada en función de la selección del usuario pudiendo ser visualizada de tres formas diferentes: a través del visualizador, mediante la cámara y en vivo.

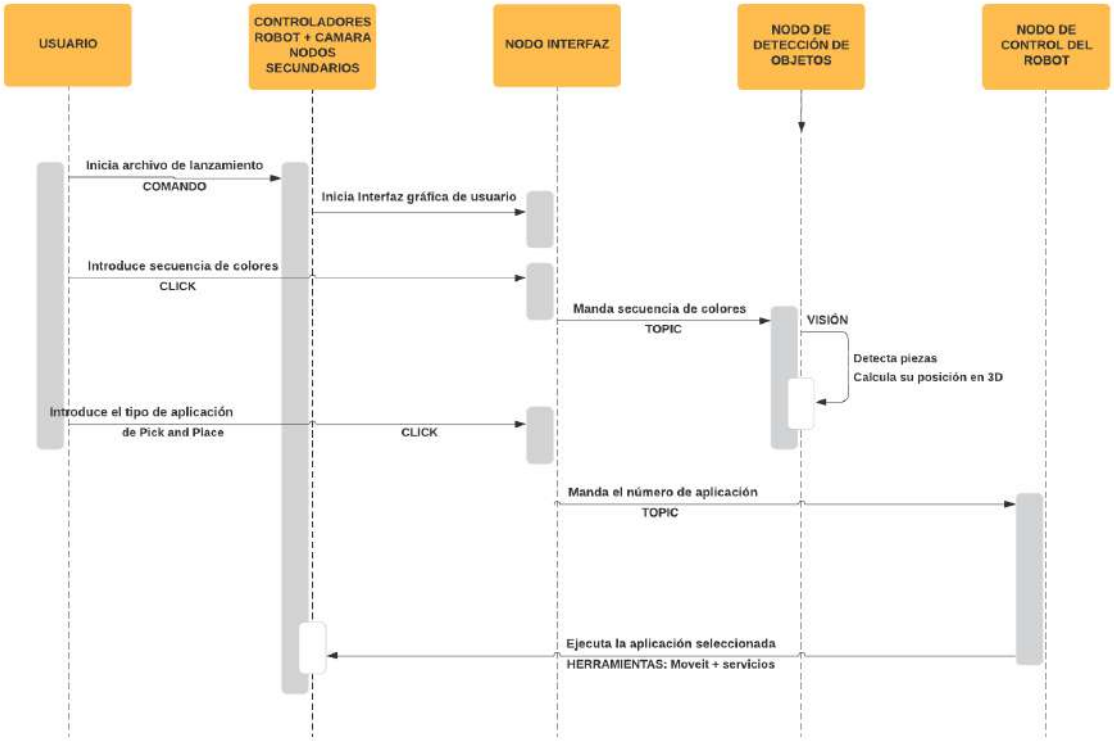


Figura 5.10: Diagrama de secuencia

5.3. Distribución de paquetes

A continuación, se va a llevar a cabo una explicación de todos los paquetes creados y clonados para el desempeño de esta aplicación con ROS2. Asimismo, se recogerán los nodos y recursos que presentan cada uno. El arbol de paquetes del proyecto se puede observar en la figura 5.11.

El procedimiento a seguir para conseguir estos paquetes sería clonar el repositorio del autor como se ve en el código 5.1.

código 5.1: Comando para clonar el repositorio

```
git clone https://github.com/mariooot13/Pick-and-Place-with-ROS2/tree/tutorial
```

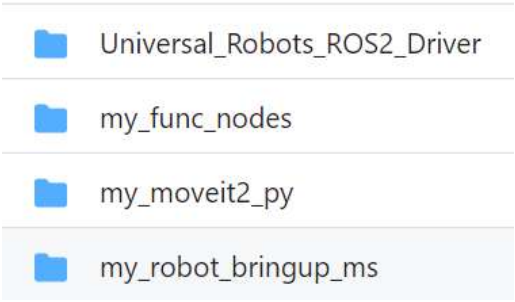


Figura 5.11: Lista de paquetes del proyecto

5.3.1. my_func_nodes

Este paquete es una pieza clave del proyecto debido a que incluye los cuatro nodos de elaboración propia vistos en la tabla 4.3 que se encargan del funcionamiento básico de todas las partes del proyecto. Adicionalmente, incluye una carpeta de recursos donde se encuentran herramientas auxiliares como una calculadora de conceptos de sistemas de referencia y ciertos recursos de apoyo para los nodos como imágenes para la interfaz gráfica. Se pueden visualizar los diferentes nodos de la carpeta en la figura 5.12.

Figura 5.12: Lista de nodos

Este es el paquete de trabajo diario del proyecto en el cual se han creado y evolucionado las tres partes fundamentales del mismo: el control del robot, la visión por computador a través de la cámara y la interfaz de usuario. Como consecuencia, si se desea crear una solución diferente o adicional, se debería crear dentro de este.

Por último, si se desea lanzar alguno de estos nodos por separado para probar cualquier parte del proyecto individualmente o incluso en paralelo, el comando que se debe introducir en la terminal es el que se refleja en el código 5.2.

código 5.2: Comando para lanzar nodos

```
ros2 run my_func_nodes control_robot_exec
ros2 run my_func_nodes camera_exec
ros2 run my_func_nodes camera_detection
ros2 run my_func_nodes interfaz_exec
```

5.3.2. Paquete de controladores de UR

Una de las primeras cosas que se deben conocer para trabajar con el UR3e es el paquete de controladores que administra *Universal Robots* con el fin de poder trabajar con cualquiera de sus modelos en un entorno de ROS2. En este proyecto, el primer paso fue clonar el repositorio dentro de una carpeta nueva y aprender la forma de trabajo de estos para poder desempeñar un trabajo individualizado posteriormente. El comando que se debe introducir para clonar este paquete es el reflejado en la figura 5.3, la rama elegida ha sido *Foxy*, dado que es la distribución de ROS2 usada durante todo el proyecto.

código 5.3: Comando para clonar los controladores de UR

```
1 git clone -b foxy https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver.
      git src/Universal_Robots_ROS2_Driver
```

La distribución de estos se puede observar en la figura 5.13.

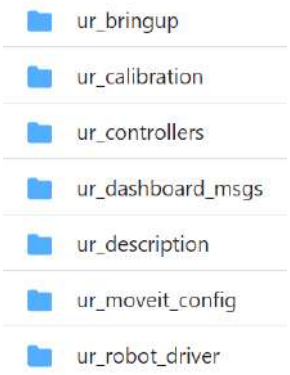


Figura 5.13: Lista de paquetes de los drivers de UR [5]

Y, además, se ha recogido una descripción de las funcionalidades más importantes de cada paquete y una explicación de por qué se han incluido en el proyecto en la tabla 5.2.

el comando encargado de lanzar este paquete se muestra en el código 5.4. En la tabla 5.3 se han recogido los argumentos más importantes a la hora de lanzar los controladores de UR. Además, se muestra como usar el robot de forma simulada dentro del entorno Rviz.

código 5.4: Comando para lanzar los controladores de UR

```
1 ros2 launch ur_bringup ur_control.launch.py ur_type:=
    ur3e robot_ip:=192.168.20.35 use_fake_hardware:=
    false launch_rviz:=true
```

5.3.2.1. Conexión UR3e vía ordenador

La conexión física del robot con un dispositivo externo se realiza mediante un cable Ethernet. En este caso, el dispositivo a conectar sería el ordenador y es necesario configurar ciertos parámetros.

[5] https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver/tree/foxy

<table>
<tr><td>ur_bringup</td><td>Contiene varios archivos de lanzamiento que lanzan diferentes nodos en función de lo que se desee hacer. En este caso, se ha utilizado el archivo llamado ur_control.launch.py, encargado de lanzar los controladores tanto para las articulaciones como para los pines de entrada y salida. Y, además, cuenta con la opción de lanzar Rviz con un hardware simulado, brindando la opción de trabajar con el modelo sin necesidad de disponer de ello físicamente. Por otro lado, tambien es interesante y se ha usado el lanzador denominado ur_moveit.launch.py, explicado en el apartado 5.3.3.</td></tr>
<tr><td>ur_moveit_config</td><td>Su función principal es ejercer de adaptador entre el controlador de movimientos del robot y el propio planificador Moveit.
Junto con esto, se encarga de recoger las configuraciones necesarias para el planificador, así como diferentes planos de colisión.</td></tr>
<tr><td>ur_description</td><td>Es un paquete imprescindible para el funcionamiento de los controladores del robot, puesto que contiene todos los datos característicos de cada modelo de UR disponible. Además, ha sido modificado para cumplir ciertos requisitos en el proyecto. Sus funciones principales han sido recogidas en la siguiente tabla:
<table>
<tr><th>Ubicación</th><th>Función</th></tr>
<tr><td>config</td><td>Es uno de los paquetes más valiosos ya que incluye todos los datos de configuración del modelo. Entre estos, se incluye la cinemática de cada articulación, parámetros como centro de masas, enlaces de cada articulación o parámetros visuales. Por último, el archivo más utilizado en este proyecto ha sido los límites de las articulaciones, en cuánto a grados y velocidad, ya que es importante a la hora de restringir ciertos movimientos.</td></tr>
<tr><td>meshes</td><td>Contiene todos los modelos con formato stl cuyo papel es representar el modelo tridimensional del robot seleccionado. Esto es lo que hace posible la visualización del modelo del robot dentro de Rviz como se observa en la figura 5.6.</td></tr>
<tr><td>urdf</td><td>Contiene archivos con formato .xacro que generan los archivos urdf[6]que almacenan descripciones importantes de componentes del modelo robótico.</td></tr>
</table></td></tr>
<tr><td>Restantes</td><td>Se puede ver una breve descipción de estos dentro del repositorio de Github de UR. Cabe destacar que la mayoría de paquetes explicados en esta tabla hacen uso de algunos de estos recursos, por lo que sesultan relevantes a la hora de implementar el proyecto.</td></tr>
</table>

Tabla 5.2: Descripción de los controladores de UR

El primer paso se lleva a cabo dentro de la tablet del robot, configurando una dirección IP dentro del apartado de sistema en configuración. El siguiente paso, sería configurar una dirección IP distinta en el ordenador. Y, posteriormente, se creará un programa en la tablet del robot seleccionando la opción de External Control asignando la dirección del ordenador. Por último se lanzarán los drivers de UR en el ordenador con la dirección IP del robot y la conexión resultará exitosa. A partir de este momento, si el robot está conectado correctamente, se deberá visualizar su configuración actual dentro del entorno de Rviz. La conexión viene explicada con detalle en el documento [49].

Si la conexión es correcta y todos los paquetes son lanzados con éxito se debería de mostrar por terminal las instrucciones de la imágen 5.15, corroborando que la conexión de los controladores y de *Moveit* es éxitosa.

Nombre del argumento	Descripción	Valor propio
ur_type	Tipo de modelo UR	UR3e
robot_ip	Asignación de IP al robot para poder llevar a cabo su conexión con el ordenador mediante cable *Ethernet*.	192.168.20.35
use_fake_hardware	Opción de activar el *hardware* del robot de forma simulada, pudiendo trabajar con este sin necesidad de disponer de ello presencialmente	*True / False*
launch_rviz	Posibilidad de lanzar *Rviz* para visualizar el robot en directo	*True / False*
initial_joint_controller	Opción de asignar una posición específica de inicio al robot en el caso de usar el *hardware* simulado. Esta posición debe ser leída de un fichero .yaml como en este caso.	*True / False*

Tabla 5.3: Argumentos para lanzar los controladores de UR

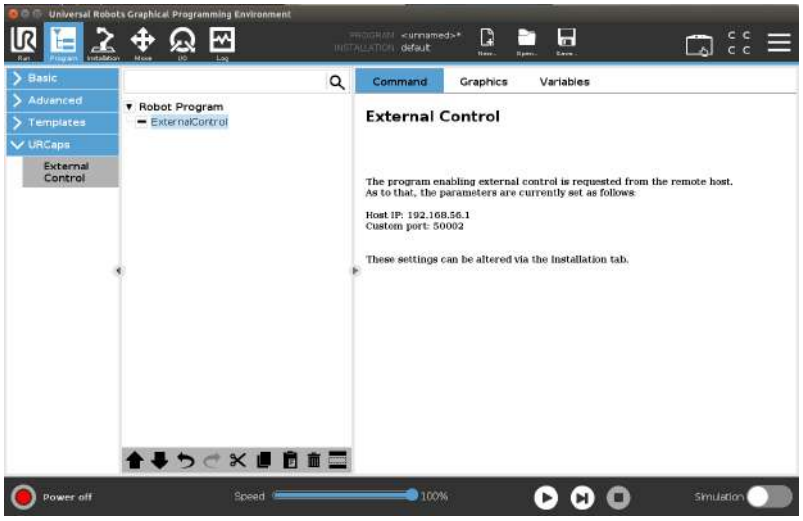


Figura 5.14: Control externo desde la tablet del UR3e

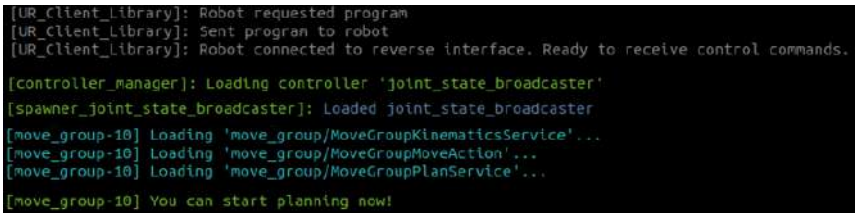


Figura 5.15: Salida por pantalla de la conexión inicial

5.3.3. my_robot_bringup_ms

Este directorio contiene un archivo de lanzamiento base encargado de ejecutar todos los demás archivos de lanzamiento y nodos de forma simultánea obteniendo una integración de todos los componentes mencionados. Su papel fundamental es poner en marcha todo lo necesario para el funcionamiento completo del proyecto. Además, cuenta con una

carpeta de configuración como complemento. A continuación se presenta una lista de todos los elementos lanzados por este archivo:

- *ur_control.launch.py*. Archivo que lanza los controladores de UR mencionados en el apartado 5.3.2. Este lanzamiento es vital para el funcionamiento completo del proyecto dado que lleva a cabo la conexión con el robot y lanza nodos importantes para su funcionamiento.
- *ur_moveit.launch.py*. Archivo que lanza el nodo *move_group* mencionado en el apartado 5.1.1.1, así como un nodo necesario para su visualización dentro de *Rviz* y otro nodo de relaciones espaciales en aplicaciones robóticas.
- *control_robot_node_exec*. Ejecutable del nodo de control principal del robot. Este nodo mencionado en el apartado 4.3.3.1 es el encargado de llevar a cabo el movimiento tanto del robot como de su herramienta con todos los algoritmos y comunicaciones que esto conlleva.
- *camera_exec*. Ejecutable del nodo de la cámara que lanza su visualización y publica las imágenes a color y en escala de grises en tiempo real con formato ros.
- *camera_detection*. Ejecutable del nodo encargado del análisis de las imágenes publicadas por la cámara y de la detección, reconocimiento y obtención de posición de los objetos. Además, incluye todo tipo de comunicaciones en el diagrama principal.
- *interfaz_exec*. Ejecutable del nodo responsable de la creación de la interfaz y de sus comunicaciones con el resto de nodos según la interacción externa.

