UNIVERSIDAD
DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Ingeniería de Sistemas y Automática

Área de Conocimiento: Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

SEGMENTACIÓN SEMÁNTICA PARA LA MANIPULACIÓN FLEXIBLE CON ROBOTS INDUSTRIALES

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Autor: MARTÍN MARTOS, PEDRO

Tutor: VÁZQUEZ MARTÍN, RICARDO

MÁLAGA, Julio de 2.025

Resumen

En este proyecto se desarrolla un sistema de manipulación robótica flexible basado en visión artificial, orientado a aplicaciones industriales. El objetivo principal es identificar objetos, localizarlos mediante segmentación de instancias y estimar automáticamente la pose (posición y orientación) para permitir su manipulación mediante un brazo robótico.

El sistema se estructura en una arquitectura modular sobre el ecosistema ROS 2, que integra componentes de percepción, control del manipulador y una interfaz gráfica en RViz2. Se ha utilizado la cámara estereoscópica ZED Mini para capturar imágenes RGB-D, y un modelo YOLO de segmentación de instancias para identificar objetos y generar sus correspondientes máscaras que, junto con la información de profundidad, permiten calcular el punto de agarre óptimo.

Los resultados obtenidos muestran un comportamiento robusto y en tiempo real, lo que valida el uso de técnicas de visión avanzada en entornos industriales no estructurados.

Palabras claves

Robótica industrial, visión artificial, segmentación de instancias, ROS 2, manipulación robótica, YOLO, ZED Mini.

Abstract

This project presents the development of a flexible robotic manipulation system based on computer vision, aimed at industrial applications. The main goal is to identify, locate objects through instance segmentation and automatically compute their pose (position and orientation) to enable their manipulation by a robotic arm.

The system follows a modular architecture built on the ROS 2 ecosystem, integrating visual perception components, manipulator control, and a graphical interface in RViz2. The ZED Mini stereo camera is used to capture RGB-D images, and a YOLO-based instance segmentation model identifies objects and generates corresponding masks. These, together with depth information, allow the optimal grasping point to be computed.

The results demonstrate robust and real-time performance, validating the application of advanced vision techniques in unstructured industrial environments.

Keywords

Industrial robotics, computer vision, instance segmentation, ROS 2, robotic manipulation, YOLO, ZED Mini.

Índice

Índice de figuras

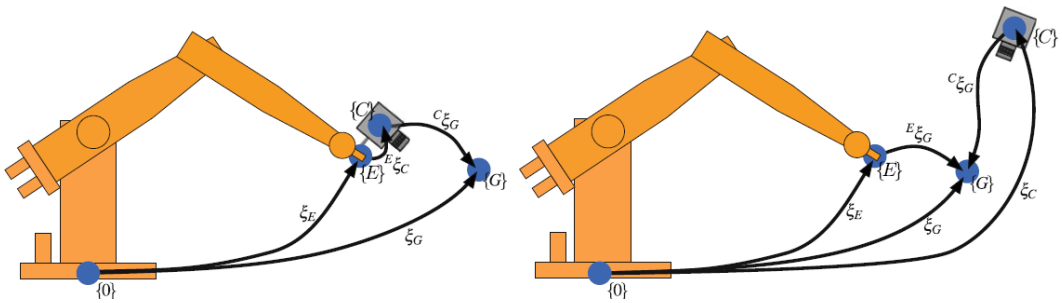
1 Introducción

En este capítulo se contextualiza el trabajo realizado, incluyendo los antecedentes más relevantes y los objetivos perseguidos. Se presenta también la estructura general del documento.

1.1 Antecedentes

Una de las limitaciones más significativas de los sistemas robóticos tradicionales es el déficit sensorial que impide la adaptación dinámica al entorno. Si el robot pudiera observar tanto su efector final como el objetivo, sería capaz de realizar tareas de forma autónoma guiado por esa percepción. El uso de información visual en tiempo real para guiar un robot hacia un objetivo es la base del control visual (*visual servoing*).

Este tipo de control se implementa típicamente en dos configuraciones: con una cámara montada en el efector final del robot (*eye-in-hand*) o fija en el entorno (*eye-to-hand*), como se muestra en la Figura 1.1.



a) Cámara montada en efector final (*eye-in-hand*) **b) Cámara fijada en espacio de trabajo (*eye-to-hand*)**

Figura 1.1. Sistemas de referencias principales en las distintas configuraciones de control por visión. Fuente: [1]

Este trabajo continúa y amplía una línea de trabajo iniciada en un TFG anterior [2], en el que se implementó una arquitectura básica de control visual sobre ROS 2, utilizando una cámara estéreo ZED y un manipulador robótico.

En aquel proyecto, la detección de objetos se basaba únicamente en la posición del centroide de las cajas delimitadoras (*bounding boxes)* proporcionadas por la red neuronal integrada en el kit de desarrollo (ZED SDK), sin procesamiento adicional de la imagen ni estimación de orientación.

En contraste, el presente trabajo propone una mejora significativa al incorporar una red neuronal más avanzada (YOLO junto con segmentación) para procesar directamente la imagen capturada por la cámara. Esto permite obtener máscaras de segmentación de los objetos facilitando un cálculo más robusto del punto y la orientación de agarre. Todo ello se integra de forma modular en ROS 2.

1.2 Motivación

Este trabajo busca dotar a un robot manipulador de capacidades de percepción robusta y flexible, orientadas a tareas de manipulación como el *bin-picking* en entornos no estructurados. Para ello, se emplea visión artificial avanzada para detectar, clasificar y localizar objetos en el espacio tridimensional del área de trabajo. La información obtenida se integra en tareas de planificación y control, permitiendo al sistema adaptarse a la variabilidad del entorno y mejorar su capacidad operativa. Este enfoque está alineado con los principios de la Industria 4.0, que promueve sistemas de fabricación más inteligentes, autónomos y flexibles.

1.3 Objetivos

El presente proyecto tiene como objetivo principal el desarrollo de un sistema capaz de detectar y segmentar objetos en un entorno desestructurado utilizando una cámara RGB-D estéreo. A partir de la información visual, el sistema debe ser capaz de calcular el punto de agarre óptimo para cada objeto y transmitirlo al robot manipulador, con el fin de ejecutar tareas de manipulación precisas y robustas, incluso en condiciones no estructuradas.

Como objetivo secundario, se plantea la implementación de un sistema de monitorización que incluya la visualización del proceso de segmentación y el estado

del manipulador. Además, se integrará un mecanismo de tolerancia a errores, que permita detectar fallos en la manipulación y realizar reintentos automáticos en caso de que el agarre no se haya realizado correctamente.

1.4 Revisión de métodos y sistemas para control por visión y manipulación robótica

Según el artículo titulado *A Tutorial On Visual Servo Control* de Hutchinson et al. [3], los orígenes del control visual se remontan a los trabajos pioneros de Shirai e Inoue (1973) [4], quienes propusieron un sistema de retroalimentación visual para corregir la posición del robot. En la Figura 1.2 se muestra un esquema típico de un lazo de control visual en bucle cerrado.

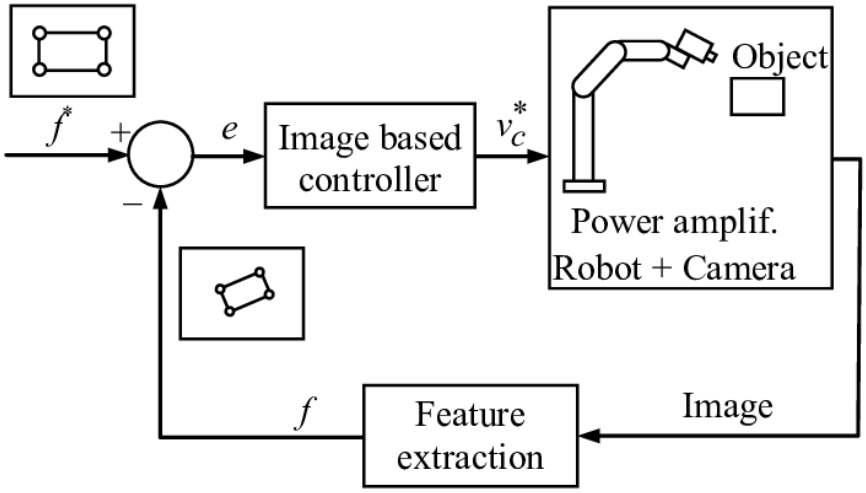


Figura 1.2. Bucle de retroalimentación típico en control visual (*image-based visual servoing*), alineado con los primeros esquemas propuestos por Shirai e Inoue [4] Fuente: [5]

Sin embargo, la baja velocidad del procesamiento visual en esa época limitó significativamente su aplicabilidad. Posteriormente, surgieron numerosos trabajos que sentaron las bases conceptuales del campo, entre ellos los desarrollos realizados en SRI International (Hill y Park [6], 1979; Makhlin [7], 1985), así como la propuesta de clasificación entre control visual basado en la posición (PBVS) y basado en la imagen (IBVS) presentada por Weiss et al. (1987) [8]. Esta clasificación, aún vigente, permitió sistematizar los enfoques de control visual y

facilitó el desarrollo de aplicaciones prácticas, como aquellas que incorporan el control de la dinámica del manipulador (Kelly [9]).

En PBVS (Figura 1.3), se utilizan características visuales, una cámara calibrada y un modelo geométrico del objeto para estimar su pose. La ley de control actúa en el espacio cartesiano. Este enfoque puede ser preciso, pero depende estrechamente de una buena calibración y suele requerir mayor capacidad computacional.

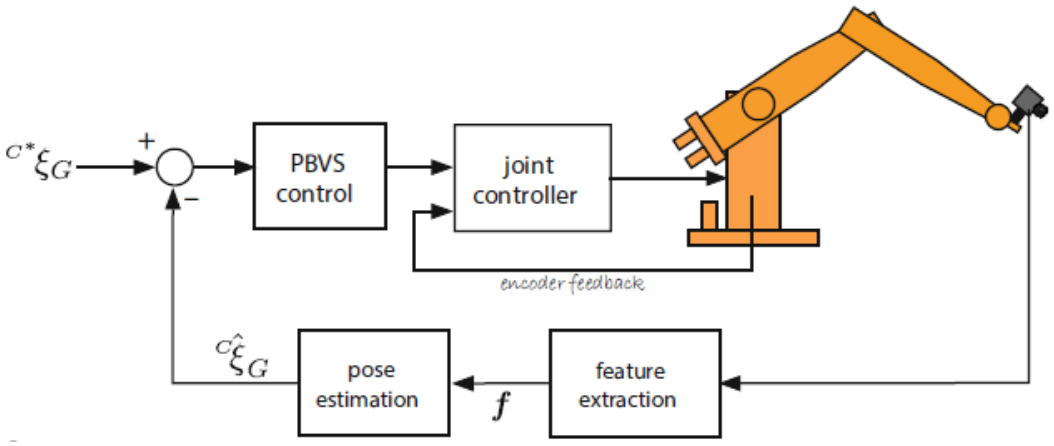


Figura 1.3. Control por visión basado en posición. Fuente: [1]

En cambio, IBVS (Figura 1.4) genera la ley de control directamente en el espacio de imagen. Se minimiza el error entre las características visuales actuales y las deseadas, sin necesidad de reconstruir la pose. Este enfoque es más robusto ante errores de calibración, aunque puede presentar trayectorias menos predecibles.

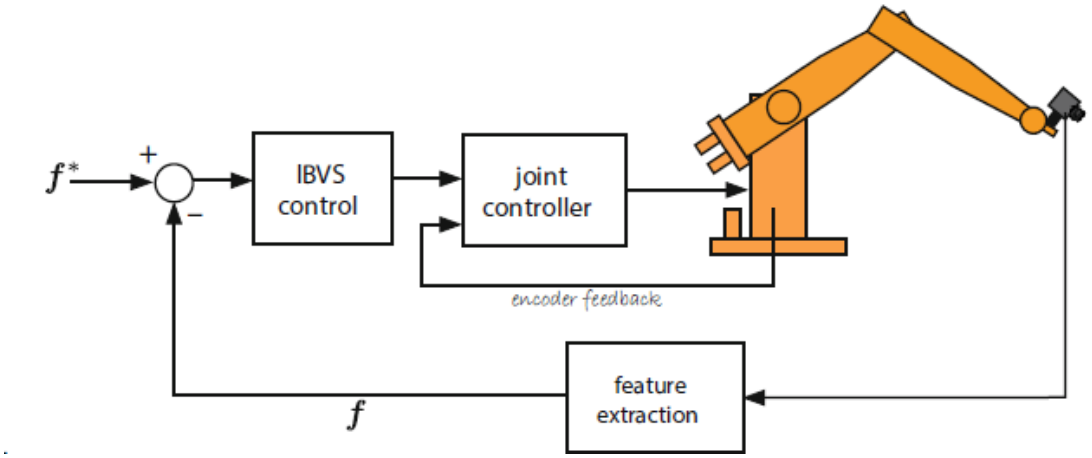


Figura 1.4. Control por visión basado en imagen. Fuente: [1]

Con el avance de la capacidad computacional y la mejora en los sensores, fue posible implementar sistemas de control visual más complejos y precisos. Esto permitió incorporar técnicas como la segmentación semántica tridimensional y el uso de redes neuronales profundas para procesar la información visual en tiempo real.

Tal como se describe en [10], en los últimos años el aprendizaje profundo ha tomado un papel protagonista en el control visual, al sustituir los métodos tradicionales de extracción de características por redes neuronales convolucionales (CNN), capaces de identificar objetos, calcular poses y generar máscaras de segmentación con alta robustez.

Este enfoque ha sido aplicado con éxito en entornos desestructurados, como en la agricultura, donde se busca automatizar tareas como la recolección de frutos (Figura 1.5). Otros trabajos exploran aplicaciones industriales, como la soldadura o el ensamblaje (Figura 1.6).

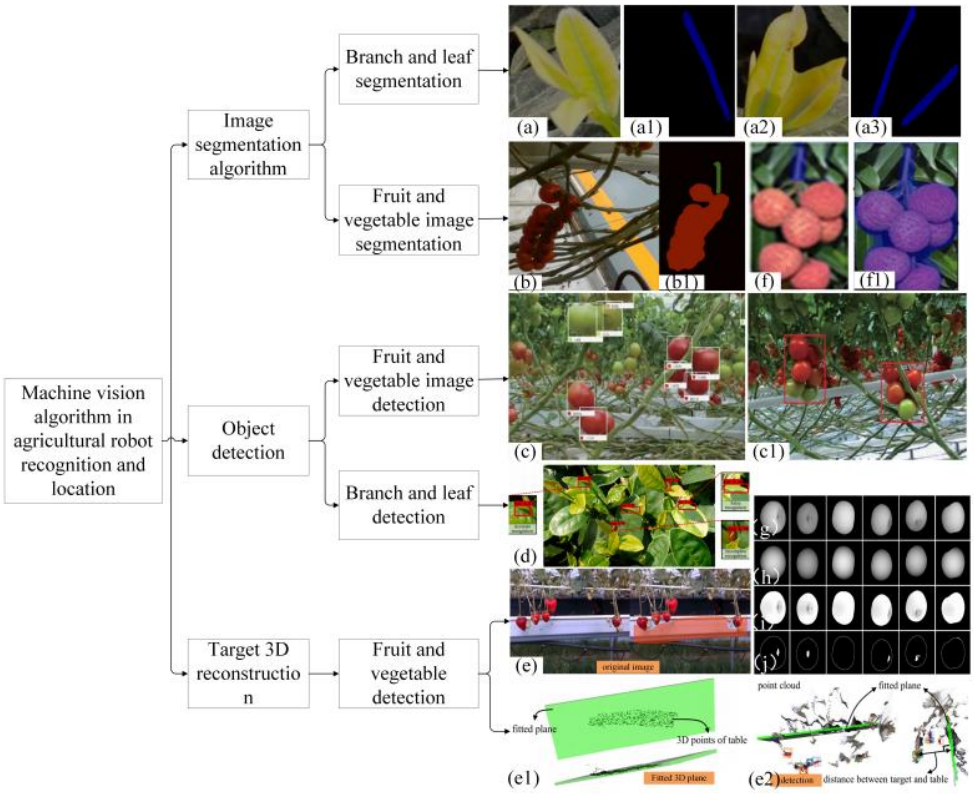


Figura 1.5. Técnicas de procesamiento de imágenes para robots agrícolas: segmentación, detección de objetos y reconstrucción 3D. Fuente: [11]

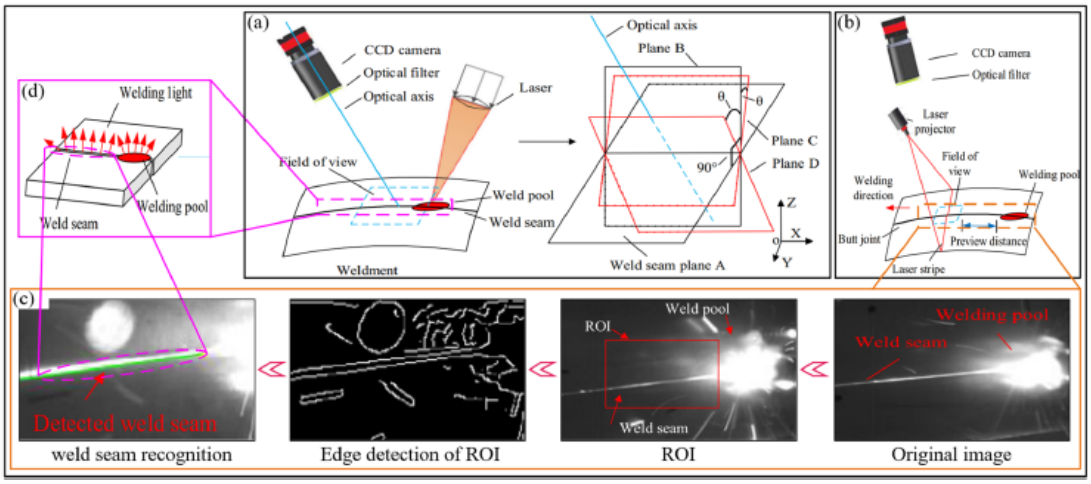


Figura 1.6. Ejemplo de control por visión en soldadura automática industrial. Fuente: [12]

Atendiendo a estas líneas de desarrollo, el trabajo expuesto en esta memoria explora un sistema de control por visión basado en redes neuronales y percepción tridimensional estereoscópica, integrado mediante ROS 2 (*Robot Operating System*) para tareas automatizadas de manipulación.

1.5 Estructura de la memoria

El presente trabajo se organiza en cinco capítulos:

El primer capítulo presenta los antecedentes del problema, la motivación del proyecto, los objetivos y una revisión de métodos y soluciones existentes en el ámbito del control por visión y la manipulación robótica.

El capítulo segundo describe las herramientas utilizadas para el desarrollo del proyecto, incluyendo el hardware (cámara ZED mini, robot xArm6, NVIDIA Jetson AGX Xavier) y el entorno de desarrollo basado en Docker.

En el tercero, se detalla el desarrollo del sistema, abarcando la implementación de los diferentes módulos: visión, control del manipulador e interfaz gráfica en RViz2 y la intercomunicación entre ellos dentro del ecosistema ROS 2.

El cuarto capítulo recoge los experimentos realizados, explicando el procedimiento seguido y presentando los resultados obtenidos, junto con su análisis.

Finalmente, el quinto capítulo presenta las conclusiones del trabajo, así como posibles mejoras y líneas de trabajo futuro.

Además, en el apéndice A se incluyen los enlaces al código fuente del proyecto (repositorio en GitHub) y a un video demostrativo del sistema en funcionamiento.

2 Métodos y herramientas

Este capítulo presenta las tecnologías y recursos empleados en el proyecto, detallando los componentes hardware y software esenciales para la implementación del sistema de visión y control robótico.

2.1 Cámara ZED mini de Stereolabs

La cámara estéreo ZED Mini (Figura 2.1), desarrollada por Stereolabs, es un sensor de visión tridimensional diseñado para aplicaciones de robótica, realidad aumentada y navegación autónoma. Funciona mediante un sistema de visión estéreo pasiva basado en dos cámaras físicas sincronizadas, situadas a una distancia fija entre sí. Cada una cuenta con su propio sensor y lente, lo que permite capturar simultáneamente dos imágenes desde perspectivas ligeramente distintas, simulando la visión binocular humana. A partir de esta disparidad, se genera un mapa de profundidad en tiempo real. Permite obtener mapas de profundidad precisos a distancias de hasta 15 metros, además de ofrecer imágenes RGB en alta definición y calcular información adicional, como nubes de puntos y estimación de movimiento (odometría visual).

Figura 2.1. Cámara ZED Mini. Fuente: [13]

Las propiedades de esta cámara, como su resolución estéreo y la precisión en la estimación de profundidad (Figura 2.2), la hacen adecuada para el entorno de aplicación de este proyecto.

Technical Specifications

Video Output	
Output Resolution	Side by Side 2x (2208x1242) @15fps 2x (1920x1080) @30fps 2x (1280x720) @60fps 2x(672x376) @100fps
Output Format	YUV 4:2:2
Field of View	Max. 102°(H) x 57°(V) x 118°(D)
RGB Sensor Type	1/3" 4MP CMOS
Active Array Size	2688x1520 pixels per sensor (4MP)
Focal Length	3.06mm (0.12") - f/2.0
Shutter	Electronic synchronized rolling shutter
Interface	USB 3.0 Type-C port

Physical	
Dimensions	124.5 x 30.5 x 26.5 mm (4.9 x 1.2 x 1.0")
Weight	62.9g - 0.14 lb
Operating Temperature	0°C to +45°C (32°F to 113°F)
Power	380mA/5V USB Powered

Camera Control

The ZED API provides low level access and control of the device and related sensors. The API allows for precise manipulation of common parameters such as frame rate, exposition time, white balance, gain, low light sensitivity. The API will also provide different resolutions.

Motion	
Motion Sensors	Gyroscope, Accelerometer Sampling Rate 800Hz
Technology	Visual-inertial stereo SLAM
6-axis Pose Accuracy	Position: +/- 1mm Orientation: 0.1 deg.
Pose Update Rate	Up to 100 Hz

Depth Sensing	
Baseline	63mm (2.4")
Depth Range Max	0.1m to 15m (0.3ft to 49ft)
Ideal Range	0.1m to 9m (0.3ft to 13.1ft)
Depth Accuracy	< 1.0% at 2m (6.6ft) < 1.8% at 4m (13.1ft)
Depth Map Resolution	Native video resolution (in Ultra mode)

System Requirements

Supported OS	Win 10, Win 11 Ubuntu 20 & 22 CentOS, Debian (via Docker) USB3.0 Interface
SDK Requirements	Dual-core 2.3GHz or faster Minimum 4GB RAM Memory NVIDIA GPU(1) Compute capability ≥ 3.0

(1) Compatible with Nvidia Jetson Nano, TX2, Xavier

Figura 2.2. Tabla de especificaciones de la cámara ZED Mini. Fuente: [14]

Stereolabs proporciona el ZED SDK para facilitar el uso de sus cámaras, incluyendo herramientas y librerías que permiten visualizar, calibrar y configurar el dispositivo, así como desarrollar aplicaciones personalizadas. Entre las existentes, en este trabajo, las herramientas gráficas como ZED_Depth_Viewer (Figura 2.3), utilizada para visualizar la información de profundidad y reconstrucciones por nube de puntos, y ZED_Explorer (Figura 2.4), para comprobaciones de funcionamiento y configuración general, se emplearon únicamente durante la fase de pruebas.

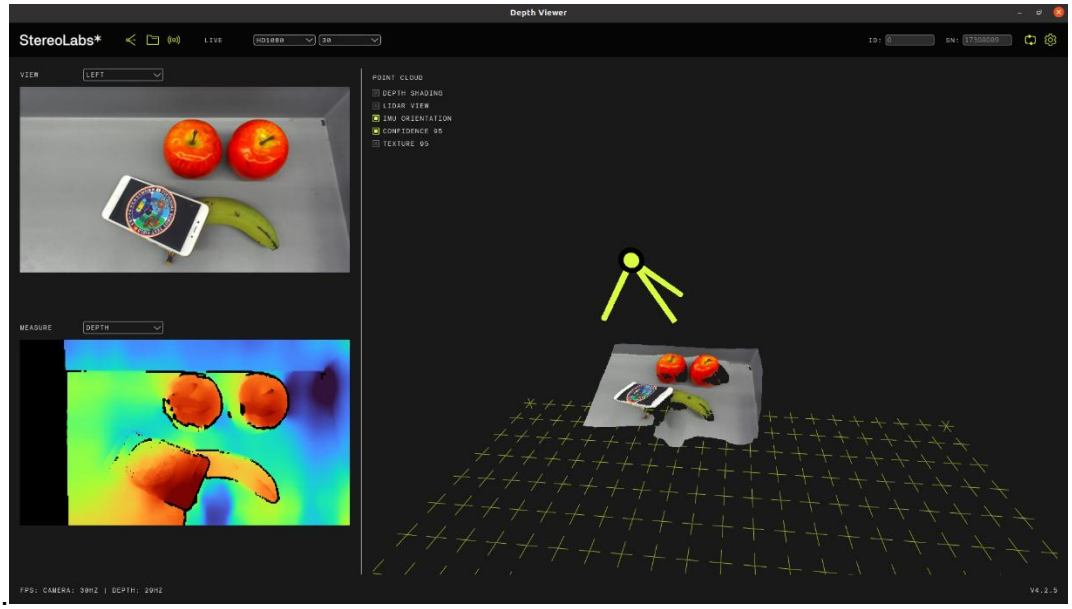


Figura 2.3. Entorno ZED_Depth_Viewer

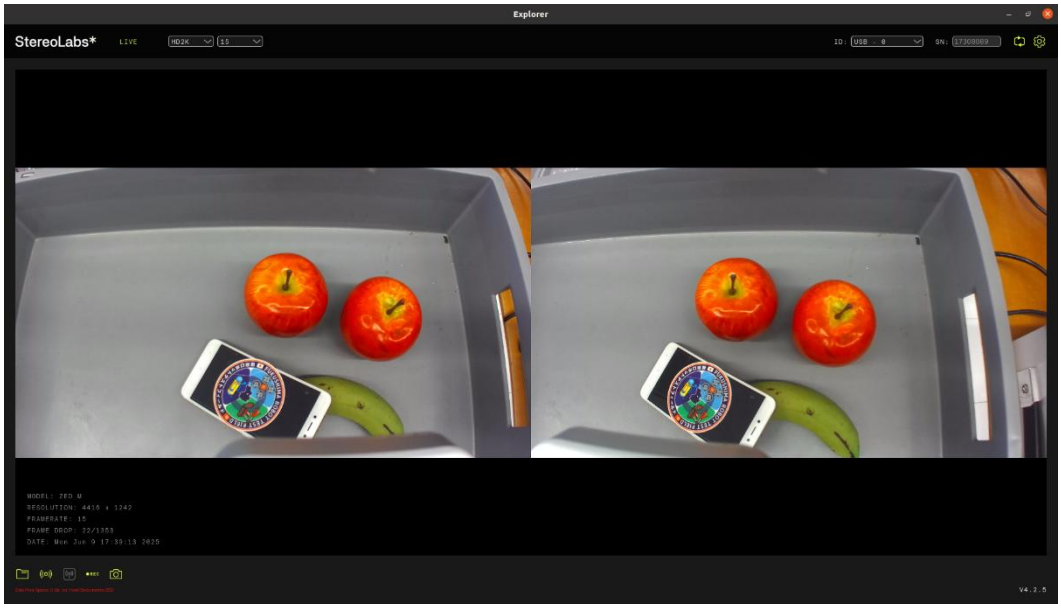


Figura 2.4. Entorno ZED_Explorer

Además de estas herramientas gráficas, el ZED SDK también ofrece recursos orientados al desarrollo que permiten una mayor integración y control del dispositivo desde código. Para el sistema de percepción se recurrió exclusivamente a la API (*Application Programming Interface*) de Python denominada Pyzed [15] una interfaz que facilita la comunicación directa con el software de la cámara, permitiendo el acceso a la información sensorial y a distintos parámetros de configuración.

En el presente trabajo, se utilizaron las capacidades de obtención de imagen RGB y mapas de profundidad proporcionadas por esta API, que constituyen la entrada del sistema de percepción.

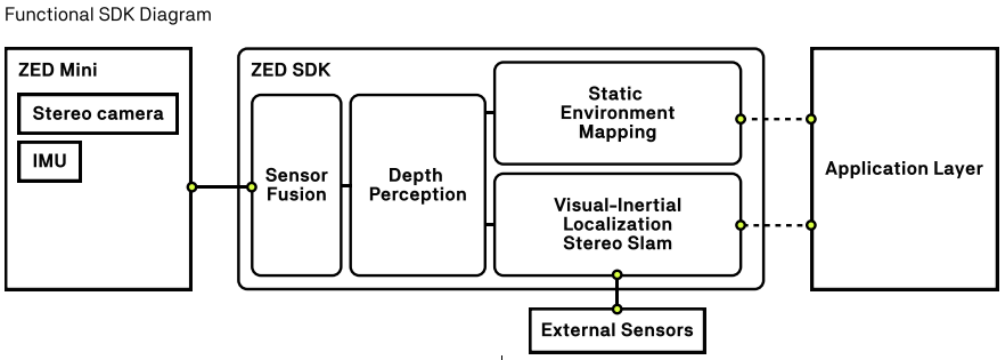


Figura 2.5. Diagrama funcional de ZED SDK. Fuente: [14]

Este esquema de la documentación oficial de Stereolabs de la Figura 2.5 representa la arquitectura general del sistema, destacando los principales módulos de procesamiento y flujo de datos que se activan al operar con la cámara.

2.2 Robot Manipulador xArm6 de UFactory

El robot manipulador empleado en este trabajo es el xArm6 de UFactory (Figura 2.6), un brazo robótico colaborativo de seis grados de libertad diseñado para aplicaciones de automatización industrial, investigación y desarrollo de prototipos.

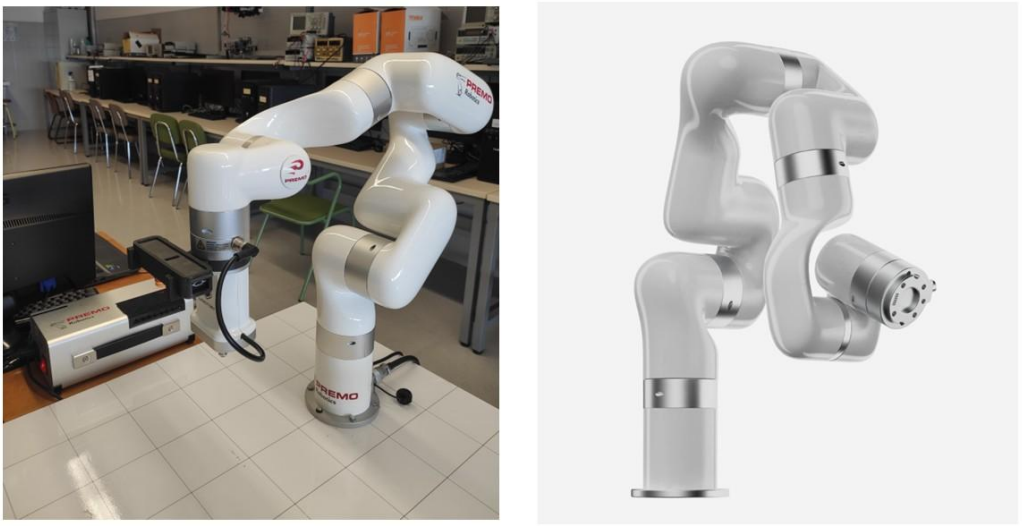


Figura 2.6. Robot Manipulador xArm6 Ufactory (a la izquierda el empleado en este trabajo). Fuente: [16]

Con una carga útil de 5 kg, un alcance de 700 mm y una alta precisión repetitiva (±0.1 mm), este robot destaca por su flexibilidad, facilidad de programación y su diseño compacto que lo hacen especialmente adecuado para integración en laboratorios, líneas de producción y proyectos de robótica avanzada.

La conexión con el controlador se ha realizado siguiendo las indicaciones del manual de usuario de xArm (Figura 2.7).

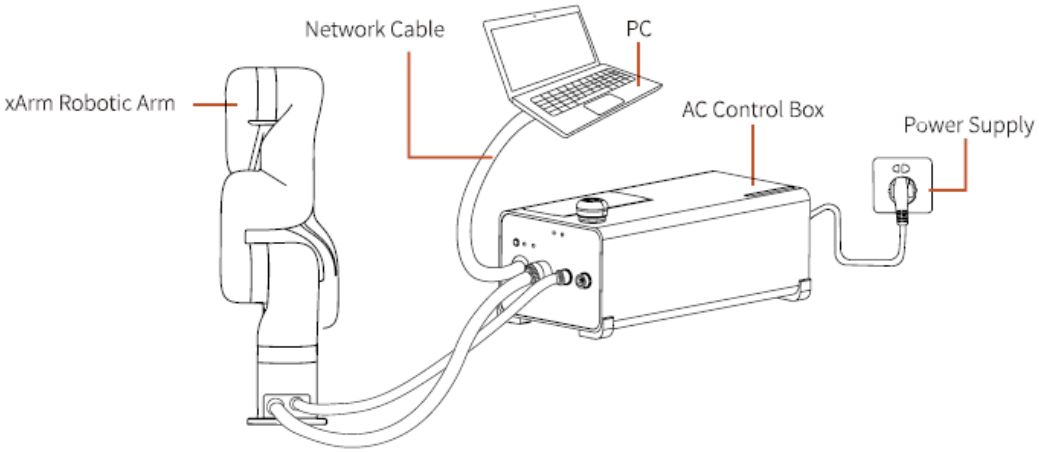


Figura 2.7. Instalación del robot según el manual de usuario de xArm de UFactory. Fuente: [17]

El efector final utilizado en este proyecto ha sido la ventosa oficial de UFactory (Figura 2.8), diseñada específicamente para integrarse con la serie xArm. A diferencia de otras soluciones basadas en sistemas neumáticos, esta ventosa no requiere aire comprimido para su funcionamiento, ya que es alimentada eléctricamente a través de una conexión directa con el manipulador.