De forma adicional, se va a explicar la estructura de este describiendo cómo se lanzan los diferentes componentes en su interior:

1. Se puede observar en el código 1 cómo lanzar un archivo de lanzamiento dentro de otro para obtener un proyecto limpio y fácil de usar. En este caso, se lanza el archivo de moveit, ubicando en una primera instancia el paquete dentro de los controladores de UR que lo contiene y, por último, introduciendo dentro de la instancia sus argumentos necesarios.

```
#Ejecucion del archivo de lanzamiento de Moveit
moveit = IncludeLaunchDescription(
    PythonLaunchDescriptionSource([
        PathJoinSubstitution([
            FindPackageShare('ur_bringup'),
            "/home/mario/workspace/ros_ur_driver/src/
                Universal_Robots_ROS2_Driver/ur_bringup/launch",'
                ur_moveit.launch.py'
        ])
    ]),
    launch_arguments={
        'ur_type': 'ur3e',
        'robot_ip': '192.168.20.35',
        'launch_rviz':'true'
    }.items()
)
```

2. Para terminar de entender el paquete actual, en el código 5.5 se observa cómo se ejecuta un nodo específico dentro del archivo de lanzamiento. En este caso, el nodo

es la detección de objetos y se introduce su paquete, nombre de ejecutable y archivo, así como su salida por pantalla.

código 5.5: Código para ejecutar un nodo en un archivo de lanzamiento

```
#Ejecucion del nodo de deteccion de objetos, camera_detection
camera_detection = Node(
    package="my_func_nodes",
    executable="camera_detection",
    name="camera_pub_pos",
    output={
        "stdout": "screen",
        "stderr": "screen",
    },
)
```

Dentro de este paquete, existe una carpeta denominada *config* en la cual se encuentran los archivos de argumentos necesarios para el lanzamiento correcto del nodo de control. Este archivo llamado param_bringup.yaml contiene el nombre de cada articulación y del controlador principal, así como sus límites de inicio y posiciones de inicio para la opción simulada.

En último lugar, el comando final, con el cual se ejecuta todos los recursos mencionados en este apartado, se muestra en el código 5.6.

código 5.6: Comando de lanzamiento en terminal

```
ros2 launch my_robot_bringup_ms control_robot.py
```

5.3.4. my_moveit_py

El último paquete incluye la librería con todas las funciones implementadas de Moveit para planificar y ejecutar trayectorias mencionadas en el apartado 5.1.1.2. Además, cuenta con el modelo del UR3e creado para especificar todos los nombres de cada articulación y otros parámetros como el nombre de los manipuladores, y poder hacer uso de las funciones descritas en la tabla 5.1 de forma correcta.

La solución adoptada del uso de este *framework* de planificación se ha realizado de forma aislada a los distintos paquetes, con la intención de poder ser sustituido por otras versiones e incluso otros planificadores si se desea. Esta práctica aporta calidad y escalabilidad al proyecto.

5.4. Implementación del control del robot

El control del robot es una parte fundamental del proyecto dado que tanto su movimiento como el uso de su herramienta hacen posible la finalidad de las aplicaciones de Pick and Place descritas de forma exitosa. Este apartado relata en detalle cómo se han llevado a cabo el cálculo y ejecución de las trayectorias que debe tomar, sus múltiples posibilidades, así como la manipulación de objetos con la herramienta.

[6]Unified Robot Description Format

5.4.1. Planificación y ejecución de trayectorias

El movimiento del robot es una de las tareas más complejas cuando se trabaja con robots manipulativos ya que necesitan tener una gran precisión y acierto para manipular sus objetos. En el momento de planificar una trayectoria libre de colisiones, es necesario conocer los siguientes requisitos:

- Punto de partida.
- Punto de destino.
- Limitaciones físicas del robot.
- Restricciones establecidas en el espacio de trabajo para la tarea concreta. [23]
- Sistema de coordenadas utilizado.

La planificación y ejecución de estas trayectorias se ha llevado a cabo en el nodo de control enunciado en el apartado 4.3.3.1 haciendo uso del nodo *Move_group* explicado en detalle en el apartado 5.1.1.1. Además, las trayectorias se observan en todo momento dentro de *Rviz* como se ilustra en la figura 5.16.

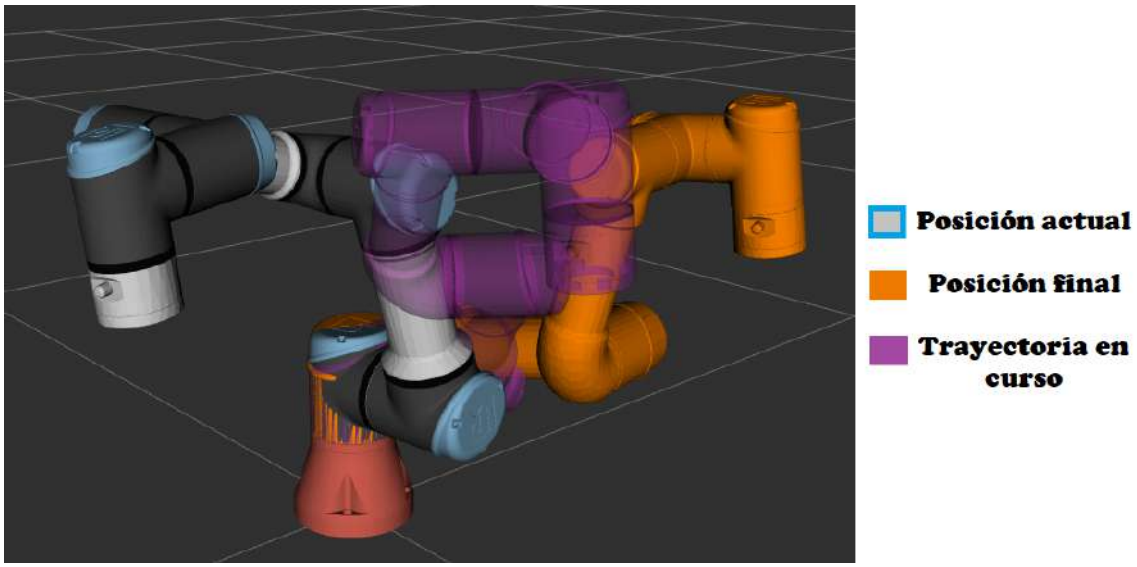


Figura 5.16: Visualización de trayectorias en Rviz

5.4.1.1. Planificadores

Existe una extensa variedad de planificadores disponibles dentro de *Moveit* para calcular las trayectorias del robot. Todos estos se encuentran disponibles dentro del paquete de controladores de UR, en concreto, en el paquete denominado *ur_moveit_config/config* se encuentra el archivo de parámetros correspondiente llamado *ompl_planning.yaml*. Se pueden observar en la figura 5.17 y, además, dentro de este se pueden ver y configurar parámetros de los mismos como rangos, factores y preferencias para los objetivos de planificación.

El planificador por defecto y el utilizado en este proyecto ha sido uno de todos los presentados denominado *RRTConnect*, cuyas siglas significan *Rapidly-exploring Random*

```
ur_manipulator:
  planner_configs:
    - SBLkConfigDefault
    - ESTkConfigDefault
    - LBKPIECEkConfigDefault
    - BKPIECEkConfigDefault
    - KPIECEkConfigDefault
    - RRTkConfigDefault
    - RRTConnectkConfigDefault
    - RRTstarkConfigDefault
    - TRRTkConfigDefault
    - PRMkConfigDefault
    - PRMstarkConfigDefault
```

Figura 5.17: Lista de planificadores disponibles

Trees. Este planificador fue creado para buscar rápidamente rutas sin colisiones para robots y sistemas dinámicos en espacios de configuración, y tiene como propósito encontrar trayectorias continuas entre una configuración inicial y una final del robot.

'El método funciona mediante la construcción incremental de dos árboles aleatorios de exploración rápida (RRT) enraizados en las configuraciones de inicio y objetivo', como se puede observar en la figura 5.18. [2]

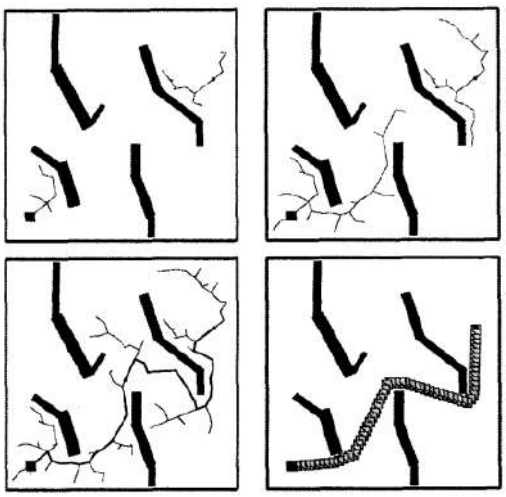


Figura 5.18: Evolución de los arboles de exploración rápida RRT [7]

5.4.1.2. Métodos y tipos de trayectorias investigados

Como dato inicial, es importante resaltar que existen y se han investigado múltiples maneras de trazar trayectorias.

[7] Imágen obtenida del documento [2]

1. En los controladores del modelo UR se incluye la posibilidad de planificar trayectorias sencillas con el uso de un paquete existente de ROS2 conocido como *ros2_control_test_nodes*, paquete de demostraciones de UR, que contiene un nodo llamado *publisher_joint_trajectory_position_controller*.

   Dentro de este nodo se puede planificar una trayectoria cartesiana mediante las coordenadas de cada uno de los grados de libertad del robot, creando una serie de puntos que se publicarán en un topic denominado */joint_trajectory_position_controller* y se comunicará con los controladores para ejecutarla. El problema de este método son las limitaciones mencionadas anteriormente, como la necesidad de conocer el ángulo específico de cada articulación en todo momento y la restricción lineal que poseen.

2. Por otro lado, la metodología seguida en este proyecto ha tomado un enfoque distinto como consecuencia del uso de la librería de *Moveit* mencionada en 5.1.1.2.

   Se ha utilizado uno de los procedimientos introducidos en el apartado 5.1.1.2. El método *move_to_pose()* ha sido considerado como la mejor opción ya que solo necesita tomar como argumento la posición y orientación del punto final para planificar y ejecutar su trayectoria. Además de esto, permite la posibilidad de planificar de forma cartesiana o libre.

5.4.2. Procedimiento implementado

La implementación del control del robot se ha realizado dentro del nodo de control explicado en el apartado 4.3.3.1 y se han seguido los siguientes pasos:

- Instanciación de la clase propia *control_robot_master*.
- Instanciación de la clase *Moveit* y uso de la misma.
- Programación en tiempo real mediante hilos de ejecución.

En cuánto a la programación implementada de la clase *control_robot_master*, se puede dividir en tres partes principales: el estado del robot, el movimiento del mismo y el uso de la herramienta.

5.4.2.1. Estado del robot

En primer lugar, dentro del constructor de la misma clase se han declarado los argumentos necesarios para conocer toda la información del robot en tiempo real, y poder utilizar ciertos nodos proporcionados por los controladores de UR. Estos corresponden con los nombres de cada articulación y ciertos argumentos que complementan la configuración inicial del robot.

Seguidamente, se comprueba si la posición de inicio establecida es correcta y de ser así, se crea una suscripción al *topic* denominado */joint_states* para tener conocimiento en tiempo real del estado físico de cada articulación con el ánimo de prevenir errores o posibles colisiones. Se puede observar en el código 5.7.

código 5.7: Comprobación del correcto estado del robot

```
1 if self.check_starting_point:
2     # Declaracion de parametros anidados
3     for name in self.joints:
4         param_name_tmp = "starting_point_limits" + "." + name
```

```
5         self.declare_parameter(param_name_tmp, [-2 * 3.14159, 2 * 3.14159])
6         self.starting_point[name] = self.get_parameter(param_name_tmp).
              value
7         #Comprobacion de un inicio correcto
8     for name in self.joints:
9         if len(self.starting_point[name]) != 2:
10            raise Exception('"starting_point" parameter is not set
                  correctly!')
11    self.joint_state_sub = self.create_subscription(
12            JointState, "joint_states", self.joint_state_callback, 10
13        )
```

Por último, existe una función denominada *joint_state_callback* que actúa de supervisor para el estado de cada articulación usando un algoritmo de verificación que finaliza en la comprobación del correcto inicio del robot sin haber excedido los límites de sus articulaciones. Se puede observar el código 5.8.

código 5.8: Comprobación de los límites del robot actualest

```
1 #Verificacion de no haber sobrepasado los limites
2 if any(limit_exceeded):
3     self.starting_point_ok = False
4 else:
5     self.starting_point_ok = True
```

Además, se han modificado ciertos parámetros de los controladores de UR limitando el rango de movimiento de cada grado de libertad en vista de adaptar las futuras trayectorias al espacio de trabajo actual, y no sobrepasar dichos límites mencionados. Este archivo se denomina *joint_limits*, se encuentra dentro del paquete *ur_description* mencionado en el apartado 5.3.2 y se puede observar en la figura 5.19.

```
joint_limits:
  shoulder_pan:
    # acceleration limits are not publicly available
    has_acceleration_limits: false
    has_effort_limits: true
    has_position_limits: true
    has_velocity_limits: true
    max_effort: 56.0
    max_position: !degrees  180.0
    max_velocity: !degrees  360.0
    min_position: !degrees  -180.0
```

Figura 5.19: Limitaciones físicas de cada articulación

5.4.2.2. Uso de la herramienta

Se ha llevado a cabo el control de la herramienta dentro de esta clase con el objetivo de poder coordinar su apertura o cierre en función del estado del robot y de los objetos fijados.

Dentro del paquete de controladores proporcionado por UR, existen multitud de servicios ya creados para hacer uso de todas las funciones que pueden ofrecer sus modelos. En este caso, existe un servicio denominado *SetIO* creado por un servidor de UR, con el objetivo de poder cambiar el estado del pin correspondiente en la caja de control. Tiene la siguiente sintaxis:

Nombre del argumento	Valor
fun	Función a desarrollar
pin	ID numérico del pin establecido
state	Estado actual del pin (Nivel de señal analógica mediante voltaje) *ON/OFF*

Tabla 5.4: Argumentos de solicitud del servicio de la herramienta

Asimismo, este servidor se puede visualizar, siempre y cuando se hayan lanzado los controladores de UR, introduciendo el comando por terminal del código 5.9.

código 5.9: Comando para observar el servicio de la pinza

```
ros2 service info /io_and_status_controller/set_io
```

Por lo tanto, se ha creado un cliente de ese servicio en el cual se configuran como parámetros de solicitud los vistos en la tabla 5.4.

código 5.10: Servicio de UR

```
client = self.create_client(SetIO,"/io_and_status_controller/set_io")
```

Se mantendrá el valor 1 para el parámetro *fun* para indicar que la función deseada es el agarre, el pin correspondiente de la caja de control para abrir o cerrar, y el estado mediante una tensión específica que son: 24 voltios para activar y 0 para desactivar el agarre de la herramienta. Es importante conocer que una vez activada la pinza, por ejemplo, se cierra cogiendo un objeto, se debe desactivar para dejarla lista de cara a la siguiente operación de apertura. De esta forma, se han creado dos funciones como la representada en el código 5.11 para abrir y cerrar la pinza donde se solicitan estos parámetros de forma sencilla. El pin 17 ejecuta el cierre de la pinza y el 16 su apertura.

código 5.11: Función de cerrar con la llamada del servicio

```
def close_gripper(self):
    self.call_gripper(1, 17, 24.0)  #Activa la pinza.
    time.sleep(0.5)
    self.call_gripper(1,17,0.0)     #Desactiva la pinza.
```

Por último, cuando se lleva a cabo cualquier función de la herramienta se visualiza por pantalla el siguiente mensaje de la figura 5.20 confirmando así, su correcto funcionamiento.

```
[ros2_control_node-2] [INFO] [1682343883.711176299] [io_and_status_
controller]: Setting digital output '16' to state: '24'.
[ros2_control_node-2] [INFO] [1682343884.212720715] [io_and_status_
controller]: Setting digital output '16' to state: '0'.
```

Figura 5.20: Salida por pantalla del controlador del gripper

Finalmente, el diagrama de nodos que representa los controladores correspondientes de la herramienta se puede visualizar en la figura 5.21. El nodo proporcionado por UR denominado *io_and_status_controller*, se encarga de gestionar el comportamiento de las

entradas y salidas conectadas de forma adicional. En este caso, la única entrada conectada es la herramienta.

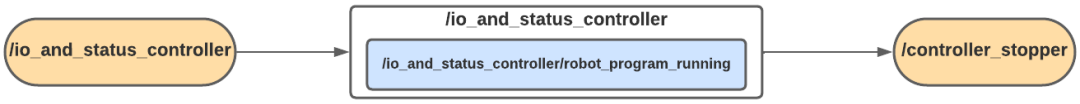


Figura 5.21: Diagrama de nodos de la herramienta

5.4.2.3. Movimiento del robot

El movimiento del robot en relación a la planificación de trayectorias se compone de tres ideas principales: la obtención de la posición proporcionada por la cámara, la ejecución de las trayectorias en base a esta y el tipo de caso de uso a realizar según la comunicación con la interfaz de usuario.

1. La primera de estas ideas se lleva a cabo mediante la suscripción del *topic* llamado *object_position*, el cual es publicado por el nodo de detección de objetos y contiene la posición y orientación del objeto a recoger como se observa en la figura 5.22.

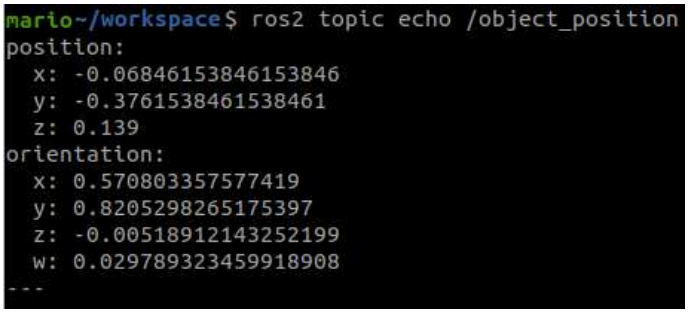


Figura 5.22: Topic object_position

Su suscripción se lleva a cabo como se observa en el código 5.12, y la función asociada al mismo en el código 5.13 que contiene comprobaciones de planos de seguridad para la altura del objeto.

código 5.12: Suscriptor al topic /object_position

```
    #Suscripcion de la posicion del objeto
    self.pose_required = Pose() #Estructura para la posicion
    self.subscriber_camera = self.create_subscription(Pose, "
        object_position",self.callback_recibo_pos_pedida, 10)
```

código 5.13: Función que obtiene la posición del objeto

```
def callback_recibo_pos_pedida(self, msg):
    self.pose_required.position.x = msg.position.x
    self.pose_required.position.y = msg.position.y

    if msg.position.z < 0.3:
        self.pose_required.position.z = msg.position.z
```

```
7     else:
8         self.pose_required.position.z = 0.139
9
10    self.pose_required.orientation.x = msg.orientation.x
11    self.pose_required.orientation.y = msg.orientation.y
12    self.pose_required.orientation.z = msg.orientation.z
13    self.pose_required.orientation.w = msg.orientation.w
```

2. En relación al movimiento del robot, gracias a la implementación de la librería mencionada de *Moveit*, así como todas las comprobaciones de su estado, se ha logrado compactarlo en el uso de una instanciación de dicha clase y el uso de la clase *control_robot_master* dentro de la función *main()*.

   En el código de la figura 5.14, se instancia la clase *Moveit* administrando toda la información del modelo creado del robot UR3e para poder usar sus funciones de forma sencilla.

código 5.14: Instanciación de la clase Moveit

```
1 moveit2 = MoveIt2(
2     node=control_node,
3     joint_names=ur3e_model.joint_names(),
4     base_link_name=ur3e_model.base_link_name(),
5     end_effector_name=ur3e_model.end_effector_name(),
6     group_name=ur3e_model.MOVE_GROUP_ARM,
7     #callback_group=self.callback_group,
8     )
```

   Por último, se crea un objeto de la clase *control_robot_master* y por lo tanto, se obtienen las posiciones de las piezas. En función del caso de uso introducido en la interfaz gráfica se mueve el robot de una manera u otra. Se ha creado un algoritmo diferente para cada caso de uso con ligeras modificaciones en las ubicaciones de destino y en la orientación de la herramienta.

   En el código 5.15 se observa el procedimiento estándar con el que se llevan a cabo cada una de las trayectorias que las aplicaciones de Pick and Place requieren.

código 5.15: Planificación y ejecución de trayectorias

```
1 #Posicion del objeto analizado por la camara:
2
3 position_r = control_node.pose_required_print()
4
5 #Planificacion y ejecucion de trayectorias:
6 moveit2.move_to_pose(position[position_r.position.x,position_r.
      position.y,position_r.position.z + 0.1], quat_xyzw=position_r.
      orientation, cartesian=True)
7
8 #Espera activa hasta la finalizacion de la ejecucion
9 moveit2.wait_until_executed()
```

5.5. Implementación de la visión por computador

En esta sección se va a explicar en profundidad cómo se ha llevado a cabo la visión por computador del proyecto con el uso de las dos librerías gráficas expuestas en el apartado

5.1.3: *OpenCV* y *DepthAI*. Además, parte de los códigos explicados son proporcionados por estas librerías. [27] [5].

La visión desarrollada incluye tres puntos principales: el reconocimiento de objetos, el cálculo de su posición tridimensional y su posterior comunicación con el control del robot.

En este caso, se han creado dos nodos que se coordinarán para obtener el resultado final: el nodo *camera*, el cual se explicará en el apartado de procesamiento de imágenes; y el nodo *object_detector*, cuya explicación se dividirá en plano vertical y horizontal. Esta distinción se lleva a cabo con el fin de formar la posición de los objetos con la combinación de ambas.

5.5.1. Procesamiento de imágenes

En primer lugar, el objetivo del primer nodo introducido en el apartado 4.3.4.1, tiene como objetivo lanzar las imágenes descritas anteriormente con un formato *ros* para, posteriormente, ser analizadas e interpretadas en el nodo de detección. A continuación se van a detallar las diferentes etapas que se han seguido hasta la formación de las imágenes.

Antes de entender el funcionamiento, es muy importante entender que cuando se habla de nodos en procesamiento de imágenes, no se refiere a los nodos de ROS2 expuestos, sino a *componentes del sistema de procesamiento de imágenes*. Para llevarlo a cabo, se instancian cuatros nodos de entrada proporcionados por *DepthAI*: uno de ellos para la imágen RGB, constituyendo el plano horizontal; y los otros tres para obtener el mapa de disparidad, del cual se obtendrá la profundidad en el plano vertical. Asimismo, se crean dos nodos de salida que serán los protagonistas finales de la detección de los objetos. El plan trazado de los nodos se puede visualizar en la tabla 5.5. Además, su sintaxis en programación es la representada en el código 5.16.[27]

Nodo de entrada	Función	Figura
ColorCamera	Obtención de imágen RGB	5.24
MonoCamera	Obtención de imágen en escala de grises, un nodo para cada cámara estéreo de los extremos	5.28
StereoDepth	Obtención de la imágen de profundidad a partir de dos imágenes estéreo con diferentes puntos de vista.	5.27

Tabla 5.5: Lista de nodos usados para el procesamiento de imágenes

código 5.16: Creación de nodos de entrada

```
#Nodo de procesamiento de entrada RGB
self.colorCam = self.pipeline.create(dai.node.ColorCamera)
#Nodo de procesamiento de salida RGB
self.xoutRgb = self.pipeline.create(dai.node.XLinkOut)
```

Una vez instanciados todos los nodos de procesamiento, el siguiente paso es asignar el nombre para la salida de la cámara rgb y otro para el pár esteréo como se observa en el código 5.17.

código 5.17: Un *Stream* para cada salida

```
#Streams
self.xout.setStreamName("disparity")
```

```
3 self.xoutRgb.setStreamName('rgb')
```

Posteriormente, se asignan las propiedades de cada cámara. Para procesar las imágenes de forma correcta es necesario llevar a cabo:

- Ajuste de la resolución de cada cámara: 400p para las cámaras *mono* y 1080p para la cámara *RGB*.
- Configuración de la calidad: modos de densidad, precisión y filtrados.
- Configuración del nodo de profundidad mediante filtros.

A continuación se enlazan los nodos de procesamiento de entrada con los de salida como se observa en el código 5.18.

código 5.18: Entrelazado de nodos

```
1 #Disparidad
2 self.monoLeft.out.link(self.depth.left)
3 self.monoRight.out.link(self.depth.right)
4 self.depth.disparity.link(self.xout.input)
5 #RGB
6 self.colorCam.video.link(self.xoutRgb.input)
```

Una vez enlazados y configurados todos los nodos, se crearán los *topics* encargados de publicar las imágenes, que es la finalidad del nodo de ROS2 actual. Se crean los siguientes publicadores:

código 5.19: Publicadores de imágenes

```
1 #Publicadores de imagenes
2 self.rgb_pub = self.create_publisher(Image, '/camera/rgb/image_raw', 10)
3 self.disparity_pub = self.create_publisher(Image, '/camera/depth/image_raw'
      , 10)
4 self.rgb_info_pub = self.create_publisher(CameraInfo, '/camera/rgb/
      camera_info', 10)
```

Seguidamente, se crea la función *frame_grabber()*, dentro de la cual se terminará el procesamiento final. Se conecta el dispositivo de la cámara mediante una clase propia de *DepthAI* y se crean dos colas para gestionar los datos recibidos de cada nodo de salida, como se observa en el código 5.20.

código 5.20: Colas de gestión de datos

```
1 with dai.Device(self.pipeline) as device:
2     #Colas para obtener los frames
3     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
4     qDepth = device.getOutputQueue(name="disparity", maxSize=4, blocking=
          False)
5     #Almacenamiento de las colas
6     inRgb = qRgb.tryGet()
7     inDisparity = qDepth.tryGet()
```

Se crean mensajes de tipo imágen de *ros* convirtiendo su formato desde OpenCV. En consecuencia, una vez subida la información a los *topics* mencionados, se podrán analizar los datos con el nodo de detección y obtener la posición exacta del objeto de forma limpia. Por último, se capturan los fotogramas en tiempo real, y se publican en los *topics* correspondientes a los publicadores creados como se observa en el código 5.21.

código 5.21: Publicación de imágenes con formato ros

```
if inDisparity is not None:
                disp_frame = inDisparity.getCvFrame()
                #Conversion de formato
                disp_frame_ros = self.bridge.cv2_to_imgmsg(disp_frame,'
                    mono8')
                disp_frame_ros.header.stamp = now
                #Publicacion de imagen
                self.disparity_pub.publish(disp_frame_ros)
```

La conversión de formato de las imágenes de *OpenCV* a *ros* y viceversa se ha realizado gracias a la librería CvBridge facilitando así el procesamiento de las imágenes. Su funcionamiento ha sido ilustrado en la figura 5.23.

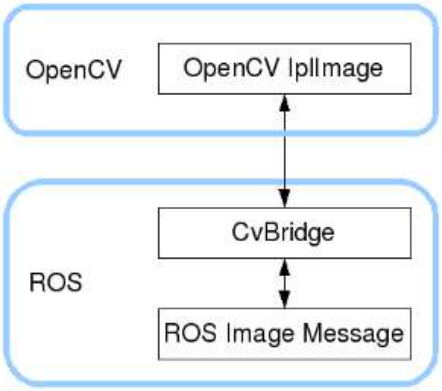


Figura 5.23: CvBridge de ROS [8]

5.5.2. Obtención de la posición en el plano horizontal

Uno de los objetivos principales de este proyecto es determinar la ubicación exacta de las piezas para poder manipularlas, así como su comunicación con el sistema de control del robot. Para ello, se necesitarán las tres dimensiones de las piezas.