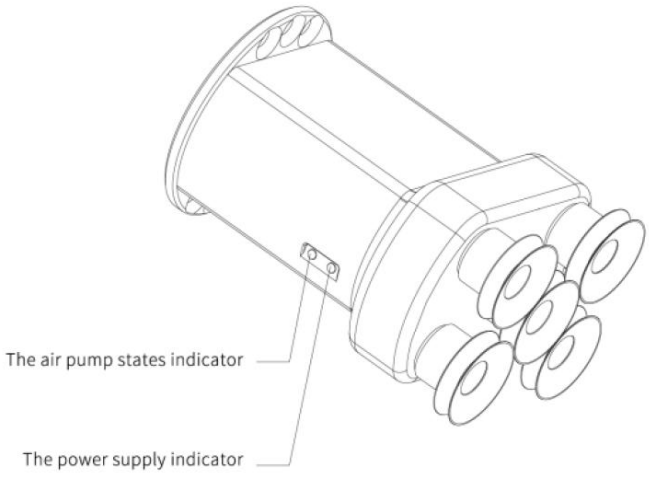


Figura 2.8. Ventosa de UFactory. Fuente: [17]

Para integrar en el manipulador simultáneamente la ventosa y la cámara, se ha empleado el soporte mostrado en la Figura 2.9 diseñado por David Martín Godoy [2] fabricado con impresión 3D, que permite acoplar la cámara en un compartimento exclusivo junto con la ventosa oficial atornillado todo en el extremo del manipulador (Figura 2.10).

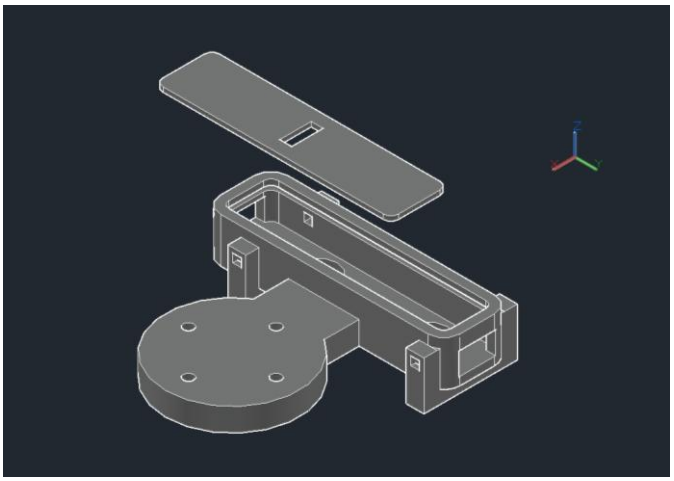


Figura 2.9. Modelo CAD del soporte para la cámara ZED Mini

Figura 2.10. Efector final del manipulador con soporte para la cámara y ventosa

A nivel de software, UFactory proporciona diferentes herramientas y API para facilitar la integración del manipulador en el entorno de desarrollo. En este proyecto se han empleado:

- UFactory Studio

  Es una aplicación web que se ejecuta directamente en el controlador del robot, lo que significa que es accesible desde un ordenador estando conectado, en este caso, por Ethernet (Figura 2.11). En el trabajo se ha empleado instalada y ejecutada localmente en Windows con el fin de realizar pruebas iniciales y visualizar los sistemas de referencias de la base y TCP (*Tool Center Point*) del robot.

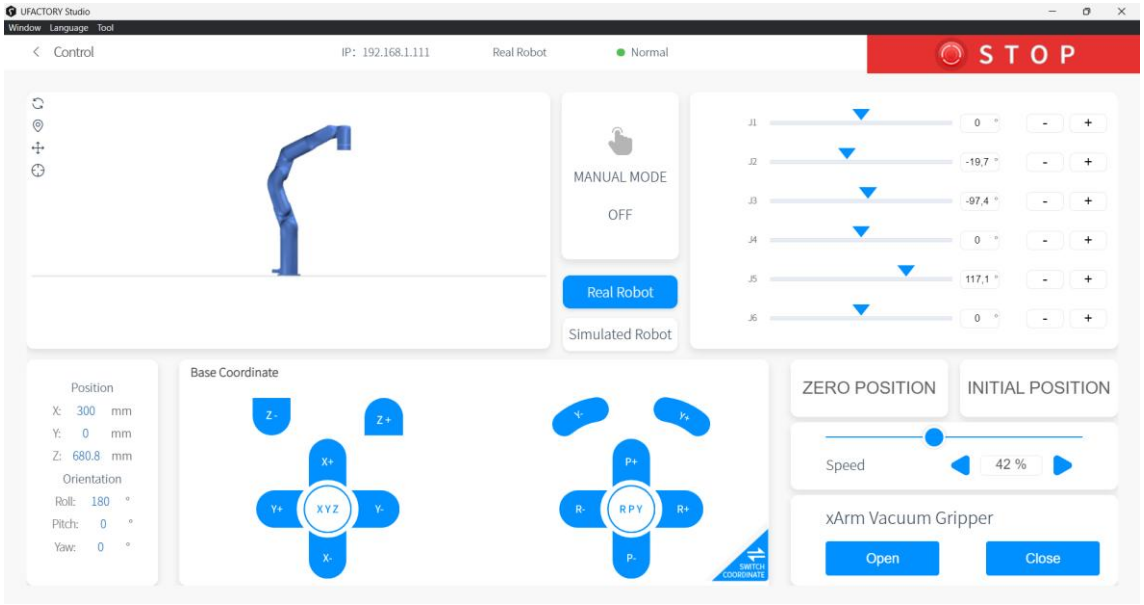


Figura 2.11. Interfaz UFactory Studio en ejecución para xArm6

- Paquete de ROS 2: xarm_ros2 [18]

Este paquete permite configurar, consultar el estado y controlar el robot desde el ecosistema ROS 2. Incluye herramientas para el control de movimiento, definición de referencias, acceso a información del manipulador y visualización en herramientas como Rviz2. Además, proporciona tipos de mensajes y servicios específicos que permiten, por ejemplo, mover el robot a una posición determinada o realizar tareas de planificación.

Es uno de los componentes fundamentales sobre los que se apoya el sistema desarrollado en este trabajo.

2.3 NVIDIA Jetson AGX Xavier

La NVIDIA Jetson AGX Xavier es una plataforma de computación acelerada diseñada para aplicaciones de robótica e inteligencia artificial embebida, que combina alto rendimiento con eficiencia energética en un formato compacto. Forma parte de la familia Jetson™ de NVIDIA, orientada al desarrollo de sistemas autónomos con procesamiento local en el dispositivo (*edge computing*), y es totalmente compatible con el SDK JetPack, que incluye herramientas y bibliotecas optimizadas para el desarrollo de software basado en IA, visión artificial y computación de alto nivel.

Figura 2.12. NVIDIA Jetson AGX Xavier. Fuente: [19]

El modelo utilizado en este proyecto ofrece hasta 32 TeraOPS (billones de operaciones por segundo) de rendimiento computacional con una capacidad de entrada/salida de hasta 750 Gbps, lo que la hace ideal para tareas intensivas en procesamiento como la detección y segmentación en tiempo real.

En este trabajo, se ha empleado la Jetson AGX Xavier como sistema anfitrión, que actuaba como unidad central de procesamiento, conectada con la cámara ZED Mini y el manipulador xArm6. En este dispositivo, residía el entorno de desarrollo que se

detalla a continuación y se ha encargado de ejecutar el ecosistema de ROS 2 que implicaba tanto el procesamiento de imágenes como el control del manipulador. El dispositivo se ha configurado con el entorno de software que se muestra en la Tabla 2.1.

Tabla 2.1. Principales herramientas instaladas en NVIDIA Jetson AGX Xavier

Nombre	Versión	Descripción
Ubuntu	20.04 LTS	Sistema operativo
JetPack	5.2.1	SDK de NVIDIA para Jetson
CUDA	11.4	Plataforma para procesamiento basado en GPU
Visual Studio Code	1.65.1	Editor de código y desarrollo software
Docker	24.0.5	Contenedores para gestión de entornos aislados.

2.4 Entorno de desarrollo: Docker

El entorno de desarrollo en el que se ha compilado el espacio de trabajo de ROS 2 con todos los paquetes que integran el sistema creado reside en un contenedor Docker. Este contenedor proporciona un entorno que contiene todas las librerías, dependencias y herramientas necesarias para la implementación del proyecto.

El contenedor Docker se monta a partir de una imagen. Una de las formas más comunes de generar una imagen personalizada es mediante un *Dockerfile*, es decir, un archivo de instrucciones, que indica los comandos y configuraciones paso a paso que deben ejecutarse para construir el entorno. En este caso, el *Dockerfile* ha sido elaborado desde cero, partiendo de una imagen base oficial de NVIDIA JetPack 5.1.2 (`nvcr.io/nvidia/l4t-jetpack:r35.4.1` [20]), optimizada para dispositivos Jetson y compatible con la versión de Ubuntu y JetPack especificadas anteriormente. El archivo completo se encuentra disponible en el repositorio del proyecto en GitHub (véase Apéndice A).

Las razones que respaldan el uso de un contenedor Docker para crear un entorno de trabajo son: el aislamiento software, que evita conflictos con otras instalaciones en el dispositivo anfitrión; la portabilidad, ya que puede reproducirse el proyecto completo en un dispositivo NVIDIA Jetson de similares características sin

problemas de compatibilidad; la seguridad frente a modificaciones de versiones y dependencias del sistema anfitrión, que, de no tratarse de un entorno independiente, provocaría errores e incompatibilidades; y, finalmente, la posibilidad de acceder, desde el contenedor en ejecución, a los puertos de entrada y salida del equipo anfitrión, que resultaba imprescindible para la conexión con el controlador del robot y la cámara estéreo.

A continuación, la Tabla 2.2 enumera las principales dependencias instaladas en el entorno Docker desarrollado.

Tabla 2.2. Listado de dependencias principales instaladas en el contenedor Docker

Nombre	Versión	Descripción
Python	3.8.10	Lenguaje de programación base.
pyzed	4.2	API para integración de cámaras ZED con Python.
ZED SDK	4.2.5	Kit oficial de desarrollo para cámaras ZED
Ultralytics	8.3.91	Librería oficial de Python para modelos YOLO
ROS 2 Foxy	-	Sistema operativo para robots en el que se basa el proyecto.
OpenCV	4.11.0.86	Biblioteca de procesamiento de imágenes.
PyTorch	2.2.0	Framework para redes neuronales y *deep learning*
torchvision	0.17.2	Librería complementaria a PyTorch para visión por computador
TensorRT	8.5.2.2	Motor de inferencia de NVIDIA para optimización de redes neuronales.
NumPy	1.23.5	Librería para computación numérica de Python

Durante el desarrollo del trabajo, el espacio de trabajo de ROS 2 ha sido un volumen compartido alojado en el dispositivo anfitrión. Esta configuración ha permitido programar cómodamente desde Visual Studio Code, mientras que la compilación y ejecución del sistema se realizaban íntegramente dentro del contenedor Docker, que incluye todas las librerías y dependencias necesarias para su correcto funcionamiento.

Robot Operating System (ROS) es una infraestructura de desarrollo ampliamente adoptada en robótica que facilita la creación de sistemas modulares y escalables.

Proporciona un entorno en el que distintos componentes (como sensores, controladores y actuadores) pueden comunicarse entre sí mediante una arquitectura distribuida.

En este entorno, el espacio de trabajo (ROS 2 *workspace*) representa el conjunto organizado de paquetes, nodos y configuraciones que definen el comportamiento del sistema robótico. Sus elementos fundamentales son:

- **Nodos**: procesos independientes que ejecutan tareas específicas.
- **Tópicos**: canales de comunicación que permiten a los nodos intercambiar mensajes de forma asíncrona mediante un modelo publicador/suscriptor.
- **Servicios**: estructuras de comunicación síncrona, donde un nodo realiza una solicitud y otro responde.
- **Archivos *launch***: códigos que permiten iniciar varios nodos y configuraciones simultáneamente.

Este enfoque modular y estandarizado permite integrar fácilmente distintos sensores, sistemas de percepción o algoritmos de control, como los utilizados en este proyecto.

2.5 Red neuronal para segmentación de instancias: Ultralytics YOLO

Uno de los principales objetivos de este trabajo es la segmentación de objetos captados por la cámara para la posterior manipulación flexible. Por este motivo, se justifica a continuación la herramienta elegida para llevar a cabo la tarea de identificación y segmentación.

En el ámbito de la visión artificial, una red neuronal es un modelo computacional inspirado en el cerebro humano, diseñado para aprender automáticamente patrones complejos a partir de grandes cantidades de datos. Entre los distintos tipos de redes, las redes convolucionales (CNN) son las más empleadas en tareas de procesamiento de imágenes, ya que permiten extraer de forma eficiente características espaciales mediante filtros entrenables.

Existen diversas arquitecturas para detección de objetos basadas en CNN. Una de las más conocidas es Faster R-CNN, que sigue un enfoque en dos etapas: primero propone regiones de interés en la imagen; y luego, en una segunda etapa, clasifica y ajusta cada región. Aunque este enfoque ofrece una alta precisión, su coste computacional lo hace menos adecuado para aplicaciones en tiempo real.

En contraposición, YOLO (*You Only Look Once*) [21] es una arquitectura de tipo *single-shot* que realiza detección y clasificación en una sola pasada, lo que la hace significativamente más rápida con un buen compromiso entre velocidad y precisión. Por estas razones, se ha optado por la implementación de Ultralytics YOLO, una solución moderna, ligera y fácilmente integrable. Ultralytics es una organización líder en visión por computador especializada en desarrollo de modelos de detección de objetos en tiempo real.

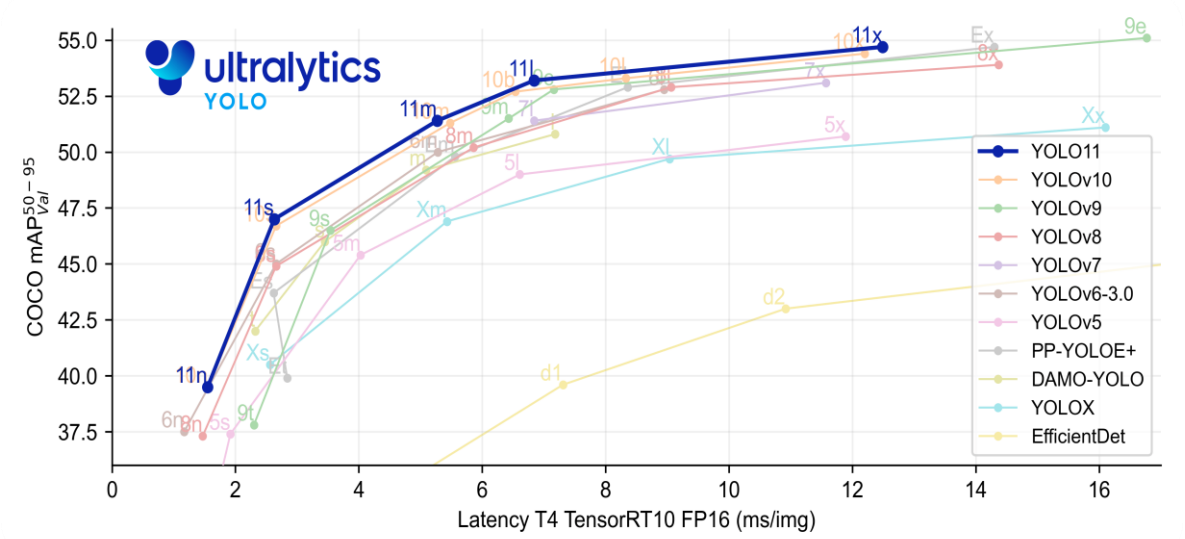


Figura 2.13. Comparativa de precisión ((mAP$_{50-95}$ en COCO) frente a latencia de inferencia (tiempo en milisegundos por imagen con TensorRT FP16 en GPU T4) para distintas versiones de YOLO y modelos similares. Fuente: [22]

En el presente trabajo, se ha empleado específicamente YOLO 11, una de las versiones más recientes del modelo, que presenta características especialmente adecuadas para su aplicación en este proyecto. En la Figura 2.13 se muestra la relación entre precisión (mAP$_{50-95}$ una métrica estándar que evalúa la calidad de las predicciones del modelo) y latencia de inferencia (ms por imagen), donde YOLO 11 destaca por ofrecer una alta precisión con una latencia muy reducida, lo que lo hace especialmente útil para aplicaciones robóticas en tiempo real. Entre sus

funcionalidades se encuentran: detección de objetos, segmentación de instancias, clasificación, estimación de poses y detección con cajas orientadas (*Oriented Bounding Boxes*). En este caso, el objetivo principal es la segmentación de instancias, que permite obtener las máscaras correspondientes a cada objeto identificado en la imagen.

Ultralytics proporciona modelos preentrenados de distintas profundidades y tamaños, desde versiones *nano* hasta *extra-large*, optimizados para tareas específicas. Considerando el equilibrio entre precisión en la inferencia y coste computacional, en este trabajo se ha seleccionado el modelo *YOLO11l-seg*, correspondiente al tamaño medio-alto.

Además, la API de Ultralytics YOLO permite abordar todo el ciclo de vida de un modelo: desde el entrenamiento, validación y predicción, hasta la exportación para despliegue, seguimiento de objetos en vídeo (*tracking*) y análisis de rendimiento hardware (*benchmarking*). Todas estas funcionalidades están integradas en una interfaz unificada, lo que facilita su implementación dentro de flujos de trabajo complejos.

Para este proyecto, se ha empleado específicamente la tarea de segmentación con seguimiento, que permite asignar un identificador único a cada objeto detectado. Esto resulta de gran utilidad para el rastreo temporal de los objetos en el espacio de trabajo, y es fundamental para el correcto funcionamiento del sistema desarrollado, como se detallará en el capítulo tercero de esta memoria.

3 Desarrollo del proyecto

El sistema desarrollado en este trabajo cumple las siguientes tareas:

1. Identificación y segmentación de objetos en el espacio de trabajo
2. Cálculo de la posición óptima de agarre
3. Manipulación del objeto
4. Control por parte del usuario para operar el sistema

El sistema desarrollado está implementado sobre ROS 2, por lo que se han desarrollado tres paquetes, cada uno especializado en distintas tareas que, en conjunto, constituyen la funcionalidad global del sistema. En primer lugar, se describen individualmente cada paquete para, a continuación, explicar la intercomunicación entre ellos.

Los paquetes desarrollados en este trabajo requieren de los siguientes paquetes que se encuentran instalados en el entorno de desarrollo Docker:

- **xarm_ros2**: API y drivers para el control del brazo robótico xArm de UFactory.
- **moveit2**: Framework de planificación y ejecución de movimientos para robots manipuladores [23].
- **vision_opencv**: Conjunto de utilidades para la integración de OpenCV con ROS 2 [24].
- **nmea_msgs**: Paquete que provee mensajes ROS para la comunicación y procesamiento de datos provenientes de dispositivos que usan el protocolo NMEA [25].

Los paquetes moveit2 y nmea_msgs no se emplean de forma directa, pero forman parte de la cadena de dependencias necesarias para el correcto funcionamiento de los paquetes que sí se utilizan directamente en el desarrollo.

3.1 Sistema de visión

El problema de visión en este trabajo aborda la identificación de los objetos en el espacio de trabajo, la segmentación de estos, la obtención del punto de agarre sobre la superficie del objeto deseado y la inclinación óptima con la que el efector final, en este caso concreto una ventosa, debe aproximarse a ella.

Cada uno de estos problemas son abordados por las diferentes tareas que realiza el nodo dedicado al sistema de visión (`yolo_zed_vision_node`) del paquete creado `zed_vision`.

En primer lugar, se realiza la configuración inicial de la cámara para obtener sus parámetros intrínsecos (necesario para aplicar las ecuaciones del modelo de la cámara que permite proyectar puntos en el espacio tridimensional real desde píxeles en el plano imagen) y se especifica el sistema de referencia usado de entre los posibles ofrecidos por la API de Stereolabs (Figura 3.1). En este proyecto se ha empleado el sistema de coordenadas denominado *COORDINATE_SYSTEM_IMAGE*, debido a su similitud con el sistema de referencia del TCP del manipulador, que facilita la transformación homogénea que se detallará más adelante en este capítulo.

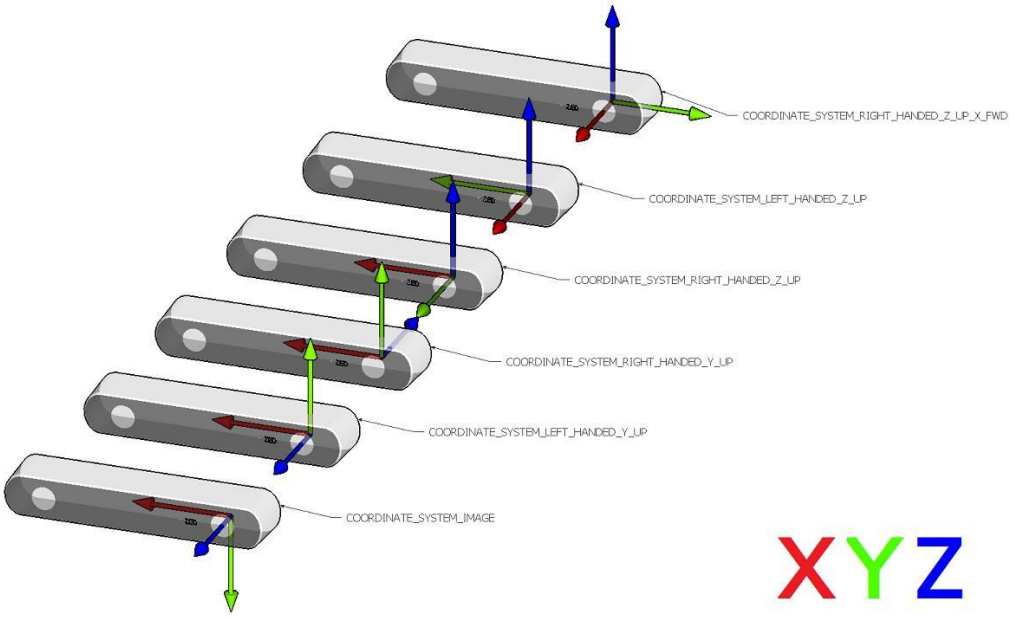


Figura 3.1. Sistemas de referencia posibles según documentación de Stereolabs. Fuente: [15]

Seguidamente, se prepara el modelo de red neuronal a emplear, como se ha mencionado en la sección 2.5, se empleará YOLO 11 de Ultralytics en su modelo para segmentación.

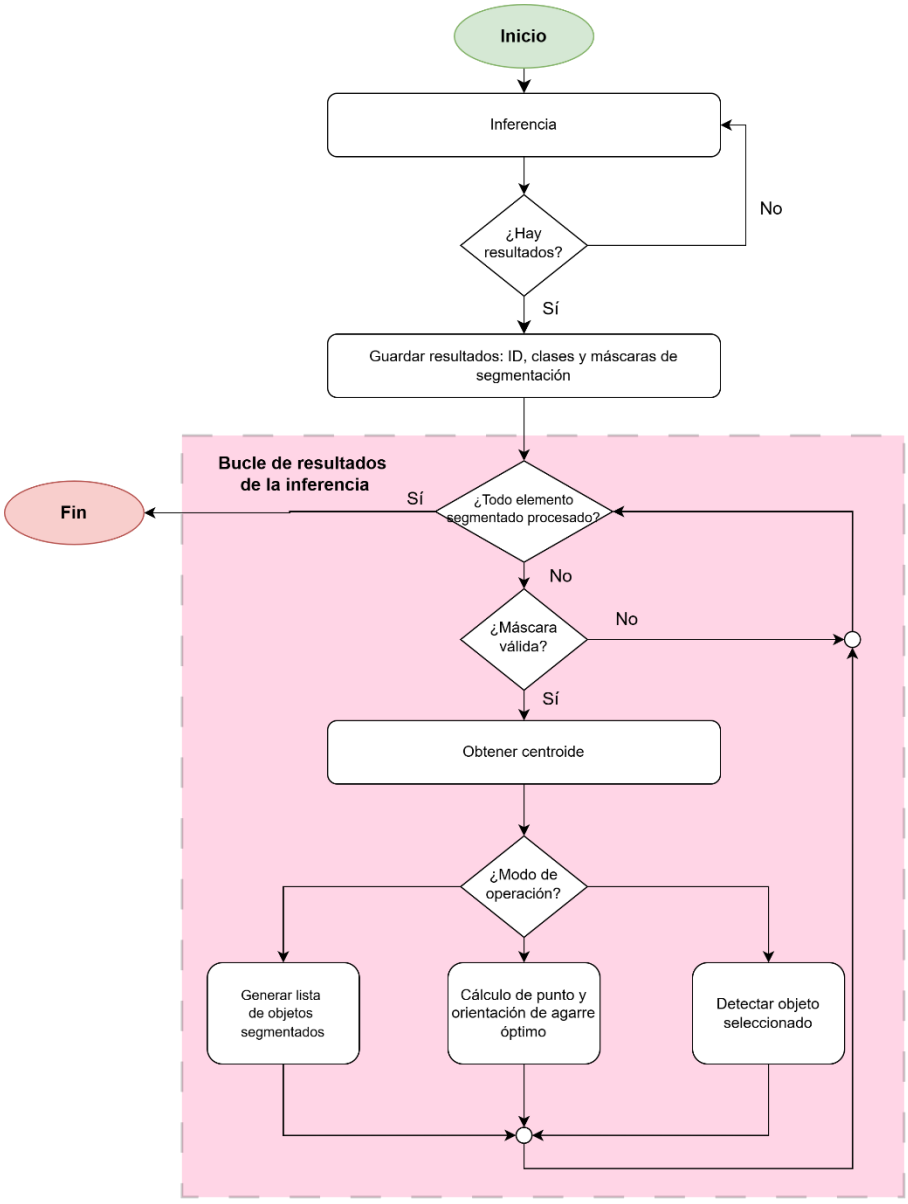


Figura 3.2. Diagrama de flujo de la lógica principal del sistema de visión

Cada vez que se ejecuta la función principal (Figura 3.2), que ocurre periódicamente gracias a un temporizador, se realiza una inferencia del fotograma actual captado por la cámara. Se obtiene una lista con los resultados: cajas delimitadoras, máscaras de segmentación, ID, clase de cada objeto identificado.

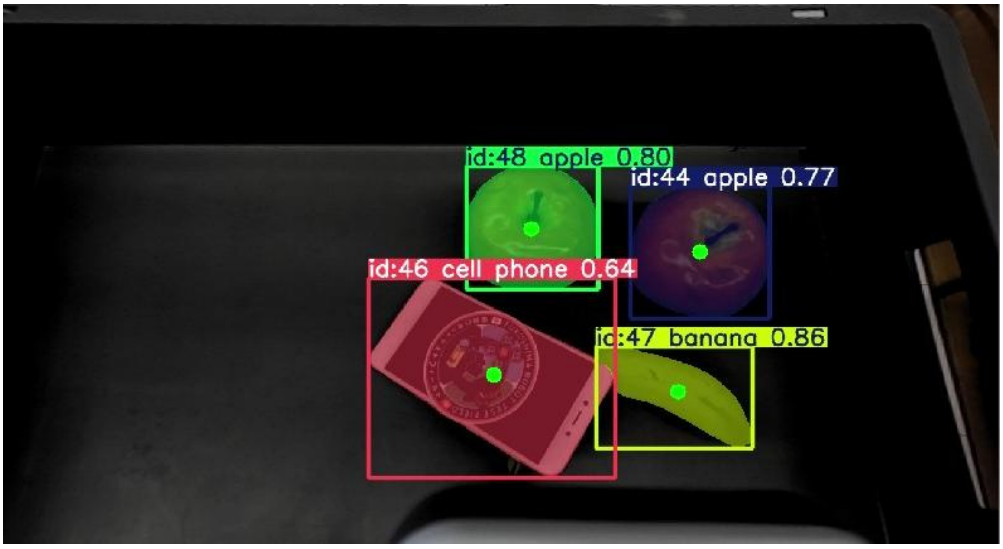


Figura 3.3 Ejemplo de segmentación semántica para detección y clasificación de objetos

En la Figura 3.3 se muestran objetos detectados mediante segmentación semántica. Las máscaras transparentes indican las regiones segmentadas para cada objeto. Las cajas delimitadoras contienen el ID del objeto y la confianza de detección. Los círculos verdes representan los centroides de cada objeto calculados a partir de las propias máscaras.

Durante el desarrollo y las pruebas del algoritmo del sistema de visión, se ha detectado la necesidad de aplicar un preprocesamiento a la imagen capturada por la cámara antes de ser analizada por la red convolucional. Este preprocesamiento tiene como objetivo reducir brillos excesivos provocados por las condiciones del entorno, ya que afectan negativamente a la correcta detección de objetos. Para ello, la imagen se transforma al espacio de color HSV y se reduce de forma significativa la componente V (valor o brillo), atenuando así los reflejos y zonas sobreexpuestas que interfieren en el rendimiento del modelo de visión.
Este ajuste ha demostrado mejorar de forma notable la estabilidad de las predicciones, especialmente en entornos con iluminación intensa y presencia de reflejos que dificultan la detección.

A continuación, se recorre la lista de resultados y, para cada objeto identificado, se comprueba la validez de la máscara para detectar posibles errores o inferencias poco precisas. Si el resultado es válido, se procesa la información individual de

cada objeto según la tarea que sea necesaria en cada etapa del proceso de manipulación, diferenciando tres:

1. Generación del listado de objetos segmentados
2. Cálculo del punto y vector normal para un agarre óptimo
3. Detectar objeto seleccionado en escena

3.1.1 Generación del listado de objetos segmentados

De cada objeto segmentado se necesita: ID, clase y las coordenadas del punto de primera aproximación al objeto que se define como el centroide de las máscaras segmentadas, pero proyectado en el espacio tridimensional sobre la superficie del objeto real. Es el primer punto con el que el manipulador se aproxima al objeto deseado, por este motivo, no necesita gran precisión.

En la página siguiente, se muestra el diagrama de flujo del algoritmo implementado para esta tarea (Figura 3.4).

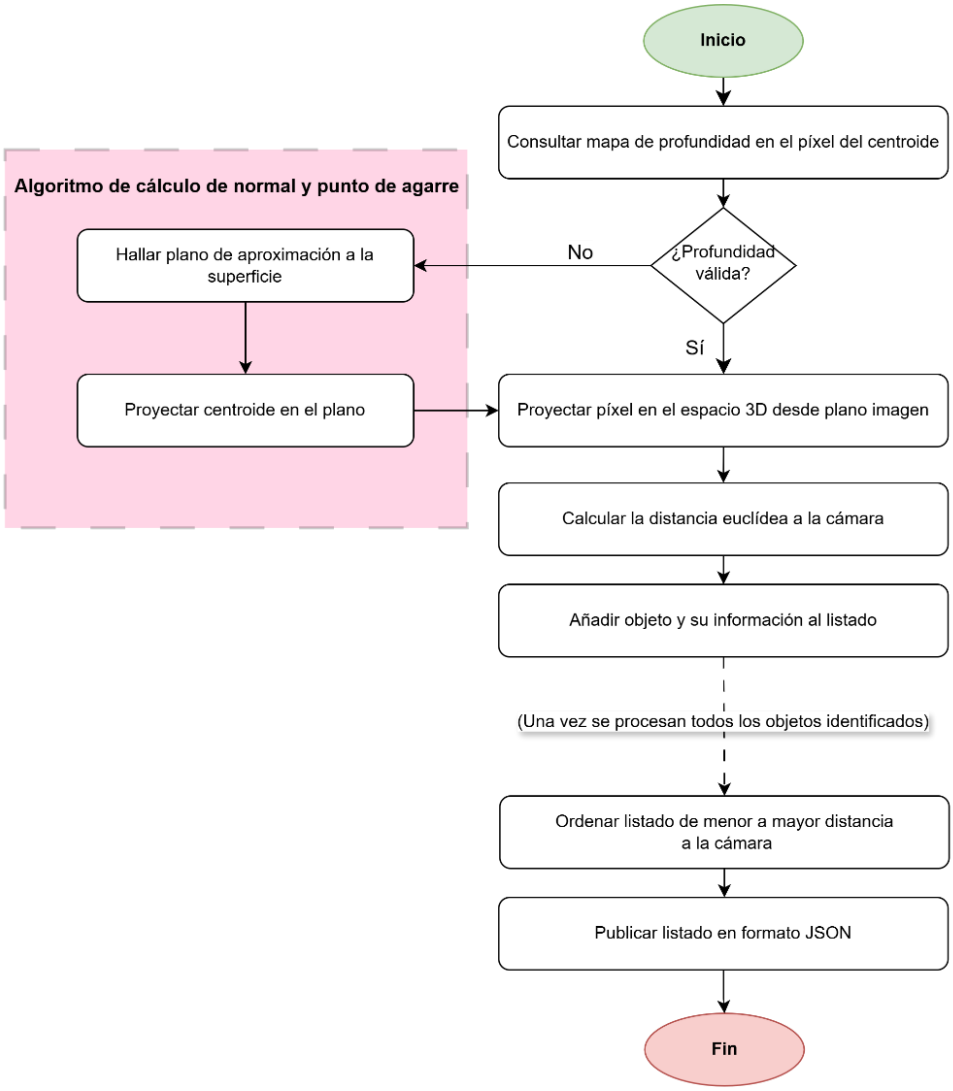


Figura 3.4. Diagrama de flujo de la generación del listado de objetos segmentados

Se consulta la componente Z del píxel del centroide de la máscara en el mapa de profundidad proporcionado por la cámara. No obstante, la medida de profundidad puede ser errónea y, en esos casos, hay que estimarla obteniendo una superficie que se aproxime a la real del objeto tomando puntos cuyas medidas sí sean válidas en alrededores al punto de consulta. La superficie más sencilla de obtener es un plano, en un entorno infinitesimalmente próximo al punto de interés. La superficie más sencilla de obtener es un plano local en torno al punto de interés. El algoritmo implementado para ello es el mismo que se utiliza posteriormente para estimar la orientación del agarre, por lo que se describe con mayor detalle en la siguiente sección sobre la tarea de “Obtención del punto y vector normal para un agarre óptimo”.

Una vez se obtiene la profundidad del centroide, se proyecta en el espacio tridimensional usando las ecuaciones 3.1 y 3.2:

$$x = (j - c_x) \cdot \frac{z}{f_x} \tag{3.1}$$

$$y = (i - c_y) \cdot \frac{z}{f_y} \tag{3.2}$$

donde (x, y) son las coordenadas en el espacio en el sistema de referencia de la cámara, (i, j) son las coordenadas del píxel en la imagen, (c_x, c_y) es el centro óptico de la cámara, (f_x, f_y) son las distancias focales en píxeles y z es la profundidad del punto medida como distancia a la cámara.