Las dos primeras dimensiones 'x' e 'y', consideradas como el plano horizontal, se han adquirido a través de las imágenes proporcionadas por la cámara RGB. En esta tarea se ha hecho uso del nodo *ColorCamera* dada su capacidad para obtener la imágen *red-green-blue* de 24 bits, como se puede observar en la figura 5.24. [27]

En primer lugar, se obtiene la imágen de color a través del *topic /camera/rgb/image_raw* y se efectúa su análisis mediante la librería OpenCV, vista en el apartado 5.1.3.1. El primer paso es llevar a cabo la suscripción al *topic* asociándolo a una función denominada *detect_object()* para el reconocimiento de las piezas y su cálculo de posición necesario. Se puede observa en el código 5.22.

código 5.22: Suscripción al topic /camera/rgb/image_raw

```
self.subscription = self.create_subscription(
    Image,
```

[8] http://wiki.ros.org/cv_bridge
[9] https://docs.luxonis.com/en/latest/

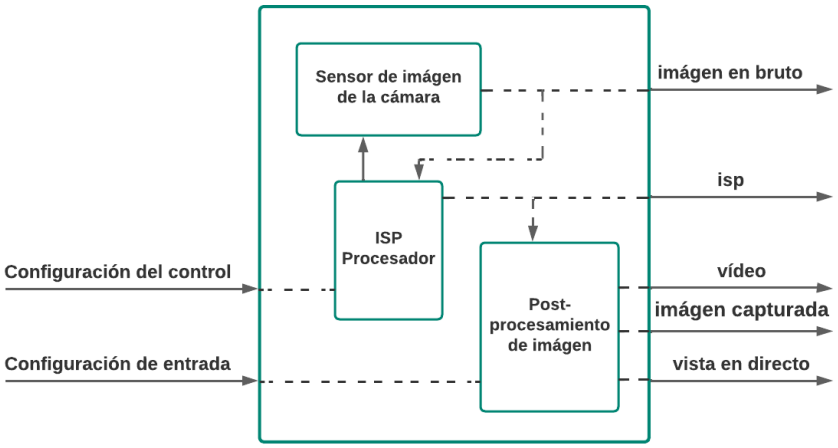


Figura 5.24: Nodo *ColorCamera* [9]

```
3     '/camera/rgb/image_raw',
4     self.detect_object,
5     10)
```

A continuación, dentro de la función mencionada, se convierte de nuevo la imagen a formato OpenCV para su análisis y se transforma el formato RGB en HSV[10]. Esta disposición es más atractiva de cara al análisis de imágenes porque separa el tono de la imágen de su intensidad y brillo característico, como se ve en el código 5.23.

código 5.23: Conversión de formatos

```
1 cv_image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
2 hsv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
```

Seguidamente, se genera una máscara binaria para poder detectar colores específicos ampliando así, el abanico de posibilidades de las aplicaciones de Pick and Place desarrolladas. Y, se filtra la imágen aplicando dicha máscara, como ilustra el código 5.24. Cabe destacar que el valor de estas máscaras binarias dependen de la selección de colores elegida en la interfaz de usuario.

código 5.24: Filtrado mediante máscara binaria

```
1 #Aplicacion de mascara binaria a la imagen HSV
2 mask = cv2.inRange(hsv_image, green_lower, green_upper)
3 filtered_image = cv2.bitwise_and(cv_image, cv_image, mask=mask)
```

La determinación de la posición en dos dimensiones descrita se llevará a cabo en tres pasos:

1. Metodología de detección de figuras.

   La metodología utilizada para detectar las piezas ha sido a través de la detección de sus contornos, basada en procedimientos de *OpenCV* usando el color y tamaño fijo de los mismos. Y, en último lugar, se obtienen sus posiciones mediante varios algoritmos matemáticos como se muestra en el código 5.25.

[10]HSV:Hue, Saturation, Value. Matiz, Saturación, Valor

código 5.25: Reconocimiento por contornos

```
#Reconocimiento de contornos
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour) #Objetos rectangulares
        if (w > 60 and h > 100 and w < 100 and h < 250):
            cv2.rectangle(cv_image, (x, y), (x+w, y+h), (0, 255, 0),
                2) #Dibujo del contorno
            #Posicion del centro del objeto en pixeles
            self.centro_x = x - w/2
            self.centro_y = y + h/2
```

El resultado final de la detección y coloreado de contornos es el que se observa en la figura 5.25.

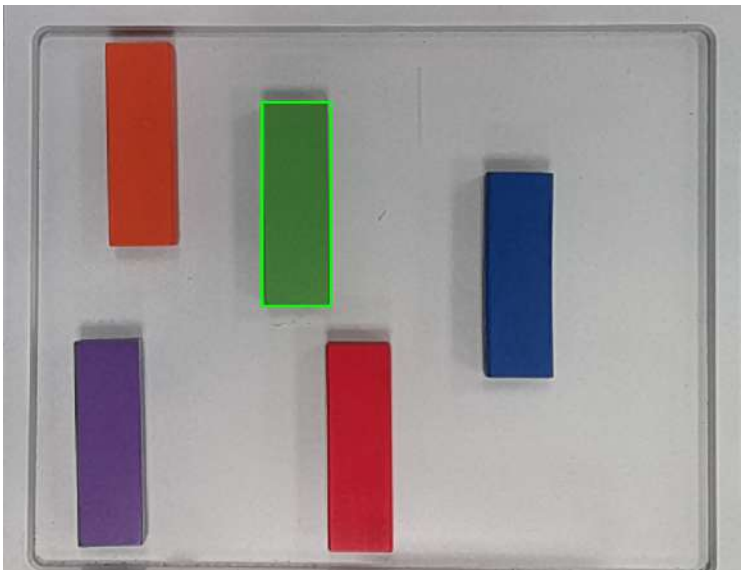


Figura 5.25: Detección de objetos por color

2. Conversión de píxeles a centímetros.

   Este procedimiento es necesario para poder convertir la posición horizontal obtenida a la deseada por el robot. Se ha llevado a cabo de forma manual con el uso de un sistema de medida tan sencillo como una regla métrica y una pequeña lógica implementada dentro del nodo de detección de la cámara. Se ha fijado una regla métrica en el espacio de trabajo, y se ha medido en tiempo real cuántos píxeles en la imágen de la cámara equivalen a 25 centímetros, obteniendo así un factor de escala preciso y eficaz. En la figura 5.26 se puede visualizar este proceso, observando la salida por pantalla del programa que determina el número de píxeles equivalentes a esta distancia fija.

   El algoritmo del código 5.26, ha utilizado dos puntos dibujados mediante OpenCV, ajustados a los 0 y 25 centímetros deseados, obteniendo así el equivalente en píxeles mediante una simple resta.

código 5.26: Algoritmo conversión pixeles-cm

```
  #Puntos obtenidos de forma experimental mediante la camara
```

Figura 5.26: Metódo de conversión píxeles-cm

```
  x2, y2, x1, y1= = 589, 367, 1210, 367
  cv2.circle(cv_image, (x2, y2), 4, (255, 255, 255), 2) #1 Punto
  cv2.circle(cv_image, (x1, y1), 4, (255, 255, 255), 2) #2 Punto
  #Numero de pixeles equivalentes
  distancia = abs(x2 - x1)
```

3. Obtención de la posición final.

   Para finalizar la obtención de la posición horizontal, se aplica el factor de escala obtenido de píxeles a centímetros y, además, se aplica una calibración relativa para ajustar el sistema de referencia de la cámara al sistema de referencia del robot. Todo este procedimiento se puede observar en el código 5.27. Las posiciones finales son *centro_robot_mm_x* y *centro_robot_mm_y*.

código 5.27: Conversiones finales

```
#Conversion pixeles-cm
factor_escala = 25 / 621 #cm / pixeles
centro_cm_x = factor_escala * self.centro_x_24bits
centro_cm_y = factor_escala * self.centro_y_24bits

#Ajuste del sistema de referencia camara-robot (mm)
centro_robot_mm_x =  - 540 + centro_cm_x * 10
centro_robot_mm_y =  - 158  -  centro_cm_y * 10

#Relacion de escala entre la imagen de 8 y 24 bits
src_width, src_height, dst_width, dst_height = 1920, 1080, 640, 400
scale_x = dst_width / src_width
scale_y = dst_height / src_height

#Posicion en la imagen de 8 bits
self.dst_x = int(src_x * scale_x)
self.dst_y = int(src_y * scale_y)
```

Adicionalmente, cabe mencionar que se calcula la posición equivalente para la imágen de 8 bits con el objetivo de obtener la dimensión ‘z’ restante a partir del conocimiento de su posición en el plano horizontal.

5.5.3. Obtención de la posición en el plano vertical

La obtención de la ubicación tridimensional completa de los objetos se completa obteniendo el valor de 'z' definido como el plano vertical, puesto que se define como la distancia entre la cámara y la pieza.

Teniendo en cuenta que la cámara se encuentra encima del robot, se necesita conocer la altura del objeto a través de una distancia que será la profundidad hasta el mismo con los correspondientes ajustes relativos. Su procedimiento hace uso de los siguientes nodos de procesamiento:

1. En primer lugar, existe un nodo denominado *StereoDepth*, proporcionado de igual forma por *DepthAI*, idóneo para esta tarea. Este se utiliza para calcular la profundidad con el uso del concepto de disparidad mencionado en el apartado 3.5.1. El objetivo es crear un mapa de disparidad del cual se obtendrá de forma precisa el valor de profundidad en cualquier punto del mapa a partir de un algoritmo.[27]

   Este nodo de *DepthAI* tiene la configuración ilustrada en la figura 5.27.

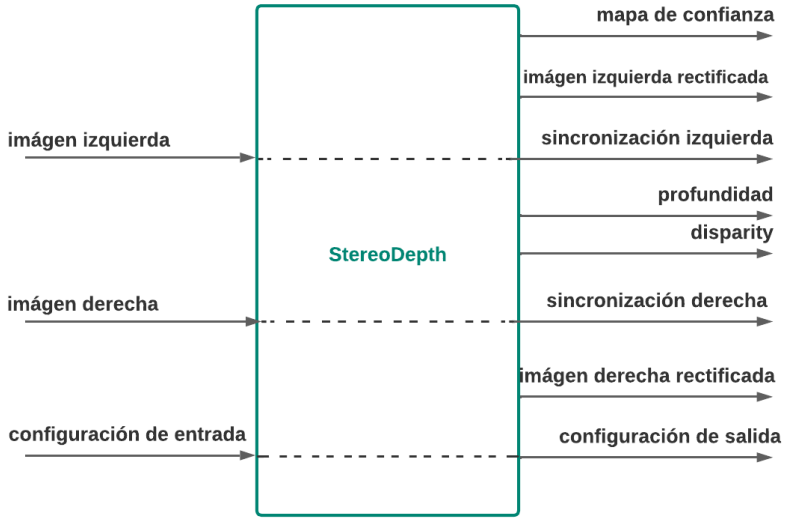


Figura 5.27: Nodo *StereoDepth* [9]

Además, se han recogido los valores de entrada y salida de este nodo en la tabla 5.6.

Entrada/Salida	Nombre	Función
Entrada	left right	Imágenes generadas por ambas cámaras de los extremos en blanco y negro 5.28
Entrada	inputConfig	Una lista extensa de valores de configuración como la densidad y resolución o diferentes parámetros de post-procesamiento necesarios para extraer información. 5.28
Salida	Disparity	Mapa de disparidad para la obtención de la profundidad

Tabla 5.6: Entradas y salidas del nodo StereoDepth

2. Por otro lado, cabe mencionar que las imágenes generadas por las cámaras de los extremos han sido obtenidas gracias al nodo *MonoCamera* de *DepthAI*, que genera una imagen con formato monocromático mediante una escala de grises de forma

sencilla. Esta imágen será muy útil dado que a la hora de visualizar el mapa de disparidad aumentará la eficiencia para medir distancias en cualquier punto. Se puede observar su simple funcionamiento en la figura 5.28.

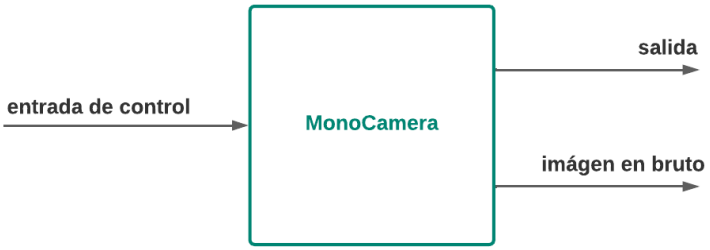


Figura 5.28: Nodo *MonoCamera* [9]

Una vez conocidos los nodos utilizados para medir el plano vertical, el proceso seguido para la implementación con este mapa de disparidad se ilustra en la figura 5.29 y sigue los siguientes pasos:

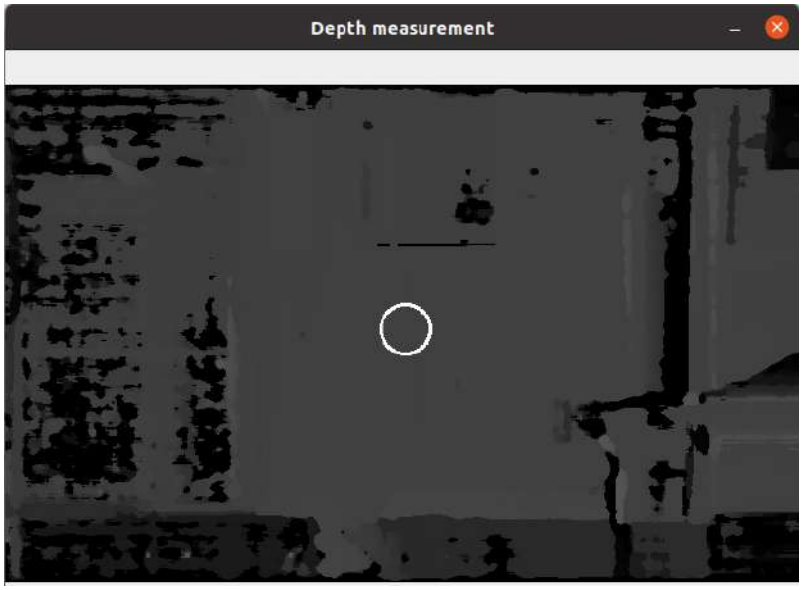


Figura 5.29: Mapa de disparidad

1. Generación del mapa de disparidad mediante una escala de grises. El mapa de disparidad se obtiene creando una suscripción del *topic* denominado */camera/depth/image_raw* y asociando una función nombrada *def_depth()* para obtener el resultado final. Se puede observar en el código 5.28.

código 5.28: Suscriptor del topic /camera/depth/image_raw

```
self.subs_disparity = self.create_subscription(
    Image,
    '/camera/depth/image_raw',
    self.def_depth,
    10)
```

2. Calibración de la distancia focal y ajuste necesario de ambas cámaras, realizado en el nodo de procesamiento de imágenes.