Este procedimiento corresponde con el modelo de cámara *pinhole*, comúnmente utilizado en visión por computador para describir la proyección de puntos 3D al plano imagen [26]. En este caso, se aplica el proceso inverso: a partir de una coordenada en el plano imagen y una medida de profundidad, se calcula la posición correspondiente en el espacio tridimensional respecto al sistema de coordenadas de la cámara.

Se calcula la distancia euclídea del punto de aproximación a la cámara mediante las coordenadas obtenidas y se registra en la lista el objeto detectado completando con los datos obtenidos.

Una vez procesados todos los objetos detectados, se ordena la lista de resultados de menor a mayor distancia a la cámara. De esta manera, se prioriza la manipulación de los objetos más próximos, lo que reduce la posibilidad de intentar recoger un objeto parcialmente oculto por otro en primer plano, facilitando así una manipulación más directa y eficiente. Finalmente, los resultados se publican en formato JSON (*JavaScript Object Notation*).

3.1.2 Obtención del punto y vector normal para un agarre óptimo

La nube de puntos proporcionada por la cámara presenta una inestabilidad demasiado elevada para esta aplicación. Por este motivo, se ha decidido utilizar el mapa de profundidad que ofrece una mayor estabilidad. Este mapa se representa como una matriz de píxeles, similar a una imagen convencional, donde a cada píxel se le asigna un valor de profundidad, expresado en milímetros. El conjunto de estos valores define una función escalar que describe la geometría superficial del objeto proyectado en el plano imagen. Esta característica es la base del método implementado para determinar la posición de agarre.

Se parte de la siguiente premisa: el punto de agarre debe situarse lo más cerca posible del centroide de la máscara de segmentación. Desde un punto de vista físico, esta ubicación representa la zona de mayor estabilidad para la manipulación del objeto y de compensación más efectiva frente a la gravedad. Además, al encontrarse dentro de la máscara segmentada, se garantiza que pertenece al objeto. Asimismo, la dirección óptima de aproximación para una herramienta de vacío (ventosa) es la normal a la superficie del objeto en el punto de agarre.

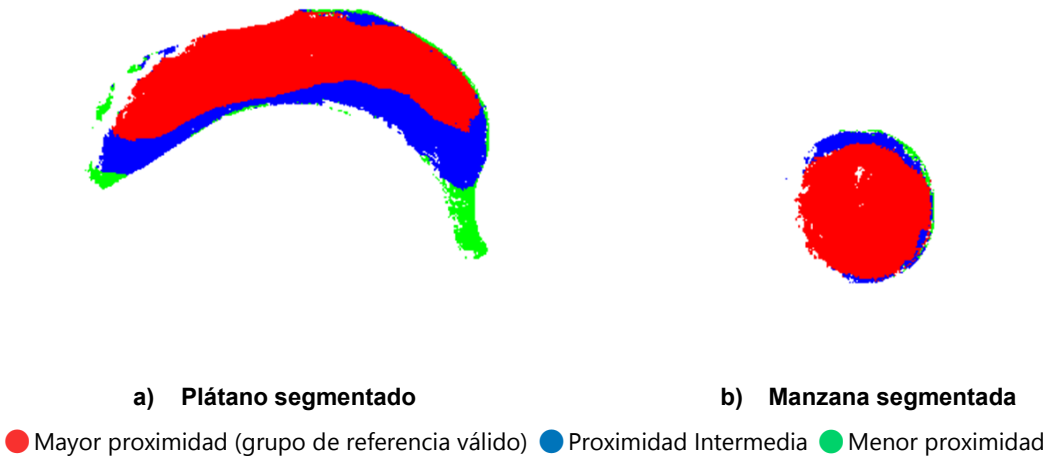


Figura 3.5. Agrupación de los píxeles de la máscara segmentada según profundidad mediante el algoritmo K-Medias.

Para identificar con precisión dicho punto, se consideran únicamente los píxeles del mapa de profundidad contenidos dentro de la máscara del objeto segmentado. Estos puntos se agrupan en tres conjuntos utilizando el algoritmo K-Medias (Figura 3.5). De los tres grupos generados, se toma como referencia aquel cuyos puntos presentan menor profundidad, es decir, los más próximos a la cámara.

La lógica implementada se muestra en el diagrama de la Figura 3.6.

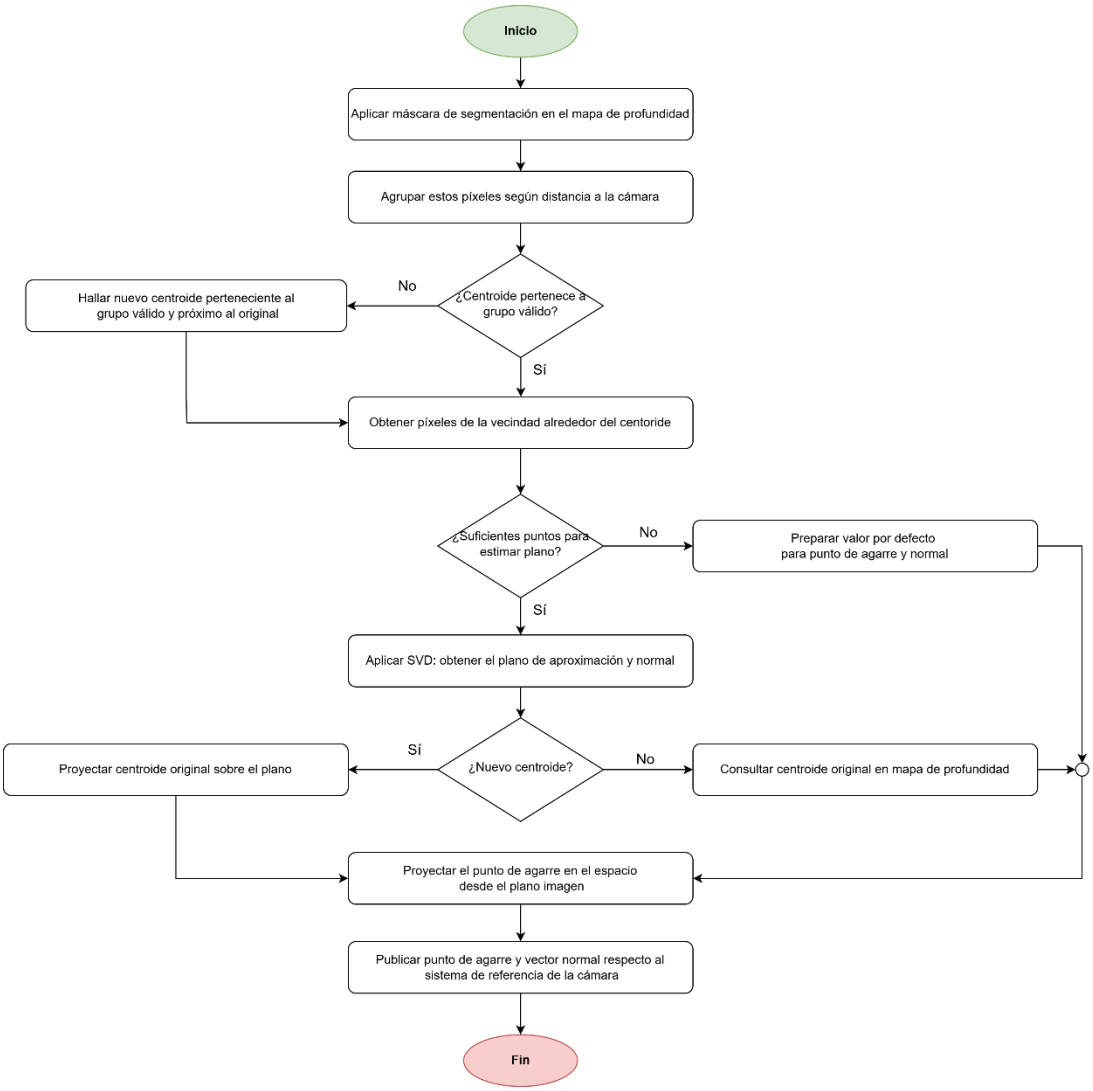


Figura 3.6. Diagrama de flujo de la obtención del punto y la orientación de agarre

En primer lugar, debe verificarse si el centroide se encuentra en la región válida, de lo contrario, se selecciona como centroide un píxel que, perteneciendo a la región válida, esté en una vecindad determinada del centroide original para disminuir el error en el cálculo la superficie de aproximación en torno a este.

El cálculo de la orientación óptima para el agarre requiere conocer la normal a la superficie del objeto en el punto seleccionado. Para ello, se estima un plano local a partir de la información del mapa de profundidad. Este plano se ajusta sobre el conjunto de puntos formado por el centroide de la máscara de segmentación y una vecindad espacial a su alrededor. El objetivo de esta estrategia es obtener una estimación robusta y estable entre fotogramas consecutivos, evitando variaciones abruptas que comprometan la fiabilidad del agarre.

Existen diferentes técnicas para el ajuste de planos en nubes de puntos. Un enfoque habitual en la literatura es el uso de RANSAC (*Random Sample Consensus*), debido a su alta tolerancia al ruido. Sin embargo, este método introduce cierta aleatoriedad, depende de parámetros como el número de iteraciones y el umbral de aceptación, y puede ser computacionalmente costoso. Es un método muy adecuado para entornos con alta incertidumbre o sin segmentación previa, pero no resulta tan eficiente para aplicaciones en tiempo real con datos prefiltrados.

En este trabajo, el preprocesado mediante segmentación y agrupación (*clustering)* permite partir de conjuntos de puntos limpios, pertenecientes mayoritariamente a una única superficie. Gracias a ello, es posible utilizar un método más eficiente como la descomposición en valores singulares (*Singular Value Decomposition*, SVD).

El procedimiento consiste en centrar los puntos respecto a su centroide y aplicar directamente SVD sobre la matriz de puntos centrados. El vector singular correspondiente al menor valor singular define la dirección de menor variabilidad del conjunto, es decir, la normal a la superficie local. A partir de esta normal y el centroide, se obtiene la ecuación del plano que mejor se ajusta a los datos. Este procedimiento permite, si es necesario, proyectar sobre ese plano el centroide cuando la medida de profundidad original no es válida, como se ha mencionado en la sección anterior.

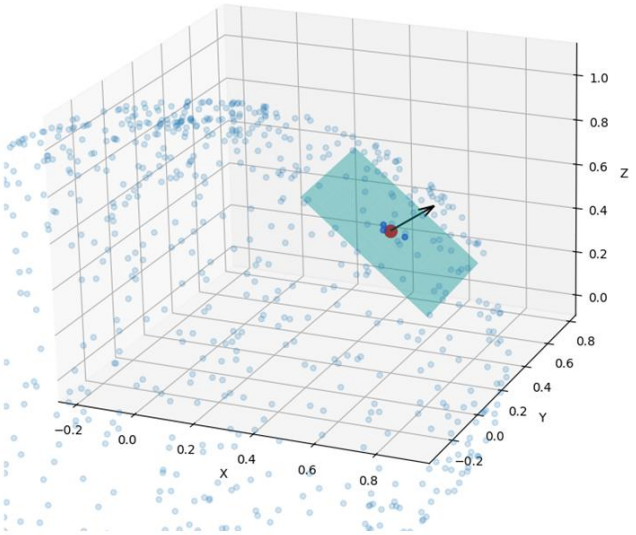


Figura 3.7. Ejemplo de estimación de plano tangente a una superficie mediante SVD

En la Figura 3.7 se ilustra lo que se consigue con este método: donde para una superficie dada, se toman puntos de una región local (en azul) centrada en el centroide (punto rojo) y se obtiene la normal (vector negro). El código utilizado para generar esta figura se encuentra disponible en el repositorio del proyecto (véase Apéndice A).

Esta técnica es determinista, no requiere parámetros adicionales y ofrece una alta precisión cuando los datos están bien condicionados, como es este caso. Además, presenta un bajo coste computacional, lo que la hace especialmente adecuada para aplicaciones que requieren respuesta en tiempo real.

El cálculo se realiza mediante la función SVD de la librería *NumPy* en *Python* y se asegura que el vector normal resultante apunte hacia la cámara, es decir, sea saliente de la superficie. (Figura 3.8).

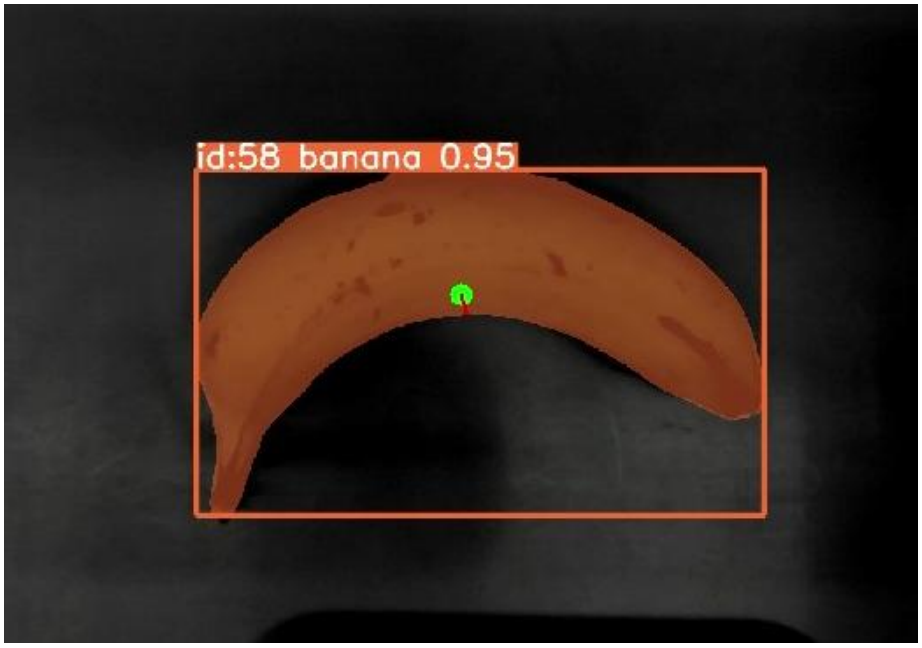


Figura 3.8. Representación del resultado del cálculo del punto y normal de la superficie de agarre en el proceso de manipulación. Punto de agarre en verde y normal a superficie en rojo (representación bidimensional)

Si no hay suficientes puntos válidos en la vecindad del centroide (mínimo tres, por definición), se asigna como vector normal uno paralelo al eje z del sistema de referencia de la cámara.

Este algoritmo no solo calcula el vector normal a la superficie, sino que también determina el punto de agarre. El punto de agarre es el centroide original de la máscara de segmentación cuya profundidad se obtiene directamente del mapa de profundidad o, si el valor no es válido, mediante su proyección sobre el plano estimado. Este mismo procedimiento ya fue empleado previamente durante la identificación de objetos, aunque, en este caso, se ejecuta con la cámara situada a menor distancia del objeto y con una orientación más favorable, lo que permite aumentar la precisión del resultado.

En resumen, de entre todos los objetos identificados en la escena actual, únicamente se procesa aquel que ha sido seleccionado que es identificable gracias al ID de seguimiento asignado por la inferencia. El resultado es el punto de agarre proyectado desde el plano imagen al espacio tridimensional en el sistema de referencia de la cámara (ecuaciones 3.1 y 3.2) junto con el vector normal correspondiente, que se publican para su uso posterior.

3.1.3 Detección de objeto seleccionado en la escena

Dada la posición relativa de la cámara respecto al efector final, no es posible observar correctamente el objeto durante el agarre. Por este motivo, tras el último avistamiento previo al agarre, es necesario comprobar si dicho objeto continúa en el mismo sitio donde se detectó por última vez. El resultado de esta comprobación permite validar de forma indirecta si la manipulación ha sido correcta, aportando así mayor robustez al proceso.

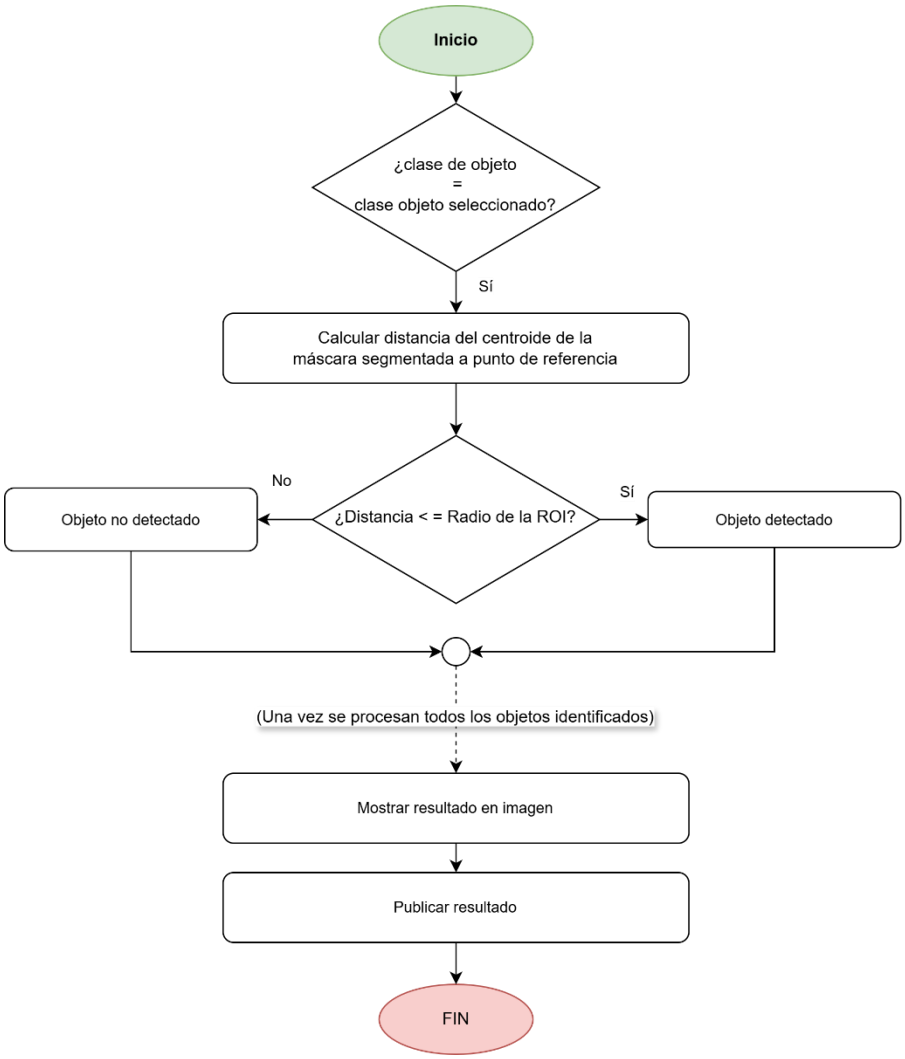
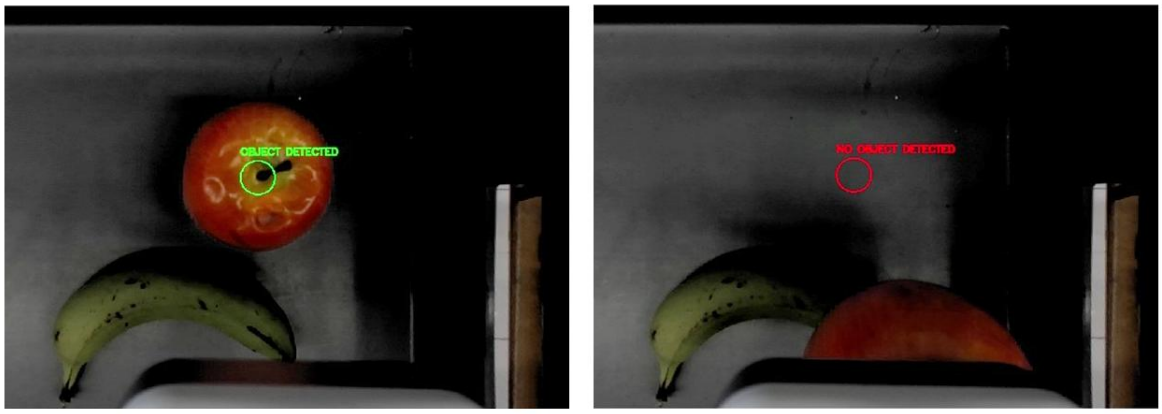


Figura 3.9. Diagrama de flujo de detección de objeto seleccionado

El mecanismo de error implementado (Figura 3.9) consiste en situar nuevamente la cámara en la misma posición previa al agarre y analizar si, en la escena actual, está presente un objeto de la misma clase que el seleccionado. En caso afirmativo, se

evalúa si el centroide de dicho objeto se encuentra dentro de una región de interés (*Region of Interest*, ROI) definida alrededor de la posición previamente registrada. Si se cumple esta condición, se interpreta que el objeto no ha sido retirado y, por tanto, que el agarre ha fallado. En cambio, si no se detecta el objeto dentro de dicha ROI, se asume que el agarre ha sido exitoso (Figura 3.10).

En caso de fallo, el sistema activa un reintento del agarre, como se detallará en la siguiente sección dedicada a la lógica de control del manipulador.



a) Objeto detectado: Agarre fallido **b) Objeto no detectado: Agarre exitoso**
Figura 3.10. Mecanismo de detección de objeto seleccionado para verificación de agarre

Este enfoque, aunque limitado, responde a la imposibilidad de emplear el identificador de seguimiento habitual ya que en el agarre el objeto puede salir del campo de visión y perder su identificador. Es una primera aproximación funcional al problema de verificación de agarre sin necesidad de sensores adicionales. Además, esta comprobación es rápida y suficiente en entornos controlados, como el que se plantea en este trabajo.

Supondrá un reintento en el agarre, que se detallará en la descripción de sistema de manipulación.

3.2 Control del manipulador

El control del robot manipulador se realiza a través de los servicios proporcionados por el paquete xarm_ros2 [18], que permite controlar el robot desde ROS 2 mediante una interfaz de servicios y tópicos. Los empleados en este proyecto son:

- `/xarm/robot_states`: Tópico por el que el controlador publica el estado, modo y pose actual del robot.
  Los valores del campo de estado considerados en este proyecto son:
  - 0 – Habilitado: Manipulador operativo y puede ejecutar comandos.
  - 1 – En ejecución: Está ejecutando un comando de movimiento.
  - 2 – En reposo: Detenido, pero preparado para recibir comandos.
  - 4 – Error: El robot ha detectado una condición de error. Es necesario limpiar el error antes de continuar.
- `/xarm/set_mode`: Servicio para establecer el modo. En este proyecto se emplea el modo cero que permite enviar posiciones objetivo al efector final de manera controlada. Existen otros modos disponibles (por ejemplo, control reactivo o enseñanza manual), aunque no son utilizados en este caso.
- `/xarm/set_state`: Servicio para establecer el estado.
- `/xarm/set_position`: Servicio para comandar movimiento. Las coordenadas de la posición deben estar en metros en el sistema de referencia de base del robot y orientación en ángulos de Euler XYZ (*roll,pitch,yaw*).
- `/xarm/clean_error`: Servicio para limpiar errores. Esto es necesario para restablecer el sistema tras estado de error.
- `/xarm/set_vacuum_gripper`: Controla el efector final de tipo ventosa. Este servicio activa o desactiva el sistema de vacío para sujetar o liberar objetos.
- `/xarm/motion_enable`: Servicio para habilitar el movimiento de las articulaciones.

La correcta administración de las llamadas a servicios e información recibida es llevada a cabo por el nodo `xarm6_controller_node` del paquete creado `xarm6_controller`.

Tras el encendido, es necesario habilitar el movimiento. Durante la ejecución, se consulta continuamente el estado del manipulador. Si se detecta una condición de error, se invoca la limpieza de errores, y se restablece el estado operativo. Luego, se configura el modo de funcionamiento correspondiente y, seguidamente, se define el modo de control. Con esto, el manipulador queda listo para recibir posiciones de destino y actuar sobre el efector final.

Este mecanismo de control constituye la base que sustenta las distintas etapas del proceso de manipulación, expuestas a continuación.

1) **Inicialización y posicionamiento de inicio**
   El manipulador se desplaza a una posición inicial desde la cual la cámara puede visualizar todo el espacio de trabajo. Desde esta vista general, el sistema de visión detecta y registra todos los objetos disponibles.

2) **Espera de orden del usuario**
   El manipulador se mantiene en la posición de inicio a la espera de que el usuario seleccione desde el panel una acción a ejecutar.

3) **Aproximación al objeto**
   Al recibir la orden, el manipulador se mueve a un punto de aproximación situado por encima del objeto seleccionado, con una ligera desviación. Esta posición asegura una mejor visibilidad del objeto por parte de la cámara, que ahora puede calcular con mayor precisión el punto y la orientación de agarre.

4) **Recepción de datos del sistema de visión**
   Se espera la recepción del punto de agarre y el vector normal a la superficie del objeto, calculados por el sistema de visión.

5) **Posicionamiento sobre el punto de agarre**

   El manipulador se sitúa sobre el punto de agarre definitivo, sin aplicar la orientación final.

6) **Orientación del TCP**

   Se ajusta la orientación del efector final en función del vector normal recibido, alineando el TCP adecuadamente para el agarre.

7) **Agarre del objeto**

   El manipulador desciende hasta hacer contacto con la superficie y activa la ventosa.

8) **Elevación del objeto**

   Se eleva el objeto a una altura segura como paso previo al siguiente movimiento.

9) **Regreso a la posición de aproximación**

   El manipulador vuelve a la posición desde la cual tenía mejor visualización del objeto seleccionado, justo antes de realizar el agarre. Esto permite verificar si el objeto sigue presente en esa ubicación y confirmar que el agarre se ha realizado correctamente.

10) **Verificación del agarre**

    Si el sistema detecta que el objeto ha sido correctamente agarrado, se continúa con la manipulación. En caso contrario, se reintenta la operación, repitiendo desde la etapa tres. En cada intento se aplica un descenso adicional de cinco milímetros, asumiendo un posible error de estimación en altura. El número máximo de intentos es parametrizable.

11) **Desplazamiento al destino**

    Se transporta el objeto hasta la posición de destino.

12) **Liberación del objeto**

    Se suelta el objeto en la ubicación destino.

13) Preparación para siguiente ciclo

El sistema se prepara para regresar a la posición de inicio para el siguiente ciclo de manipulación.

A continuación, se destacan algunos aspectos que influyen directamente en el control del manipulador durante estas etapas.

- Transformación al sistema de referencia de la base del robot

  Las posiciones obtenidas por el sistema de visión están expresadas en sistema de referencia de la cámara. Este nodo se encarga de transformarlas en el sistema de referencia de la base del robot, que es el que utiliza el controlador del xArm6.

  Este proceso implica una transformación entre tres sistemas de coordenadas: la cámara, el TCP del robot y la base, compuesta por dos etapas:

  1) Transformación del sistema de la cámara al TCP: dado que la cámara está montada en un soporte fijo respecto al extremo del manipulador, la relación entre ambos sistemas es constante. Para obtener esta transformación se ha recurrido a una medición física sobre el soporte teniendo en cuenta que el sistema de referencia de la cámara se encuentra en la lente izquierda. Las medidas correspondientes a ejes X e Y se muestran en la acotación simplificada de la Figura 3.11. La traslación en el eje Z es muy pequeña en comparación con los otros ejes, por lo que se considera despreciable a efectos prácticos.

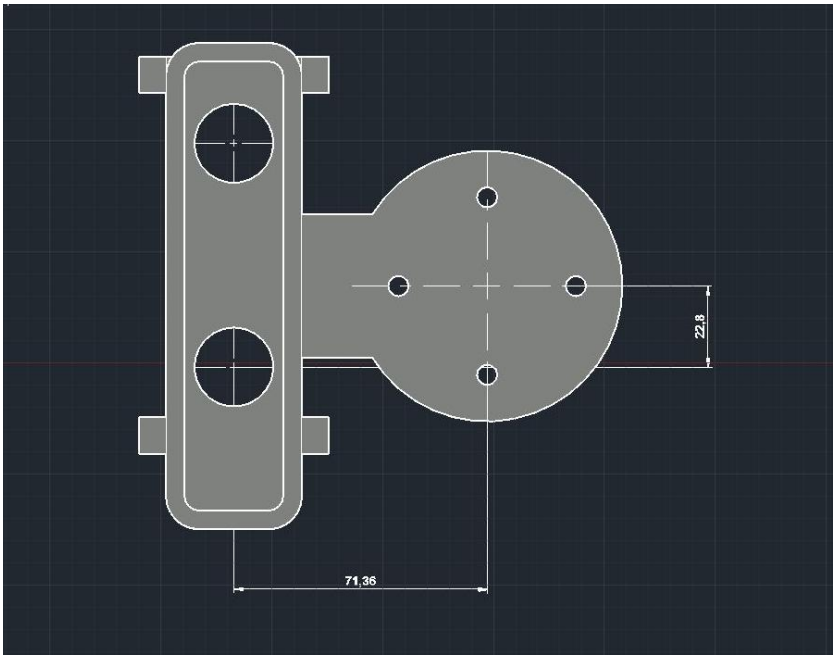


Figura 3.11. Acotación simplificada del modelo del soporte para cámara en milímetros

El sistema de referencia de la cámara se ha especificado en la sección 3.1 sobre el sistema de visión. Asimismo, conocido el sistema del TCP que se puede observar en la Figura 3.12 y las medidas realizadas, se puede determinar la relación de transformación entre ambos sistemas de referencia.

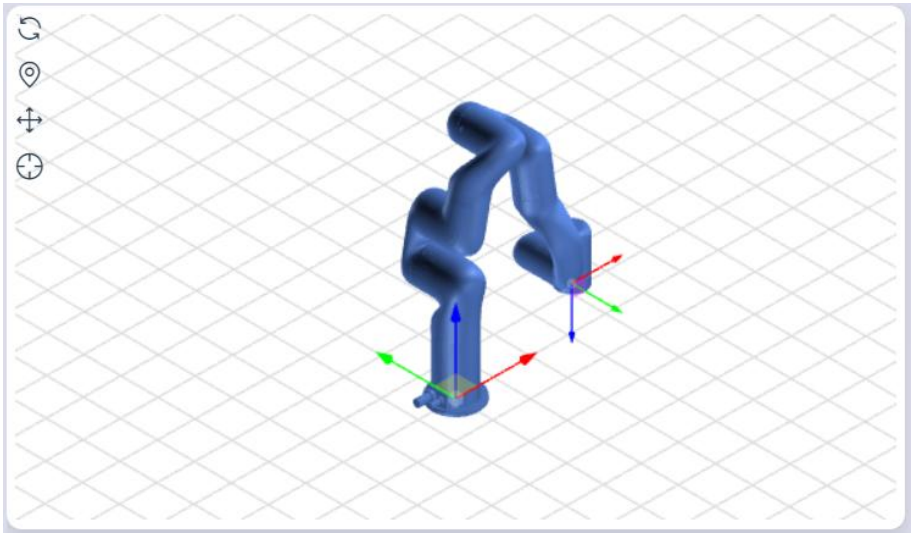


Figura 3.12. Sistemas de referencia de TCP y base del manipulador xArm6 (UFactory Studio)

La relación resultante entre ambos sistemas de coordenadas queda representada por la siguiente matriz de transformación homogénea:

$$T_{Cam}^{TCP} = \begin{bmatrix} \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{0.071} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{-0.023} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

2) Transformación del sistema del TCP a la base del robot. Se obtiene en tiempo real a partir del sistema de transformaciones implementado por el controlador de xArm6. Publica dinámicamente esta información mediante la infraestructura estándar de transformaciones del entorno de ROS permitiendo conocer en cada instante la posición y orientación del TCP respecto a la base del manipulador.

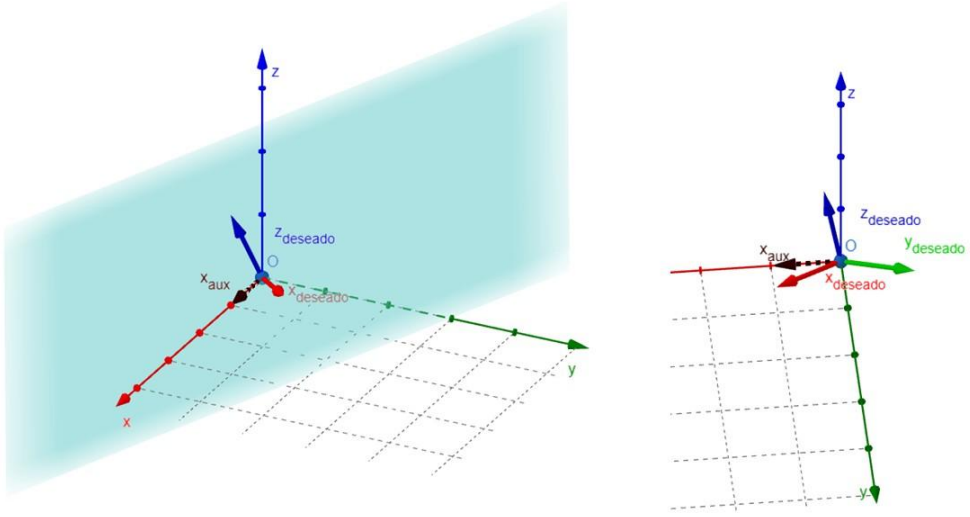
- Obtención de la orientación para alinear el efector final a partir del vector normal a la superficie

  Mientras que la transformación de los puntos (X, Y, Z) consiste en una aplicación directa de transformaciones homogéneas, la orientación requiere un tratamiento adicional. El controlador exige que la orientación se especifique en ángulos de Euler (XYZ), pero la información que se recibe desde el sistema de visión consiste únicamente en el vector normal saliente de la superficie del objeto, expresado en el sistema de referencia de la cámara.

  Este vector normal se transforma primero al sistema de referencia de la base del robot. Luego se invierte el vector resultante para obtener aquel que permite orientar el efector final, esto es, entrante a la superficie y saliente, por tanto, de la cámara.