3. Cálculo de la distancia al objeto mediante la disparidad.

   Se crea una función auxiliar donde se usa una función típica de *OpenCV* para obtener el valor de disparidad en un punto concreto y mediante el siguiente algoritmo matemático se obtiene la distancia en centímetros:

$$\text{Distancia} = \frac{\text{Linea base} \cdot \text{Distancia focal}}{\text{Valor de disparidad}}$$

   Donde:

   - Distancia: distancia final en centímetros.
   - Línea_base: distancia física entre ambas cámaras disponible en la documentación del fabricante como en la figura 4.10.
   - Distancia focal: distancia entre el punto focal y el centro óptico de la lente.
   - Valor de disparidad: valor obtenido en la función implementada en el código 5.30.

   Dicha función auxiliar se puede visualizar en el código 5.29.

código 5.29: Distancia desde la disparidad

```
def get_distance_from_disparity(disparity_image, x, y):
    #Parametros de la camara -> Importante calibracion previa
    baseline = 7.5 #en cm
    focal_length =  451.144 #calibrada en pixeles

    # Calcula la distancia a partir de la disparidad
    disparity_value = disparity_image[y,x] #Funcion de OpenCV
    distance = (baseline * focal_length) / disparity_value

    return distance
```

En segundo y último lugar, se implementa la función *def_depth()* cerrando el ciclo de la tarea descrita con un filtro de medidas de profundidad en centímetros como se ve en el código 5.30. Esta convierte la imágen del topic en formato *OpenCV* para poder aplicar la función auxiliar descrita y obtener así la distancia exacta en el centro del objeto, rodeando el centro como ejecución final.

código 5.30: Obtención de z en centímetros

```
#Conversion de formato a OpenCV
cv_image = self.bridge.imgmsg_to_cv2(msg, 'mono8')
#Uso de la funcion auxiliar para la obtencion de la profundidad
depth_val = get_distance_from_disparity(cv_image,x,y)
cv2.circle(cv_image, (x, y), 20, (255, 255, 255), 2)
#Visualizacion de la imagen
cv2.imshow('Depth measurement', cv_image)
cv2.waitKey(3)
```

5.5.4. Resultado final

Finalmente, gracias a la obtención de ambos planos, horizontal y vertical, se puede formar una posición tridimensional dentro del nodo de detección a través de la función del código 5.31 y publicar la posición en el *topic /object_position* comunicándose con el control del robot.

código 5.31: Publicación de la posicion al nodo de control

```
def publish_news(self):

    msg = Pose()

    msg.position.x = self.position_x_to_robot * 0.001
    msg.position.y = self.position_y_to_robot * 0.001
    msg.position.z = self.position_z_to_robot * 0.001

    msg.orientation.x = self.quaternion[0]
    msg.orientation.y = self.quaternion[1]
    msg.orientation.z = self.quaternion[2]
    msg.orientation.w = self.quaternion[3]

    self.publisher_.publish(msg)
```

5.5.4.1. Alineación de los sistemas de referencia

Para la posición final del *topic* mencionado, es necesario realizar previamente, la alineación del sistema de referencia de la cámara con el del robot. Las características técnicas necesarias para este apartado, han sido extraídas del documento [24].

Este método puede realizarse de múltiples formas, pero, en este caso, se ha aprovechado que la cámara se encuentra paralela a la mesa de trabajo, convirtiendo este procedimiento en un proceso realmente sencillo. En general, la calibración entre sistemas de referencia se lleva a cabo mediante la creación de una matriz de transformación tridimensional. Este proceso es común a todos los métodos, y en el caso presente, la matriz de 4x4 dimensiones se verá reducida dado que la orientación de la herramienta se encuentra fija en la mayor parte de este proyecto.

La matriz de transformación tiene la siguiente forma:

$$\begin{bmatrix} a11 & a12 & a13 & Tx \\ a21 & a22 & a23 & Ty \\ a31 & a32 & a33 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde sus elementos son:

- aXX: coeficientes de rotación.
- Tx, Ty, Tz: coeficientes de traslación.

En este caso, los términos de rotación son nulos debido a que la orientación siempre será la misma y los términos de traslación se calculan de la siguiente manera:

1. Se visualiza la imágen que proporciona la cámara y se ubica el punto (0,0) de la misma.

2. Se coloca manualmente el robot en dicha posición y se anota su posición tridimensional.
3. Tx y Ty son la diferencia relativa entre la posición (0,0) de la cámara y la posición actual del robot. Asimismo, Tz será la distancia directa de la cámara a la mesa de trabajo.

Finalmente, la matriz quedaría de la siguiente forma:

$$\begin{bmatrix} 1 & 0 & 0 & Dx \\ 0 & 1 & 0 & Dy \\ 0 & 0 & 1 & Dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.6. Implementación de la interfaz gráfica de usuario

La interfaz gráfica de usuario ha sido la clave fundamental para unificar el proyecto con una usabilidad e interacción eficaz y sencilla, así como con una comunicación necesaria con las demás partes del proyecto.

En referencia a la parte técnica de su creación, gracias a la librería *Tkinter*, se ha diseñado una comunicación dentro del nodo de la interfaz con botones, imágenes y textos a modo de indicaciones, todos estos configurados de forma intuitiva. Su comunicación específica se puede observar en el diagrama de nodos de la figura 5.30.

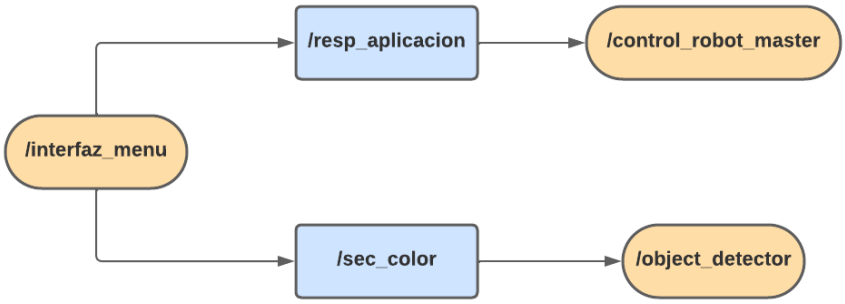


Figura 5.30: Diagrama de nodos de la interfaz

Dentro del nodo denominado *interfaz_menu*, se han creado dos publicadores como se observa en el código 5.32: uno para el tipo de caso de uso y otro para la secuencia de colores que establece el orden de los objetos a detectar por la cámara.

código 5.32: Publicadores de la interfaz

```
1 self.publisher_resp = self.create_publisher(Int8, "resp_aplicacion", 10)
2 self.publisher_color = self.create_publisher(String, "sec_color", 10)
```

Se han declarado varias funciones que manejan los datos a publicar, de forma que cuando se pulsa un botón, el valor de los datos se actualizan y, si cumplen los requisitos, se publican. Las funciones son de dos tipos:

- Colores. El código 5.33 es un ejemplo de las funciones creadas para cada color en el cual si se han introducido ya los cuatro colores establecidos por defecto, se

publicará un *string* que contenga la inicial de cada color. Posteriormente, el nodo de detección se suscribirá a este obteniendo la secuencia que necesita para empezar su funcionamiento.

código 5.33: Función para los botones de la secuencia de colores

```
def color_azul(self):
    self.sec_color.data += "a"
    if len(self.sec_color.data) == 4:
        self.publisher_color.publish(self.sec_color)
```

- Casos de uso. Por otro lado, el código 5.34 es otro ejemplo de las funciones asignadas a los botones de los casos de uso. Si se pulsa dicho botón se publicará un valor del uno al tres, estableciendo el tipo de aplicación que se va a realizar, el cual será recibido por el nodo de control.

código 5.34: Función para los botones de casos de uso

```
msg = Int8()
msg.data = 3
self.publisher_resp.publish(msg)
```

Una vez seleccionado ambos tipos, comenzaría el flujo de operación de las aplicaciones de Pick and Place.

Finalmente, la apariencia de los *topics* en los que se publican se ven reflejados en la figura 5.31.

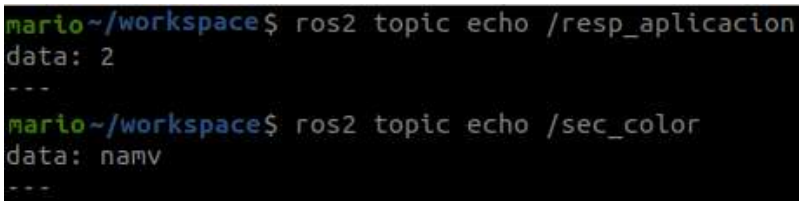


Figura 5.31: Topics de comunicación de la interfaz

Una vez conocida la parte técnica de la interfaz de usuario, se va a explicar la interacción con el usuario en dos pasos fundamentales:

1. Se selecciona la secuencia de colores de las piezas haciendo un click por cada una. Las aplicaciones implementadas están creadas para cuatro piezas por defecto, por lo que en este caso se debería pulsar cuatro colores de forma secuencial como en la figura 5.32 en el orden elegido.

Figura 5.32: Selección de secuencia de colores

2. Se selecciona el tipo de aplicación que se desee con un solo click en uno de los botones de la figura 5.33.

Figura 5.33: Selección del tipo de aplicación

Capítulo 6

Resultados y discusión

En este capítulo se expone la implementación final del desarrollo del proyecto integrado por completo, incluyendo pruebas realizadas del mismo. Se raliza un análisis exhaustivo de los resultados obtenidos y de los impactos supuestos por el proyecto.

6.1. Resultados

6.1.1. Flujo normal de funcionamiento

En una primera instancia, se presentan los pasos principales para el correcto funcionamiento de la implementación final del proyecto, los cuales se han reflejado en la figura 6.1.

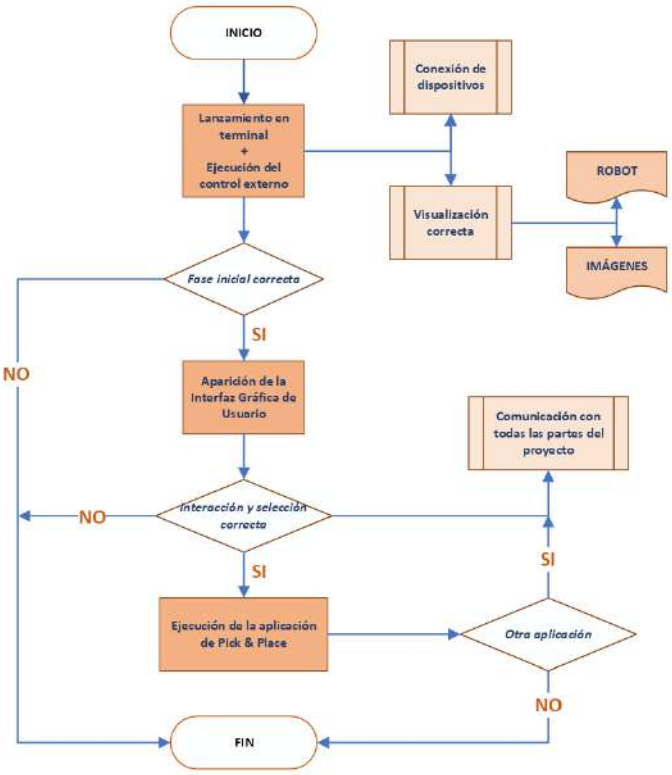


Figura 6.1: Flujograma de alto nivel de la integración final

6.1.2. Casos de uso

A continuación, se van a mostrar los escenarios inicial y final de cada caso de uso, así como su función básica:

- Orden de piezas. El propósito de esta aplicación es recoger del espacio inicial las cuatro piezas en el orden seleccionado por el usuario y depositarlas, según este orden, en una ubicación específica que simule un almacén como en la figura 6.7. Su implementación es la siguiente:

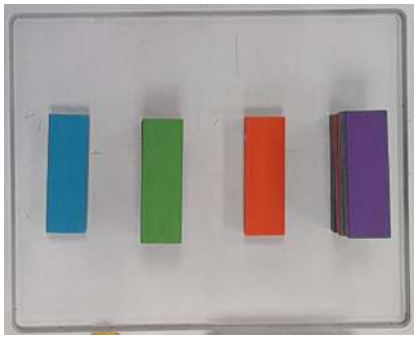


Figura 6.2: Escenario inicial del orden de piezas

Figura 6.3: Escenario final del orden de piezas

- Paletizado de piezas. El objetivo de esta aplicación es recoger de igual forma las piezas seleccionadas y realizar posteriormente, un paletizado de las mismas en un espacio seleccionado para ello con ubicaciones fijas. Debe verse de la siguiente forma:

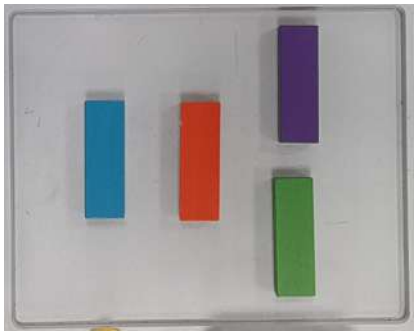


Figura 6.4: Escenario inicial del paletizado

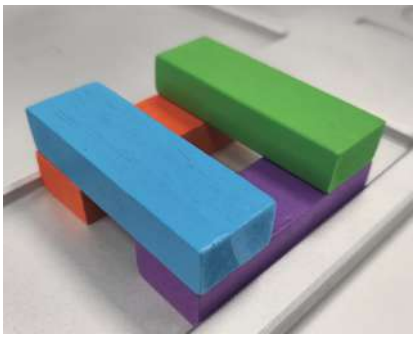


Figura 6.5: Escenario final del paletizado

- Despaletizado de piezas. La última aplicación ofrece la posibilidad de despaletizar las piezas después de haber realizado la anterior aplicación. En este caso, la cámara detecta las piezas por orden de igual manera, y las va depositando en el espacio de trabajo, con el objetivo de preparar el espacio para realizar otra posible aplicación. Su resultado es el que se observa a continuación:

6.2. Discusión

Una vez introducida la ejecución final de las aplicaciones, así como sus escenarios, los resultados obtenidos que se van a presentar a continuación analizan la efectividad de

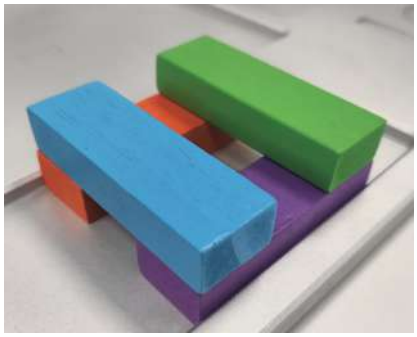


Figura 6.6: Escenario inicial del despaletizado

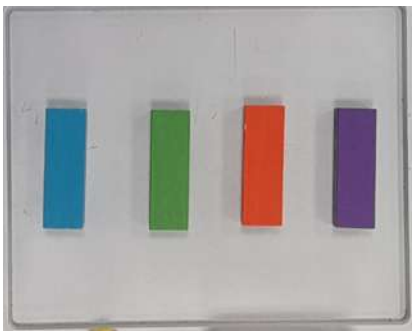


Figura 6.7: Escenario final del despaletizado

la integración final del proyecto, así como el acierto y las limitaciones de este. Se han completado las tres aplicaciones propuestas de Pick and Place con una ejecución exitosa en cada una de sus partes: visión artificial, control del robot e interacción con el usuario.