  No es posible definir una rotación que alinee el sistema del efector final únicamente a partir del vector normal, ya que este solo define el eje Z del sistema deseado. Por ende, se construye un sistema de referencia auxiliar,

pero se busca que su disposición implique la menor rotación posible desde la orientación actual del efector final para evitar giros innecesarios.



a) Proyección ortogonal del vector x auxiliar **b) Obtención del vector Y resultante**

Figura 3.13. Ilustración de la obtención de sistema de referencia auxiliar

Por tanto:

- Se elige un vector auxiliar, inicialmente el eje X del sistema de referencia de la base (1,0,0).
- Luego se calcula el eje X deseado proyectando el vector anterior sobre el plano ortogonal a Z (Figura 3.13 a)
- Finalmente se calcula el eje Y como el producto vectorial entre los Z y X (Figura 3.13 b).

Si el vector auxiliar resulta casi paralelo a Z, se cambia a (0,1,0) y se aplica el mismo proceso análogamente.

La Figura 3.13 ha sido realizada con GeoGebra 3D con fines ilustrativos. El archivo fuente está disponible en el repositorio del proyecto (véase Apéndice A).

Una vez definido el sistema de referencia, se puede hallar la transformación homogénea de rotación que permite alinear el efector final con la orientación deseada. A partir de la matriz de rotación se obtienen los ángulos de Euler XYZ requeridos por el controlador.

- Comprobación de la posición actual del robot

Para verificar si el robot ha alcanzado la pose deseada, se realiza una comparación entre la posición y la orientación actual publicada por el robot y la pose objetivo. La comparación de las componentes de posición puede realizarse directamente considerando una tolerancia definida, mientras que la orientación representa una mayor dificultad.

Las orientaciones del robot están expresadas en ángulos de Euler (*roll, pitch, yaw*), un formato que presenta ambigüedades y discontinuidades cerca de ciertos valores (por ejemplo, ±π), lo que dificulta una comparación directa. Para resolver esto, se transforman tanto la orientación actual como la deseada a cuaterniones. Esta representación evita las discontinuidades de Euler y permite realizar comparaciones más robustas y computacionalmente más eficientes.

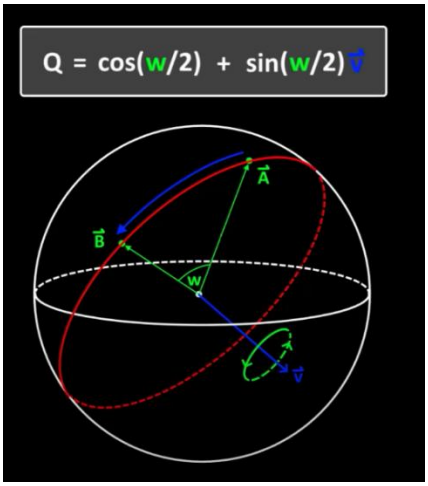


Figura 3.14. Representación gráfica de cuaterniones y el camino más corto entre ambos. Fuente: [27]

La diferencia entre ambas orientaciones se calcula como el ángulo del trayecto más corto sobre la superficie de la esfera unitaria que conecta ambos cuaterniones. En esta representación, el ángulo entre ellos equivale

al ángulo de apertura del arco geodésico que los une, como se ilustra en la Figura 3.14 donde ambos cuaternios son los vectores A y B. Esta diferencia angular ω se compara con una tolerancia predefinida.

De este modo, se garantiza una verificación precisa y confiable del alineamiento espacial del efector final.

3.3 Interfaz gráfica de control y visualización en RViz2

Para aprovechar al máximo el sistema de manipulación desarrollado en este proyecto, se ha implementado una interfaz gráfica de control que permite al usuario seleccionar qué objeto o clase recolectar, así como recibir información en tiempo real sobre la etapa del proceso de manipulación en la que se encuentra el sistema.

Esta interfaz se ha integrado como un panel personalizado dentro de RViz2, la herramienta de visualización tridimensional del ecosistema ROS. RViz2 se emplea para representar tanto el estado actual del robot como los elementos de su entorno. En este caso, el paquete del controlador ofrece un *launch file* que permite abrir una instancia de RViz2 conectada en tiempo real al robot físico (Figura 3.15). Asimismo, se enriquece la visualización incluyendo la imagen procesada por el nodo de visión.

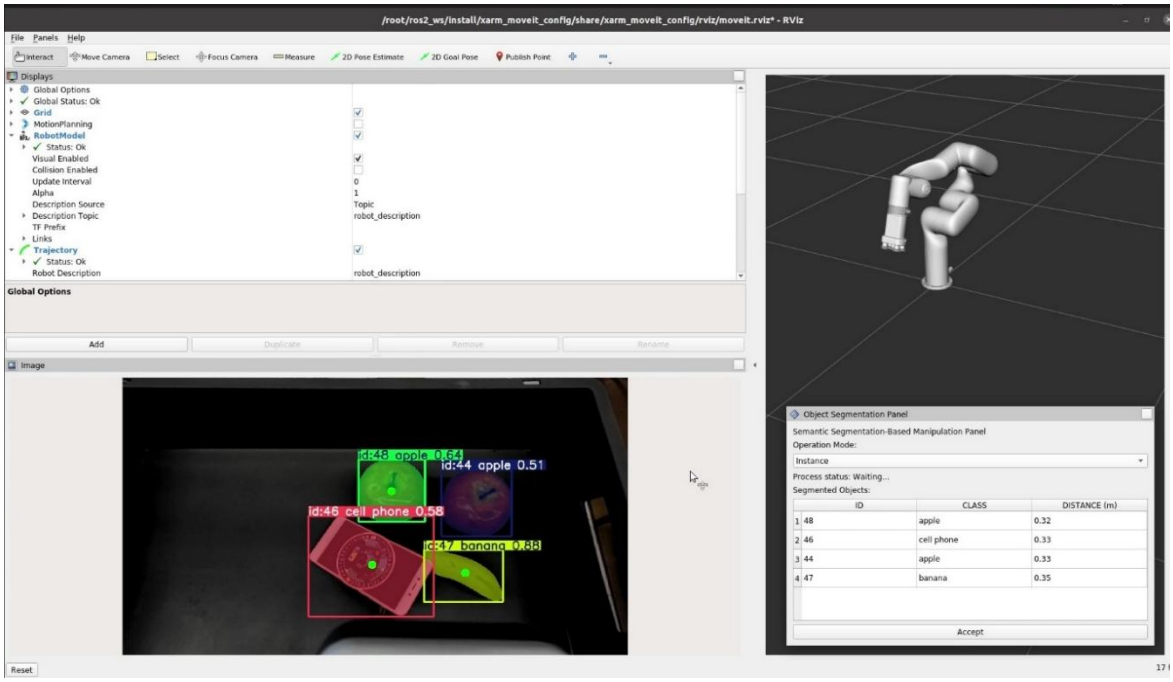


Figura 3.15. Interfaz gráfica de visualización RViz y el panel de control

El panel ha sido implementado utilizando Qt [28], un *framework* ampliamente utilizado para el desarrollo de interfaces gráficas en aplicaciones basadas en C++. Se constituye como un nodo más del paquete creado para el proyecto `segmentation_panel`, llamado `segmentation_control_panel`.

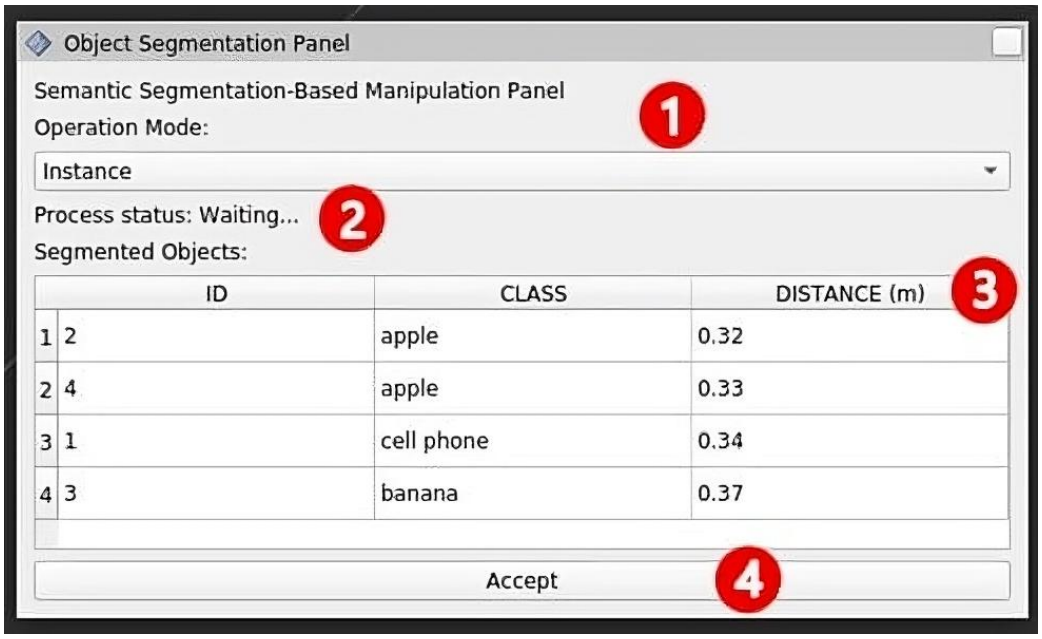


Figura 3.16. Interfaz gráfica de control

A continuación, se aclaran los elementos que componen el panel y su funcionamiento (Figura 3.16):

1) Desplegable para seleccionar modo de operación

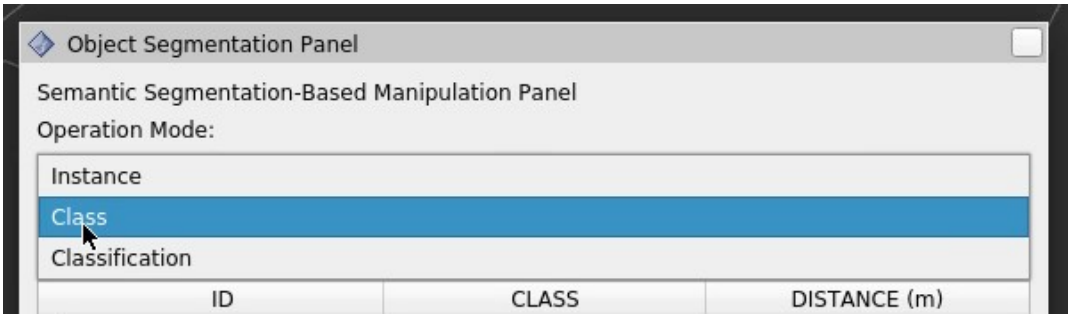


Figura 3.17. Desplegable de selección de modo de operación del sistema

El sistema posee tres modos de operación (Figura 3.17):

- **Selección por instancia**: el usuario debe elegir un objeto concreto de la tabla y el sistema procesará únicamente ese objeto especificado.
- **Selección por clase**: el sistema procesará todos los objetos presentes en el espacio de trabajo que pertenezcan a la misma clase que el objeto de la tabla seleccionado por el usuario.
- **Clasificación**: este modo se ha desarrollado para poder realizar el experimento que muestra la funcionalidad completa del sistema y los resultados obtenidos detallados en la sección cuatro de esta memoria.

2) Indicador del estado del proceso

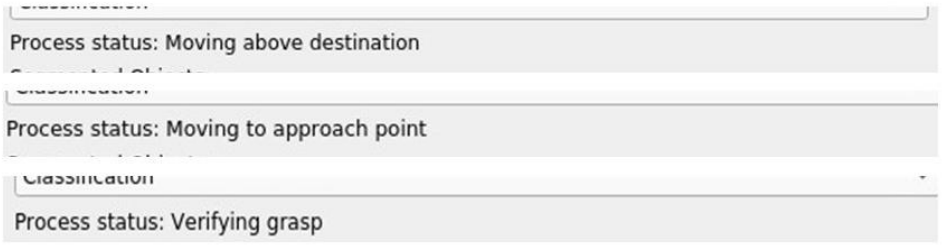


Figura 3.18. Ejemplos del indicador de estado del proceso en ejecución

Este indicador muestra un texto que indica la etapa del proceso de manipulación en el que se encuentra el sistema (Figura 3.18). Se rige por la etapas especificadas en la sección 3.2.

3) Tabla de objetos segmentados

Cada fila se corresponde con un objeto segmentado e incluye su identificador de segmentación, la clase y la distancia a la cámara. Los registros se encuentran ordenados de menor a mayor distancia a la cámara.

4) Botón de aceptar

Da lugar al inicio de la acción especificada por el usuario previamente.

3.4 Intercomunicación de los módulos en el ecosistema de ROS 2

En esta sección se muestran únicamente los canales de comunicación diseñados en el marco de este trabajo (Figura 3.19). Las dependencias externas, como los servicios nativos del controlador del xArm6, ya se han descrito en el apartado anterior 3.2 correspondiente al nodo de control del robot.

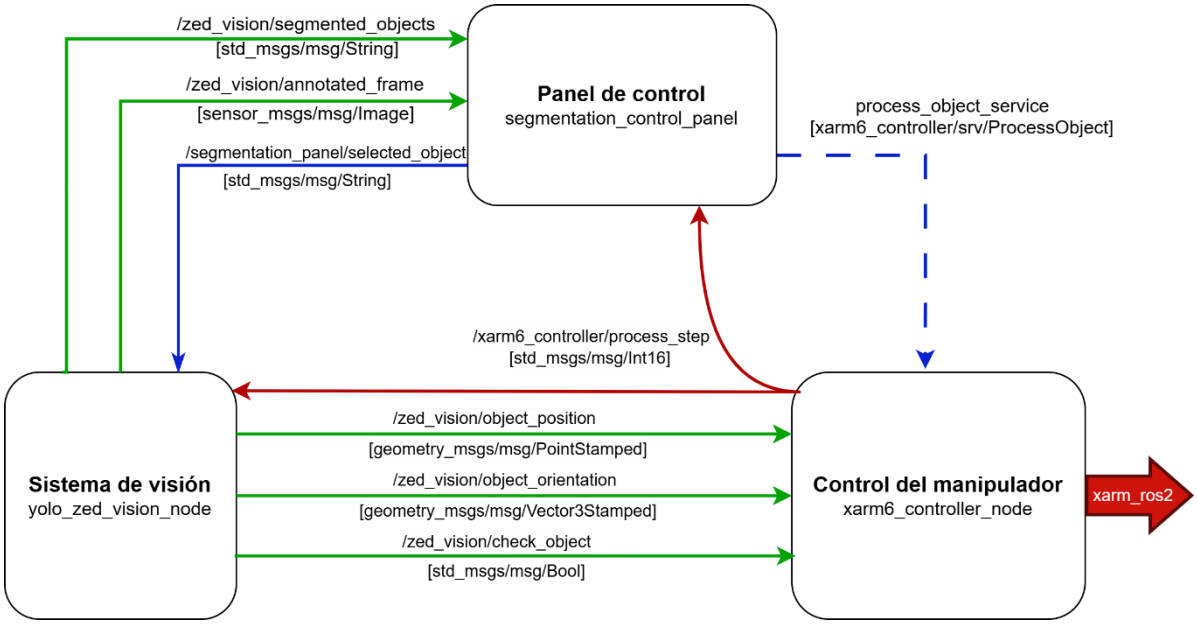


Figura 3.19. Diagrama de nodos y comunicaciones ROS desarrolladas en el proyecto

Con el fin de asegurar una comunicación estable entre nodos y evitar pérdidas de información, se ha utilizado una política de calidad de servicio (QoS) que permite a los nodos recibir los mensajes más recientes incluso si no estaban activos en el momento de su publicación.

Se procede a describir el funcionamiento interno del sistema durante un ciclo de operación normal. Para ello, se tomará como referencia el diagrama de secuencia de la Figura 3.20 que muestra las interacciones entre los distintos nodos del sistema. El diagrama completo puede dividirse en siete bloques que facilitan su comprensión y que serán desarrollados a continuación.