En cuánto al control del robot, se ha obtenido una ejecución muy positiva, dado que se planifican y ejecutan trayectorias de forma correcta con múltiples posibilidades disponibles. Se ha probado una precisión muy cerca del 100 % de acierto tanto en trayectorias cartesianas como libres, en cada uno de los casos de uso. La única limitación de estas ha sido la posición de la cámara, ya que estas trayectorias nunca podían ser planificadas cerca de su ubicación, superior al espacio de trabajo. Por otro lado, el uso de su herramienta también ha concluido en un resultado positivo, debido a que gracias al servicio implementado, se abre y cierra la pinza en función de la necesidad con un completo acierto.

Respecto a la visión artificial, se ha logrado hacer uso del reconocimiento de objetos y cálculo de posición de forma precisa. Se ha obtenido un resultado realmente favorable ya que se ha conseguido otro de los objetivos iniciales, realizar estas aplicaciones de forma autónoma con una simple indicación de alto nivel. La localización de los objetos ha sido clave en este proyecto, ha supuesto ser una de las partes más complejas a nivel técnico debido a todos los factores necesarios a tener en cuenta. A continuación, se realiza el análisis de la detección completa dividida en dos partes: el reconocimiento en el eje vertical y en el eje horizontal.

- Por un lado, la obtención de las posiciones horizontales es muy acertada, ya que la metodología utilizada es precisa y efectiva. El resultado ha sido muy exacto para la mayoría del espacio de recogida de piezas, con una efectividad del 82 %. La única limitación ha sido el efecto de la conversión de píxeles a centímetros, debido al factor distancia de la cámara al objeto. Los objetos que se encuentran cerca de la vertical principal de la cámara son reconocidos a la perfección, pero aquellos que se alejan bastante dentro del espacio de trabajo de piezas pueden tener alguna variación. Esto se debe a que el área reconocida del objeto no es el mismo para ambas situaciones y la conversión de píxeles a centímetros es fija.
- Por otro lado, el análisis de la obtención de la posición vertical del objeto se ha separado en dos casos: aquellos objetos cercanos a la mesa de trabajo y aquellos situados en posiciones más lejanas.

    a) Los objetos situados cerca de la mesa de trabajo con una altura que difiere poco en la original medida por la cámara, han logrado un resultado muy positivo como se puede ver en la figura 6.8. Esta gráfica lineal representa el error entre

los valores obtenido y deseado. Se obtiene un acierto del 75 %, en el cual el 25 % restante, sólo contenía un 10 % de medidas erróneas significativas que fueron corregidas en la programación con varias condiciones adicionales.

b) Los objetos más alejados de la mesa de trabajo han obtenido un peor resultado con un 60 % de medidas exactas, pero las demás medidas no han discrepado tanto del valor esperado, obteniendo un error promedio bajo. Se puede visualizar en la gráfica de dispersión de la figura 6.9.

Como sintesis, se ha comprobado que la medición de profundidad mejora cuando supera los 60 centímetros aproximadamente.

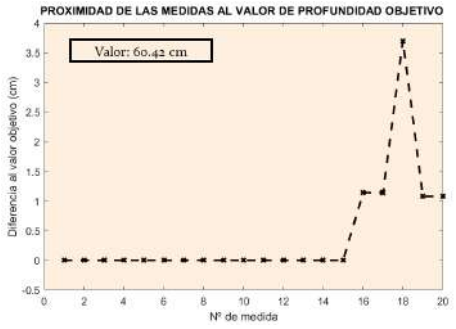


Figura 6.8: Gráfica lineal para objetos cerca del tablero

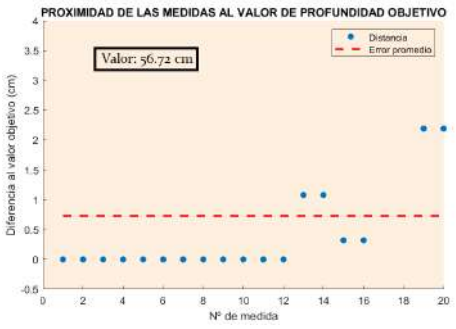


Figura 6.9: Gráfica lineal para objetos lejos del tablero

En lo que concierne a la interacción con el usuario, se ha cumplido con otro de los objetivos principales que era diseñar una interfaz gráfica para poder hacer uso del proyecto sin necesidad de poseer aspectos técnicos sobre este. Por lo que, el uso de la interfaz gráfica ha sido un éxito por su perfecta comunicación con las diferentes partes del proyecto. Ha sido muy importante para la integración final y su resultado ha resultado muy constructivo en términos de temporización.

Finalmente, se ha hecho un estudio de la cantidad de aciertos por piezas en caso de uso, el cual engloba todas las partes del proyecto. Las piezas son manipuladas de forma éxitosa para dar credibilidad a este análisis de resultados. Se puede visualizar en la figura 6.10 el número de aciertos para una secuencia de cuatro piezas.

Figura 6.10: Diagrama de barras en representación del acierto de piezas

6.3. Evaluación de impactos

Una vez analizado el uso de las aplicaciones de Pick and Place implementadas mediante soluciones robotizadas y automatizadas, se puede extraer que, a día de hoy, supone una gran mejora a nivel industrial, prestando además, múltiples beneficios sociales con una gran presencia sostenible a escala mundial. Se va a analizar su impacto en los siguientes aspectos:

1. En cuánto al impacto social, se puede afirmar que representan una mejora de las condiciones laborales para la mayoría de industrias gracias a las ventajas mencionadas y referenciadas en el apartado 2.3.1. Una de las más importantes sería el aumento de calidad del día a día de aquellos trabajadores. Hace unos años realizaban tareas repetitivas con cargas que les exponían a situaciones no favorables. Además, el fácil uso de este tipo de aplicaciones ha hecho que personas con cualificaciones básicas puedan optar a este tipo de empleos de una forma segura, ya que, además, cuentan con sistemas de protección para los entornos de trabajo.

   Por último, es importante añadir que este campo ha despertado frentes de innovación e investigación aportando un campo lleno de oportunidades laborales, así como el futuro de muchas operaciones actuales.

2. Respecto al impacto medioambiental de la automatización de estas aplicaciones, mejoran la sostenibilidad debido a un mejor aprovechamiento de los recursos existentes y a una reducción de efectos negativos. [33]

   Algunos de los cambios más notables son:

   - Reducción de residuos. La precisión y optimización que suponen, ha resultado en una mejor gestión de la cantidad de materias primas utilizadas en las industrias, manejando los materiales de forma más exacta. Además, las aplicaciones de Pick and Place pueden destinarse a la separación de residuos y materiales orgánicos.
   - Ahorro de energía. Su optimización hace que consuman menos energía en comparación con los sistemas tradicionales en antiguos procesos de producción. [33]

3. Finalmente, el aspecto económico es uno de los más beneficiados de la automatización de este tipo de tareas ya que, como se mencionó en el apartado 2.3.1, suponen un aumento de la productividad a un coste y tiempo menor. Optimizan el proceso de producción por completo obteniendo un mayor rendimiento y una competitividad a nivel laboral muy positiva, que desemboca en precios ambiciosos. [16]

   Finalmente, promueven nuevos empleos para todo tipo de públicos, además, de ser consideradas inversiones de larga durabilidad, ya que estas tecnologías cuentan con cuidados y protecciones especiales. [6]

Capítulo 7

Gestión del proyecto

En este capítulo se describe la gestión del proyecto con el ánimo de ilustrar las diferentes fases del mismo de forma cronológica, así como la planificación de cada una de estas. Asimismo, incluye el presupuesto global del proyecto.

7.1. Ciclo de vida

En cuánto al ciclo de vida del proyecto, las fases del proyecto realizado han sido las siguientes:

- Fase inicial. Incluye la fijación de requisitos y propuestas del trabajo a diseñar, así como la familiarización de los entornos con los que se va a desarrollar.
- Planificación e Investigación. Abarca toda la búsqueda del estado actual del proyecto a desarrollar y la selección de todos los recursos necesarios para llevarlo a cabo.
- Control del robot. Contiene todo el uso de controladores del robot en ROS2, así como el estudio de su propio funcionamiento. Se investiga su movimiento mediante trayectorias, así como el uso de su herramienta.
- Aprendizaje e implementación de la visión por computador. Comprende toda la investigación de la parte de visión en cuanto a procesamiento, reconocimiento y ajuste de sistemas de referencia.
- Creación de una interfaz de usuario. Se crean todas las comunicaciones necesarias para unificar el proyecto mediante dicha interfaz.
- Integración. Engloba todo el proyecto, uniendo las tres fases anteriores, creando así las aplicaciones finales.
- Pruebas y conclusiones. Comprende todas las puestas en marcha, calibraciones y ajustes. Por último, esta fase final cierra el ciclo con el fin de la documentación y conclusiones del proyecto.

7.2. Planificación

A continación, se ha creado un diagrama de Gantt como se observa en la figura 7.1, indicando cada etapa del mismo cronológicamente para terminar de completar el ciclo de vida del proyecto en detalle.

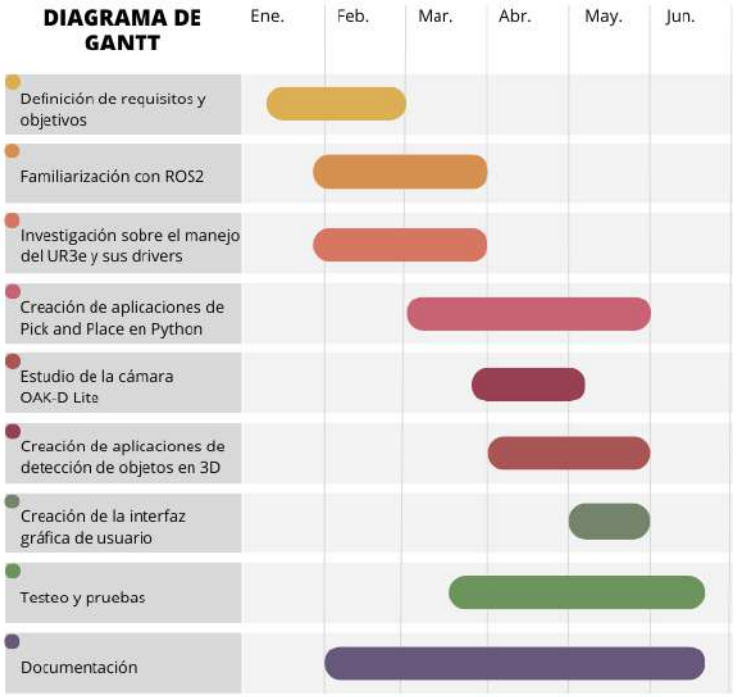


Figura 7.1: Diagrama de Gantt

7.3. Presupuesto

En este apartado se muestra el presupuesto desglosado del proyecto creado y está disponible en la tabla 7.1. Por un lado, se muestran todos los componentes físicos con su coste y número de unidades necesarios. Por otro lado, se ha recogido el coste de la mano de obra que supondría este proyecto para un Ingeniero Junior de Software. En suma, el presupuesto final aproximado es de 24.686,73 euros.

Componentes	Precio €	Unidades	Marca	Precio final
Robot colaborativo UR3e	17545,00	1	UR	
Herramienta HRC-03	1924,00	1	Zimmer	
Cámara ODK-D-Lite-AF	149,00	1	Luxonis	
Trípode ligero	28,77	1	Amazon Basics	
Cables Ethernet Cable USB-C	5,99 6,99	3 1	Akake INIU	
Madera Tornillos/Tuercas Pieza hecha en impresión 3D con PLC	15,00	70x5cm 6/6 10 1	Elaboración propia	
				19.686,73 €

Mano de obra	Oficio	Precio € / hora	Nº horas	Precio final
Propia	Ingeniero Junior de Software	10	500 horas	5000 €

Tabla 7.1: Presupuesto del proyecto

Capítulo 8

Conclusiones

El propósito principal de este proyecto ha sido implementar diferentes aplicaciones de Pick and Place de alto nivel disponibles para todos los públicos con la mayor autonomía posible gracias a la visión artificial y a la programación en ROS2.

El estudio de investigación realizado acerca de la programación en ROS2 orientada a robots colaborativos, visión artificial en aplicaciones de Pick and Place y el uso de ciertas librerías en Python, ha concluido éxitosamente. Como resultado, se han afianzado óptimamente los conocimientos como para integrarlos de forma unificada. Además, se ha estudiado el impacto de cada una de estas tecnologías adquiriendo una visión del proyecto que va más allá de un estudio académico.

Como relevancia, para llevar a cabo la implementación final descrita del proyecto, se han utilizado muchas otras metodologías que, finalmente, no se incluyeron. Sin embargo, esta investigación adicional ha sumado un gran valor al proyecto, ampliando el abanico de campos investigados en el mismo.

Por otra parte, la creación y puesta en marcha de cada una de estas partes han presentado numerosas dificultades, habiendo hecho de todo el ciclo de vida del proyecto un continuo proceso de formación.

La gran cantidad de información existente acerca del robot colaborativo usado ha llevado a realizar una síntesis de información. Se han adquirido claves importantes para poder crear un control del robot sencillo y eficiente, llevándolo a cabo de manera fructífera. Asimismo, el uso de la cámara tridimensional, mediante visión artificial, ha supuesto un estudio de metodologías tanto técnicas como del propio modelo de la cámara. Este proceso se ha llevado a cabo de manera satisfactoria, ya que según el análisis realizado de esta, su funcionalidad es muy positiva y ha logrado ser una pieza fundamental para la autonomía propuesta de estas aplicaciones.

Por último, se ha construir una base consistente para que cualquier persona pueda usar estas aplicaciones siguiendo el manual de usuario propuesto en el apéndice A.1. Esta base se refiere a la interacción entre el control del robot y el usuario a través de la interfaz gráfica creada.Además, se ha compactado todo el proyecto en un solo archivo de lanzamiento, a pesar de incluir multitud de lanzamientos complejos, logrando una óptima sincronización entre estos.

No obstante, se han identificado ciertas limitaciones del proyecto:

- La planificación de trayectorias debido al espacio de trabajo restringido por la posición de la cámara.
- La restricción fija que posee la pinza por el método de calibración implementado.

A pesar de esto, se han solucionado con resultados muy favorables y se han observado posibles mejoras a sugerir para implementaciones futuras.

Por último, se ha logrado realizar el proyecto con una planificación e implicación excepcional, integrando así cada componente a su debido tiempo en función de las prioridades del proyecto, consiguiendo finalmente el desarrollo de todas ellas. Esto ha sido posible gracias a la continua resolución de todos los obstáculos que se han presentado durante el ciclo de vida del proyecto.

En resumen, una vez finalizado el proyecto, se puede afirmar que se han cumplido cada uno de los propósitos fijados en el inicio del mismo.

8.1. Desarrollos futuros

Se van a proponer las principales mejoras que se pueden realizar sobre este proyecto:

- Añadido del espacio de trabajo completo dentro del visualizador Rviz con ayuda de la cámara tridimensional.
- Uso de un procedimiento de calibración mediante marcadores o tableros, con el objetivo de poder sincronizar los sistemas de referencia de la cámara y el robot con una ubicación cualquiera, pudiendo controlar así cualquier posición y orientación de los objetos y herramienta.
- Sincronización de varios robots colaborativos para coordinar tareas de este estilo de forma conjunta.

Apéndice A

Anexo

En este apéndice se incluye un manual de usuario creado para poder hacer uso del proyecto creado. Se añaden, adicionalmente, un diagrama de articulaciones y el diagrama de nodos original de la implementación final.

A.1. Manual de usuario

La motivación de este proyecto consiste en implementar diferentes aplicaciones de Pick and Place con el robot colaborativo UR3e y visión artificial. El objetivo es realizarlo con la ayuda de ROS2 en *Python*. La principal idea es reconocer varios objetos rectangulares de diferentes colores con ayuda de la cámara, convertir sus posiciones y que el robot sea capaz de realizar aplicaciones con estas. Se realizan en función de la petición del usuario introducida a través de una interfaz gráfica. Además, se va a visualizar en tiempo real el comportamiento tanto de la cámara como del robot con *Rviz*.

Los recursos necesarios para seguir este proyecto se encuentran disponibles en los siguientes enlaces:

1. Link del repositorio con todos los recursos disponibles:
   `https://github.com/mariooot13/Pick-and-Place-with-ROS2/tree/tutorial`[1]
2. Link de un vídeo propio explicativo de Youtube, en el cual se observa el flujo normal de funcionamiento del proyecto.
   `https://youtu.be/0bSZxQ-wXPE`
3. El diagrama de nodos del proyecto describe su funcionamiento y comunicaciones por completo. Se encuentra disponible en el anexo A.3.

A.1.1. Materiales necesarios

En cuánto a *software*, se necesita tener instalado correctamente *Python*, ROS2, *OpenCV*, *DepthAI* y *Tkinter*. En cuánto a hardware, en este proyecto se ha utilizado:

- Robot colaborativo UR3e.
- Cámara 3D con modelo OAK-D Lite AF.

[1] `https://github.com/mariooot13/Pick-and-Place-with-ROS2/tree/tutorial`

- Herramienta *Gripper* HRC-03.
- Estructura con soporte para la cámara, de forma que quede vertical al espacio de trabajo.
- Cables de conexión: 3 tipo Ethernet y 1 tipo USB-C.

A.1.2. Descripción de paquetes

- *my_func_nodes*. Nodos propios, creados para el correcto y completo funcionamiento del proyecto.
  - ▷▷ *my_func_nodes*
    - ▷ *camera.py*: Nodo encargado de generar las imágenes que visualiza la cámara para su posterior comunicación con el nodo de detección.
    - ▷ *camera_pub_pos.py*: Nodo encargado de analizar dichas imágenes y obetener la posición de los objetos para comunicarse con el control del robot.
    - ▷ *control_robot_master.py*: Nodo que controla el movimiento del robot mediante *Moveit*, así como su herramienta. Además, gestiona todas las comunicaciones internasy estado actual del robot.
    - •
    - ▷ *interfaz_menu.py*: Nodo encargado de crear y lanzar la interfaz gráfica de usuario la cual se comunica con los demás nodos, integrando así el proyecto.
  - ▷▷ *resources*
    - ▷ *euler_to_quat.py*: Calculadora que transforma vector de rotación a cuaternio para ajustar la orientación del gripper.
    - ▷ Imágenes para la interfaz gráfica.
- *my_moveit2_py*. Recursos en relación a planificación de trayectorias con *Moveit*.
  - ▷▷ *my_moveit2_py*
    - ▷ *Moveit2_resources.py*: Librería basada en *Moveit* con funciones para planificar y ejecutar trayectorias del robot.
    - ▷ *ur3e_model.py*: Modelo configurado del robot UR3e para el correcto funcionamiento de la librería.
- *my_robot_bringup_ms* Paquete de lanzamiento.
  - ▷▷ *launch*
    - ▷ *control_robot.py*: Archivo de lanzamiento principal del proyecto.
    - ▷ *launch_descripition_resources*: Información complementaria.
  - ▷▷ *config*: Parámetros característicos para el funcionamiento correcto de los controladores.
    - ▷ *argsforlaunching.yaml*
    - ▷ *param_bringup.yaml*

A.1.3. Primeros pasos

Este manual de usuario ha sido creado para aquellos usuarios que tienen una distribución de Ubuntu. En caso contrario, serán necesarias acciones adicionales.

1. *ROS2 Foxy y dependencias*. En primer lugar, se necesita instalar ROS2 y todas sus dependencias de forma correcta. Se ha usado la distribución Foxy. Se puede encontrar la documentación de ROS2 *Foxy* para su instalación en el siguiente enlace: `https://docs.ros.org/en/foxy/Installation.html`

2. Librerías necesarias. A continuación, se explicará como se llevar a cabo la instalación por terminal de todas las librerías mencionadas en requisitos.

   - *Python3 & Tkinter*. Escriba en la terminal el siguiente comando:
     sudo apt install python3-colcon-common-extensions
   - *OpenCV* y dependencias para ROS2. Se recomienda clonar su repositorio oficial: git clone https://github.com/opencv/opencv.git Posteriormente hay que configurarlo según la documentación de instalación. Se puede consultar su documentación en `https://opencv.org/`
   - *DepthAI*. Se debe clonar el repositorio de DepthAI para ROS2 y construirlo de forma apropiada con el siguiente comando por terminal:
     git clone https://github.com/luxonis/depthai_ros.git

3. *UR ROS2 Drivers*. El siguiente paso es clonar los controladores de UR y configurar sus paquetes. Se puede realizar siguiendo las instrucciones del apartado *Getting started* del repositorio UR en el siguiente enlace: `https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver/tree/foxy`
   Es importante crear un directorio de trabajo *workspace*, como viene indicado.

4. Repositorio actual. Una vez se tienen clonados los controladores de UR, es el momento de clonar este repositorio en la rama Tutorial.

   a) Dirigite hacia /workspace/ros_ur_driver/src

   b) Opción 1. Se puede descargar el archivo comprimido con formato .zip y copialo dentro del directorio src.
   Opción 2. Se debe introducir el comando del código A.1 por terminal dentro del espacio de trabajo:

código A.1: Comando para clonar el repositorio actual. Manual de usuario

```
1  git clone https://github.com/mariooot13/Pick-
     and-Place-with-ROS2/tree/tutorial
```

A.1.4. Consideraciones previas

Existen una serie de recomendaciones que se debe saber para el trabajo correcto del proyecto:

- Para llevar a cabo de forma correcta toda la visión artificial, es necesario alinear previamente el sistema de referencia en el nodo de detección.

- Cualquier duda adicional que pueda surgir, está resuelta en la memoria de este trabajo de fin de grado adjunta.

Además, se debe conocer como llevar a cabo el primer paso antes de lanzar el proyecto, que es la correcta conexión de los dispositivos.

a) Conexión del robot. Dentro del archivo de lanzamiento se ha asignado una dirección IP, la cual, deberá ser la seleccionada en la tablet del robot. Se debe conectar el cable *Ethernet* al ordenador, y visualizar que dirección IP tiene el ordenador. En la tablet, se crea un programa con control externo en el que se asigna la dirección IP del ordenador.

b) Instalación de la herramienta. La herramienta debe estar instalada en la tablet del robot. En este caso, el archivo de instalación se denomina *urcap_installation*.

c) Cámara. Se conecta la cámara mediante un cable con conexión USB-C y se debe asegurar su correcto posicionamiento.

A.1.5. Archivo de lanzamiento del proyecto

Dentro del archivo de lanzamiento, se ejecutan los siguientes componentes, los cuales, se muestran mediante el nombre de su ejecutable:

- ur_control.launch.py. Archivo que lanza los controladores de UR, encargado de la conexión con el robot y, a su vez, lanza nodos importantes para su funcionamiento.
- ur_moveit.launch.py. Archivo que lanza el nodo *move_group*, para la planificación de trayectorias. Además, lanza el visualizador *Rviz* y otros nodos complementarios de trasnformaciones robóticas.
- control_robot_node_exec. Ejecutable del nodo de control principal del robot, control_robot_master.
- camera_exec. Ejecutable del nodo que genera las imágenes de la cámara, lanza su visualización y publica en tiempo real con formato *ros*.
- camera_detection. Ejecutable del nodo encargado del análisis de las imágenes publicadas por la cámara y de la detección, reconocimiento y obtención de posición de los objetos. Además, incluye todo tipo de comunicaciones en el diagrama principal.
- interfaz_exec. Ejecutable del nodo responsable de la creación de la interfaz y de sus comunicaciones con el resto de nodos según la interacción externa.

A.1.5.1. Lanzamiento individual

A pesar de haber creado un archivo de lanzamiento que facilite el funcionamiento del proyecto en un único comando, existe la posibilidad de ejecutar cada nodo de forma individual gracias a sus ejecutables:

código A.2: Comando de ejecución genérico

```
1  ros2 run my_func_nodes "nombre del ejecutable"
```

A.1.6. Uso

1. Abre una terminal en tu ordenador.
2. Necesitarás escribir el siguiente comando por pantalla.

código A.3: Comando de configuración del entorno de trabajo

```
source install/setup.bash
```

3. Lanza el archivo de lanzamiento en dicha terminal.

código A.4: Comando de lanzamiento del proyecto

```
ros2 launch my_robot_bringup_ms control_robot.py
```

4. Dale al *PLAY* en el programa del robot en la tablet.
5. Interacciona con la interfaz gráfica lanzada.
   - Selecciona la secuencia de colores del orden de los objetos a manipular.
   - Selecciona el caso de uso a realizar.

A.2. *Frames* de las articulaciones del UR3e

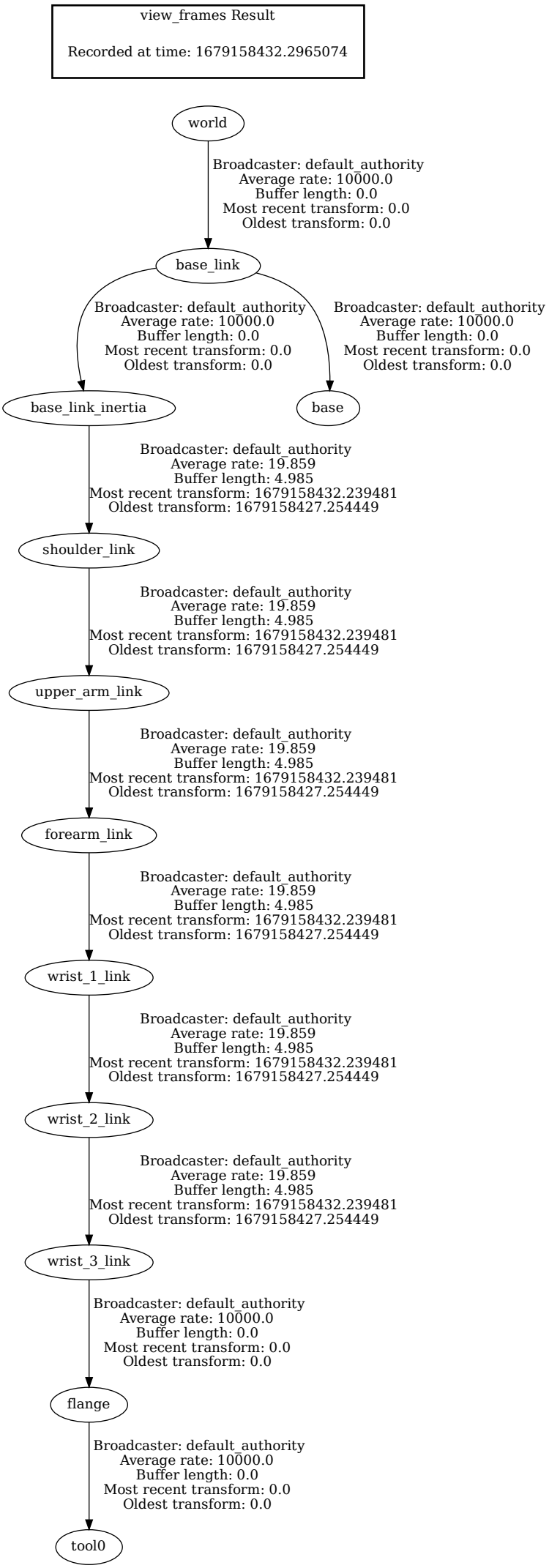
En este anexo, disponible en la página 88, se agrega un esquema visual que muestra los *frames* de cada articulación del robot obtenidos a través del comando del código A.5. Este proporciona información sobre la posición y orientación de cada eslabon en relación al sistema de referencia origen. Dichos recursos resultan necesarios para trabajar en la planificación del robot.

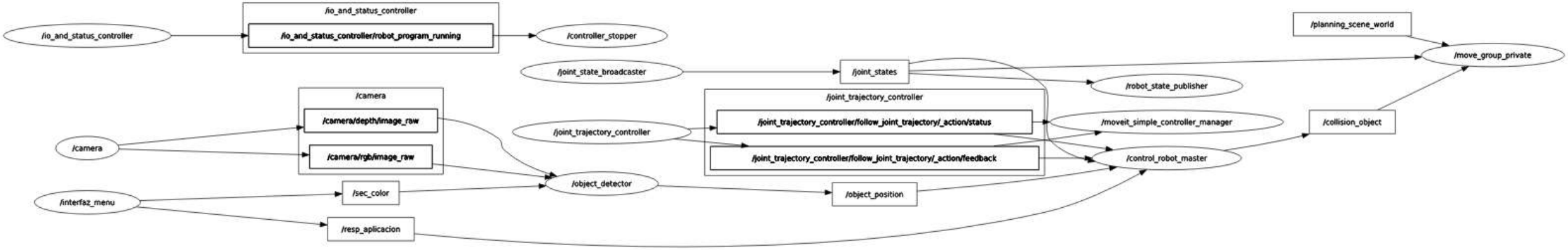
código A.5: Comando de visualización de frames

```
ros2 run tf2_tools view_frames.py
```

A.3. Diagrama de nodos original

Se adjunta el diagrama de nodos original obtenido a través de la herramienta *rqt_graph*, durante el funcionamiento de un caso de uso del proyecto. Se encuentra en la página 89.


view_frames Result
Recorded at time: 1679158432.2965074
world
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link_inertia
base
Broadcaster: default_authority
Average rate: 19.859
Buffer length: 4.985
Most recent transform: 1679158432.239481
Oldest transform: 1679158427.254449
shoulder_link
Broadcaster: default_authority
Average rate: 19.859
Buffer length: 4.985
Most recent transform: 1679158432.239481
Oldest transform: 1679158427.254449
upper_arm_link
Broadcaster: default_authority
Average rate: 19.859
Buffer length: 4.985
Most recent transform: 1679158432.239481
Oldest transform: 1679158427.254449
forearm_link
Broadcaster: default_authority
Average rate: 19.859
Buffer length: 4.985
Most recent transform: 1679158432.239481
Oldest transform: 1679158427.254449
wrist_1_link
Broadcaster: default_authority
Average rate: 19.859
Buffer length: 4.985
Most recent transform: 1679158432.239481
Oldest transform: 1679158427.254449
wrist_2_link
Broadcaster: default_authority
Average rate: 19.859
Buffer length: 4.985
Most recent transform: 1679158432.239481
Oldest transform: 1679158427.254449
wrist_3_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
flange
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
tool0


/io_and_status_controller
/io_and_status_controller
/io_and_status_controller/robot_program_running
/controller_stopper
/planning_scene_world
/move_group_private
/joint_state_broadcaster
/joint_states
/robot_state_publisher
/camera
/camera/depth/image_raw
/camera/rgb/image_raw
/camera
/joint_trajectory_controller
/joint_trajectory_controller/follow_joint_trajectory/_action/status
/joint_trajectory_controller/follow_joint_trajectory/_action/feedback
/joint_trajectory_controller
/moveit_simple_controller_manager
/collision_object
/control_robot_master
/interfaz_menu
/sec_color
/object_detector
/object_position
/resp_aplicacion

Bibliografía

[1] Robótica en la Quinta Revolución Industrial. In *2022 IEEE 14.ª Conferencia internacional sobre humanoides, nanotecnología, tecnología de la información, comunicación y control, medio ambiente y gestión (HNICEM)*.

[2] RRT-connect: un enfoque eficiente para la planificación de rutas de consulta única. In *Actas 2000 ICRA. Conferencia del Milenio. Conferencia Internacional IEEE sobre Robótica y Automatización. Actas de simposios (Cat. No.00CH37065)*.

[3] La robótica colaborativa se afianza en la transformación de la industria automotriz. *Revista de Robots*, 2022. Recuperado de `https://revistaderobots.com/robots-y-robotica/la-robotica-colaborativa-se-afianza-en-la-transformacion-de-la-industria-automotriz/` Último acceso en junio de 2023. (Copiar y pegar en un navegador).

[4] Interfaces gráficas de usuario con tk. tkinter documentation. 3.11.3, 2023. Recuperado de `https://docs.python.org/` Último acceso en mayo de 2023.

[5] Opencv tutorials. 2023. Recuperado de `https://opencv.org/` Último acceso en mayo de 2023.

[6] A Adriaensen, F Costantino, G Di Gravio, and R Patriarca. Teaming with industrial cobots: A socio-technical perspective on safety analysis. *Human Factors and Ergonomics in Manufacturing & Service Industries*, 32(2):173–198, 2022.

[7] AndrejOrsula. Repositorio pymoveit2. 2021. Recuperado de `https://github.com/AndrejOrsula/pymoveit2` Último acceso en abril de 2023.

[8] Universal Robots A/S. Universal robots e-series manual de usuario. *UR*, 5.0.0:1–200, 2018.

[9] Irati Zamalloa Ugarte Víctor Mayoral Vilches Carlos San Vicente Gutiérrez, Lander Usategui San Juan. Towards a distributed and real-time framework for robots: Evaluation of ros 2.0 communications for real-time robotic applications. 2018.

[10] Clearview. Introducción a la visión artificial en 3d y diferencias entre aplicaciones 2d y 3d. Recuperado de `https://www.clearview-imaging.com/es/blog/introduccion-a-la-vision-artificial-en-3d-y-diferencias-entre-aplicaciones-2d-y-3d` Último acceso en junio de 2023. (Copiar y pegar en un navegador).

[11] Cade Cobots. Pick and place. qué es y aplicación de la robótica colaborativa. 2021. Recuperado de `https://cadecobots.com/pick-and-place-que-es-y-aplicacion-de-la-robotica-colaborativa/` Último acceso en mayo de 2023.

[12] Pablo Manuel Valle Concepcion. Introducción al análisis cuaterniónico y a sus aplicaciones. *Universidad de la Laguna*, 2022.

[13] Sachin Chitta Nikolaus Correll David Coleman, Ioan A. Şucan. Reducing the barrier to entry of complex robotic software: a moveit! case study. *Journal of Software Engineering for Robotics*, 5:3–16, 2014.

[14] Revista de Robots. ¿qué es un robot delta? 2023. Recuperado de `https://revistaderobots.com/robots-y-robotica/robot-delta-aplicaciones-y-precios/`. Último acceso en junio de 2023.

[15] Revista de Robots. ¿qué es un robot scara? 2023. Recuperado de `https://revistaderobots.com/robots-y-robotica/robot-scara-articulados-caracteristicas-y-marcas/`. Último acceso en junio de 2023.

[16] Shirine El Zaatari, Mohamed Marei, Weidong Li, and Zahid Usman. Cobot programming for collaborative industrial tasks: An overview. *Robotics and Autonomous Systems*, 116:162–180, 2019.

[17] Geek Gasteiz. Ros2 vs ros (1) – ¿migramos? 2018. Recuperado de https://geekgasteiz.wordpress.com//2018/11/01/ros2-vs-ros-1-migramos/ Último acceso en junio de 2023.

[18] Zimmer Gorup. Pinza de prensión eléctrica hrc-03 series. 2023. Recuperado de https://www.directindustry.es/prod/zimmer-group/product-14534-2222425.html. Último acceso en abril de 2023.

[19] J.S. Gyorfi and Chi-Haur Wu. Coordinated planning and control of multiple robots and machines for surface-mount manufacturing. In *Proceedings. IEEE International Joint Symposia on Intelligence and Systems (Cat. No.98EX174)*, pages 268–272, 1998.

[20] Mohammad Nurul Hassan Reza, Chinnasamy Agamudai Nambi Malarvizhi, Sreenivasan Jayashree, and Muhammad Mohiuddin. Industry 4.0–technological revolution and sustainable firm performance. In *2021 Emerging Trends in Industry 4.0 (ETI 4.0)*, pages 1–6, 2021.

[21] Clearpath Robotics Inc. User manual turtlebot. 2023. Recuperado de https://turtlebot.github.io/turtlebot4-user-manual/software/rviz.html Último acceso en abril de 2023.

[22] Mordor Intelligence. Mercado de robots colaborativos: Crecimiento, tendencias, impacto de covid-19 y pronÓsticos (2023 - 2028). 2023. Recuperado de https://www.mordorintelligence.com/es/industry-reports/collaborative-robot-market. Último acceso en mayo de 2023.

[23] Steven M. La Valle. Motion planning. *IEEE Robotics Automation Magazine*, 18(2):108–118, 2011.

[24] F. Hugo Ramírez Leyva. *Robótica 2. Modelado Cinemática de Robots*, volume Cubículo 3. Instituto de Electrónica y Mecatrónica, 2012.

[25] Xin Li and Yiliang Shi. Computer vision imaging based on artificial intelligence. In *2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pages 22–25, 2018.

[26] Xin Li and Yiliang Shi. Computer vision imaging based on artificial intelligence. In *2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pages 22–25, 2018.

[27] Luxonis. Depthai's documentation, 2023. Recuperado de https://docs.luxonis.com/en/latest/ Último acceso en abril de 2023.

[28] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[29] Matlab. Simulink para diseño y simulación de sistemas. 2023. Recuperado de https://es.mathworks.com/solutions/system-design-simulation.html Último acceso en junio de 2023.

[30] Mohammad Ehsan Matour and Alexander Winkler. A portable vision-based and force controlled collaborative robot system for entertainment purposes. In *ISR Europe 2022; 54th International Symposium on Robotics*, pages 1–6, 2022.

[31] Justinas Mišeikis, Kyrre Glette, Ole Jakob Elle, and Jim Torresen. Multi 3d camera mapping for predictive and reflexive robot manipulator trajectory estimation. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

[32] Hussein Mnyusiwalla, Pavlos Triantafyllou, Panagiotis Sotiropoulos, Máximo A. Roa, Werner Friedl, Ashok M. Sundaram, Duncan Russell, and Graham Deacon. A bin-picking benchmark for systematic evaluation of robotic pick-and-place systems. *IEEE Robotics and Automation Letters*, 5(2):1389–1396, 2020.

[33] Nebext. Inteligencia artificial y robótica para una industria más sostenible. *Advanced Factories. EXPO & CONGRESS*, 2023. Recuperado de https://www.advancedfactories.com/inteligencia-artificial-robotica-industria-sostenible/. Último acceso en junio de 2023.

[34] Oasys. ¿qué es la industria 5.0 y cuál es su objetivo? 5.0, 2023. Recuperado de https://oasys-sw.com/ Último acceso en mayo de 2023.

[35] I. Ohya, A. Kosaka, and A. Kak. Vision-based navigation by a mobile robot with obstacle avoidance using single-camera vision and ultrasonic sensing. *IEEE Transactions on Robotics and Automation*, 14(6):969–978, 1998.

[36] Jordi Pelegrí. La cuarta revolución indusrial: cobots y automatización. *Universal Robots Blog*, 5.0, 2019. Recuperado de `https://www.universal-robots.com/es/blog/la-cuarta-revolucion-industrial-cobots-y-automatizacion/` Último acceso en mayo de 2023.

[37] Jinchang Ren, Theodore Vlachos, and Vasileios Argyriou. Immersive and perceptual human-computer interaction using computer vision techniques. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 66–72, 2010.

[38] ROS teacher at Master of Industry 4.0 Ricardo Tellez. Ros2 developers' guide. *The Construct*, page 14, 2021.

[39] F.M. Rico. *A Concise Introduction to Robot Programming with ROS2*. CRC Press, 2022.

[40] Filippo Sanfilippo and Kristin Ytterstad Pettersen. Openmrh: A modular robotic hand generator plugin for openrave. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1–6, 2015.

[41] Inesem Business School. Grados de libertad en robótica. 2023. Recuperado de `https://www.inesem.es/revistadigital/gestion-integrada/diferencia-robotica-grados-libertad-movilidad-3`. Último acceso en junio de 2023.

[42] Dirk Thomas. Cambios entre ros 1 y ros 2. 2017. Recuperado de `http://design.ros2.org/articles/changes.html` Último acceso en junio de 2023.

[43] Wikipedia. Robótica. 2023. Recuperado de `https://es.wikipedia.org/wiki/Robotica` Último acceso en junio de 2023.

[44] Wikpedia. Brazo robótico. 2023. Recuperado de `https://es.wikipedia.org/wiki/Brazo_robotico` Último acceso en junio de 2023.

[45] Yonggan Yan, Shuxiang Guo, Chuqiao Lyu, Duohao Zhao, and Zhijun Lin. Sea-based humanoid finger-functional parallel gripper with two actuators: Pg2 gripper. *IEEE Transactions on Instrumentation and Measurement*, 72:1–13, 2023.

[46] Hongshan Yu, Ke Zhao, Yaonan Wang, Luo Kan, Mingui Sun, and Wenyan Jia. Registration and fusion for tof camera and 2d camera reading. In *2013 Chinese Automation Congress*, pages 679–684, 2013.

[47] Tao Yu, Shishi Zeng, Yanpei Luo, and Yao Ding. Application research of online label defect detection based on machine vision. In *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, volume 6, pages 1398–1402, 2022.

[48] Shuai Zhao, Yao Tong, Gang Chen, and Yiran Zhang. Testing the feasibility of quantizing the progress of stroke patients' rehabilitation with a computer vision method. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning International Conference on Computer Engineering and Applications (CVIDL ICCEA)*, pages 1–4, 2022.

[49] Qiu zhe. Ros 2 documentation: Foxy. 2.0, 2021. Recuperado de `https://docs.ros.org/` Último acceso en mayo de 2023.