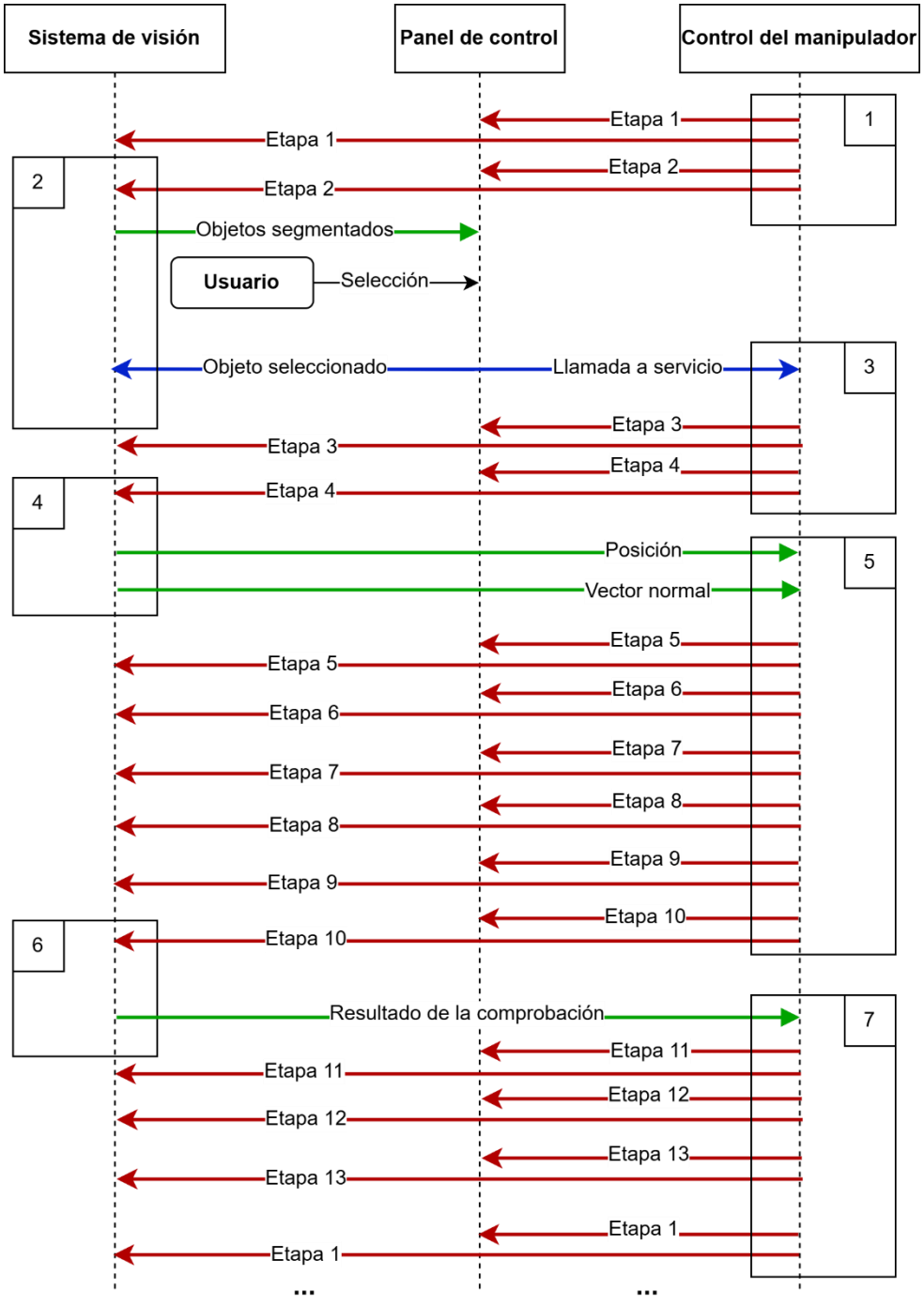


Figura 3.20. Diagrama de flujo del sistema en un ciclo del proceso de manipulación

El nodo de control del manipulador publica la etapa del proceso de manipulación (desarrolladas en la sección 3.2) en la que se encuentra el sistema a través del tópico `/xarm6_controller/process_step` cada vez que hay una transición de etapa. Por claridad, no se ha incluido en el diagrama la publicación continua de la imagen anotada por parte del sistema de visión, enviada mediante el tópico

`/zed_vision/annotated_frame` para su visualización en el panel. Esta funcionalidad se implementa con el soporte del paquete vision_opencv, que permite convertir imágenes en formato OpenCV a mensajes compatibles con ROS 2 para su publicación.

1) **Inicialización del sistema**

   El sistema se inicializa. El manipulador se posiciona en la ubicación de inicio definida.

2) **Espera de petición de usuario**

   El sistema de visión genera la lista de objetos identificados y la publica a través del tópico `/zed_vision/segmented_objects` en formato JSON para que pueda mostrarse en la tabla del panel. Cuando el usuario realiza una petición, el panel y llama al servicio `process_object_service` que pone en marcha el proceso de manipulación y también publica el objeto seleccionado mediante el tópico `/segmentation_panel/selected_object` que es recibido por el sistema de visión.

3) **Puesta en marcha**

   El sistema de control del manipulador recibe la petición por medio del servicio, que incluye el identificador de seguimiento, la clase, y el punto de aproximación respecto al sistema de referencia de la cámara. En la etapa tres, el manipulador se sitúa en dicho punto de aproximación para mejorar la visibilidad del objeto por parte de la cámara, con el propósito de aumentar la precisión del posterior cálculo del punto de agarre y orientación, que se activa en la transición de la cuarta etapa.

4) **Cálculo del punto de agarre y vector normal**

   Una vez el sistema se encuentra en la etapa cuatro, el sistema de visión calcula el punto de agarre y el vector normal procesando únicamente la segmentación del objeto seleccionado, ya que dispone de su ID para distinguirlo de los demás objetos de la escena. Los resultados son

publicados en los tópicos `/zed_vision/object_position` y `/zed_vision/object_orientation` respectivamente.

5) **Coger objeto**

Tras recibir la posición y vector normal, se expresan respecto al sistema de referencia de la base del robot y se obtiene la orientación a partir del vector normal. Con esta información, el manipulador ejecuta la acción de agarre. Esta maniobra abarca desde la etapa cinco hasta la nueve. Esta última etapa incluye posicionarse nuevamente en el punto de aproximación tras haber cogido el objeto, para que la cámara pueda visualizar el punto donde se avistó el objeto antes del agarre.

6) **Verificación del agarre**

La transición a la etapa diez indica al nodo de visión que tiene que analizar la escena para comprobar si el objeto seleccionado está presente. Publica la respuesta a través del tópico `/zed_vision/check_object`.

7) **Llevar objeto a destino y preparación para siguiente ciclo de operación**

Si el agarre se ha verificado correctamente, el objeto se traslada al punto de destino. Esta fase abarca las etapas de la once a la trece e incluye la preparación para un nuevo ciclo de operación. Si el agarre ha fallado y aún no se ha superado el número máximo de reintentos, el sistema vuelve a la etapa cuatro.

Cabe señalar que, si el modo de operación seleccionado es por clase o es clasificación, el sistema no espera una nueva entrada del usuario, sino que consulta automáticamente la lista para seleccionar el próximo objeto a procesar y cumplir con el objetivo.

4 Experimentos y resultados

Con el fin de verificar el funcionamiento del sistema desarrollado, se ha llevado a cabo un experimento en un entorno controlado que simula una situación de clasificación automática de objetos. El objetivo es evaluar la capacidad del sistema para identificar y segmentar los objetos, así como manipularlos y depositarlos en la zona correspondiente según su clase.

4.1 Materiales y método

La Figura 4.1 muestra el entorno de pruebas montado para este propósito, compuesto por el material base del proyecto (manipulador xArm6, la cámara de ZED Mini y NVIDIA Jetson como computador) y las áreas de depósito designadas para cada categoría.

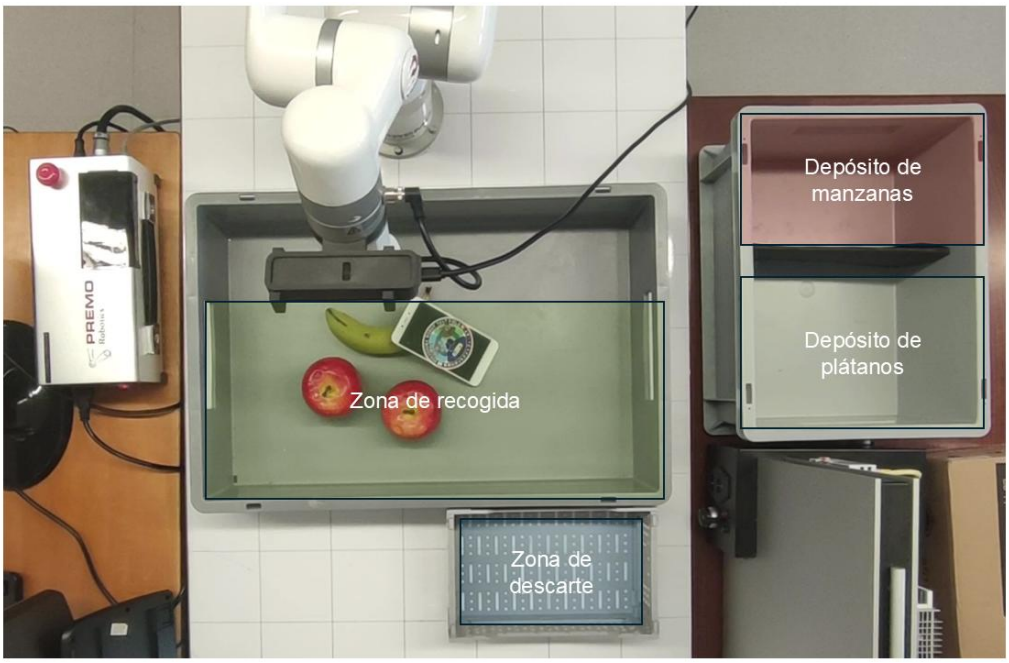


Figura 4.1. Zona de trabajo del experimento

La red neuronal empleada está preentrenada con el conjunto de datos COCO (*Common Objects in Context*) [29] que incluye una amplia selección de clases de objetos. Para las pruebas, se han seleccionado tres objetos con geometrías variadas, con la finalidad de extraer mayores conclusiones sobre el comportamiento

del sistema: manzana como objeto no convexo, centrado en su caja delimitadora pero con superficie curva; plátano, geometría cóncava ideal para testear que el punto de agarre seleccionado se encuentra dentro del contorno del objeto y teléfono móvil, objeto plano que, dependiendo de su disposición, puede presentar una ligera pendiente, ofreciendo una superficie distinta a las anteriores. Se definieron tres zonas de depósito en función de la clase del objeto: una para plátanos, otra para manzanas, y una tercera denominada zona de descarte, destinada a aquellos objetos que no pertenecen a las clases anteriores (como el teléfono móvil, en este caso).

Como se ha mencionado anteriormente en la presente memoria, el sistema ofrece tres modos de operación: selección de una instancia, selección por clase y la clasificación automática. Todos los modos de operación comparten la misma lógica de procesamiento y comunicación entre nodos. Se diferencian únicamente en la estrategia para seleccionar el siguiente objeto a procesar. En este experimento se ha empleado el modo de clasificación automática, ya que permite evaluar de forma integral el sistema: para que este modo funcione correctamente, es necesario que lo hagan también las estructuras lógicas y de comunicación utilizadas en los otros dos modos. Por tanto, esta modalidad es la prueba más completa del sistema desarrollado.

En esta prueba se evaluaron los siguientes aspectos:

- Correcta comunicación entre nodos del sistema: visualización en tiempo real coherente con lo que sucede en el mundo real.
- Ejecución secuencial de las distintas etapas del proceso de manipulación.
- Precisión del cálculo del punto y orientación de agarre ante diversos tipos de geometrías: curva, cóncava y plana.
- Cumplimiento de la petición del usuario realizada desde el panel de control.

4.2 Desarrollo del experimento

Una vez puesto en marcha el sistema y completada la etapa de inicialización, el manipulador se situó en la posición de inicio, quedando preparado para recibir una petición a través del panel de control. Simultáneamente, se dispuso de la visualización en tiempo real del entorno. El sistema de visión identificó los objetos presentes en la zona de recogida, los clasificó y se mostraron en la tabla del panel, ordenados por proximidad a la cámara. En este punto, se seleccionó el modo de clasificación automática, tal como se muestra en la Figura 4.2.

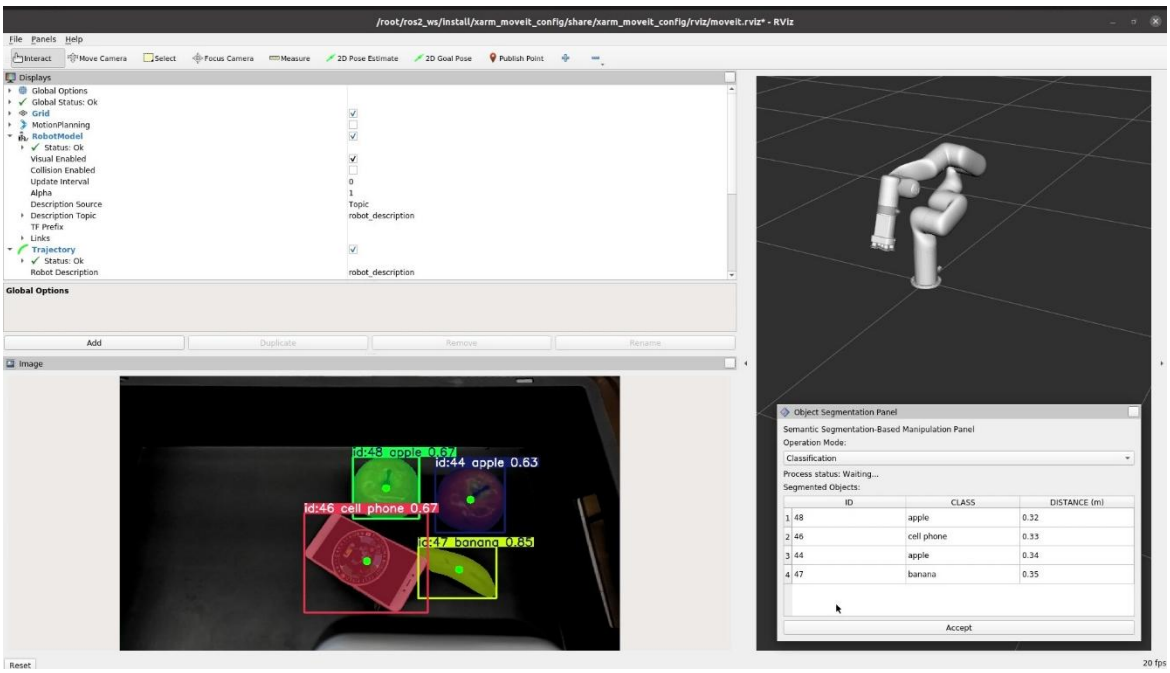


Figura 4.2. Etapa inicial de identificación de los objetos en el espacio de trabajo

Tras pulsar el botón de confirmación, comenzó el proceso de manipulación. El primer objeto procesado fue una manzana, cuyo ID correspondía a la instancia más cercana a la cámara. El manipulador se posicionó en el punto de aproximación situado encima del objeto con un ligero desfase para garantizar que el objeto estuviera centrado en el plano imagen y así facilitara el cálculo preciso del punto de agarre y vector normal (Figura 4.3).

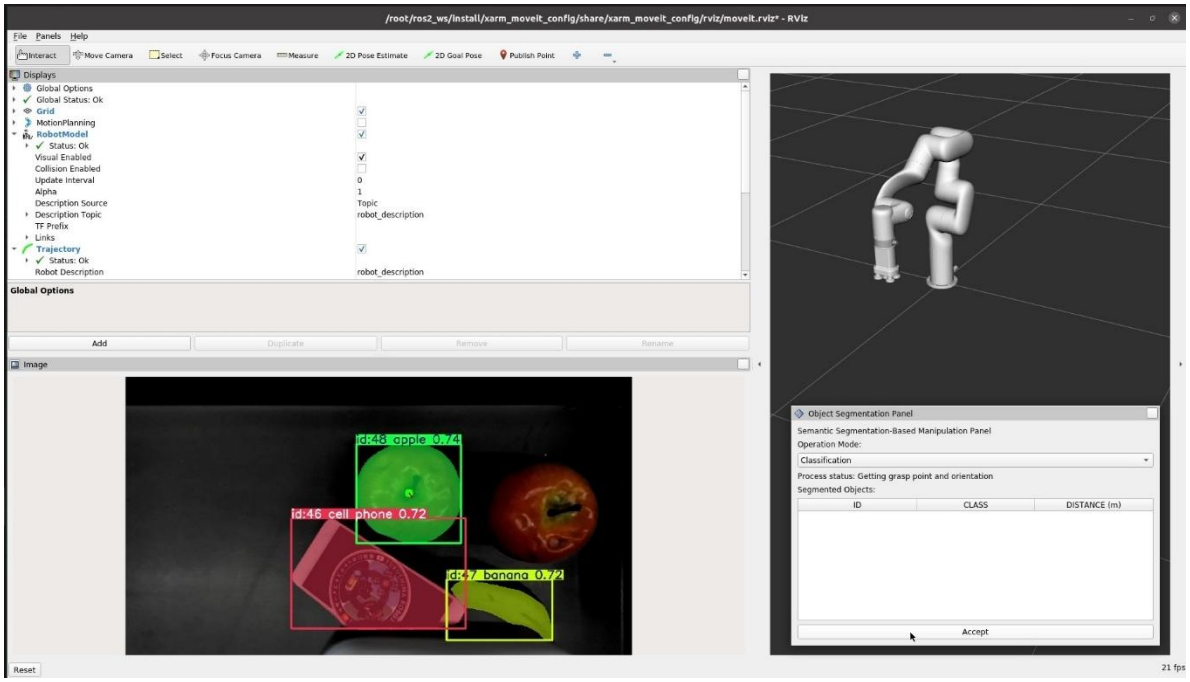


Figura 4.3. Cálculo de punto de agarre (punto verde) y orientación (vector rojo) de agarre desde punto de aproximación del objeto

A continuación, se ejecutó la maniobra de agarre mostrada en la Figura 4.4, de izquierda a derecha: el manipulador se posicionó sobre el punto de agarre sin orientación, después se orientó para alinearse con el vector normal, descendió para entrar en contacto perpendicularmente con la superficie del objeto y finalmente elevó el objeto hacia una posición de seguridad.

Figura 4.4. Maniobra de agarre para objeto de clase manzana

Acto seguido, el manipulador volvió al punto de aproximación para que el sistema de visión verificara si el objeto seguía presente. En la Figura 4.5 se observa que el sistema comunicó que el objeto no fue detectado, indicando así que el agarre fue

exitoso. El manipulador procedió entonces a transportar el objeto a su zona destino asignada según su clase (Figura 4.6).

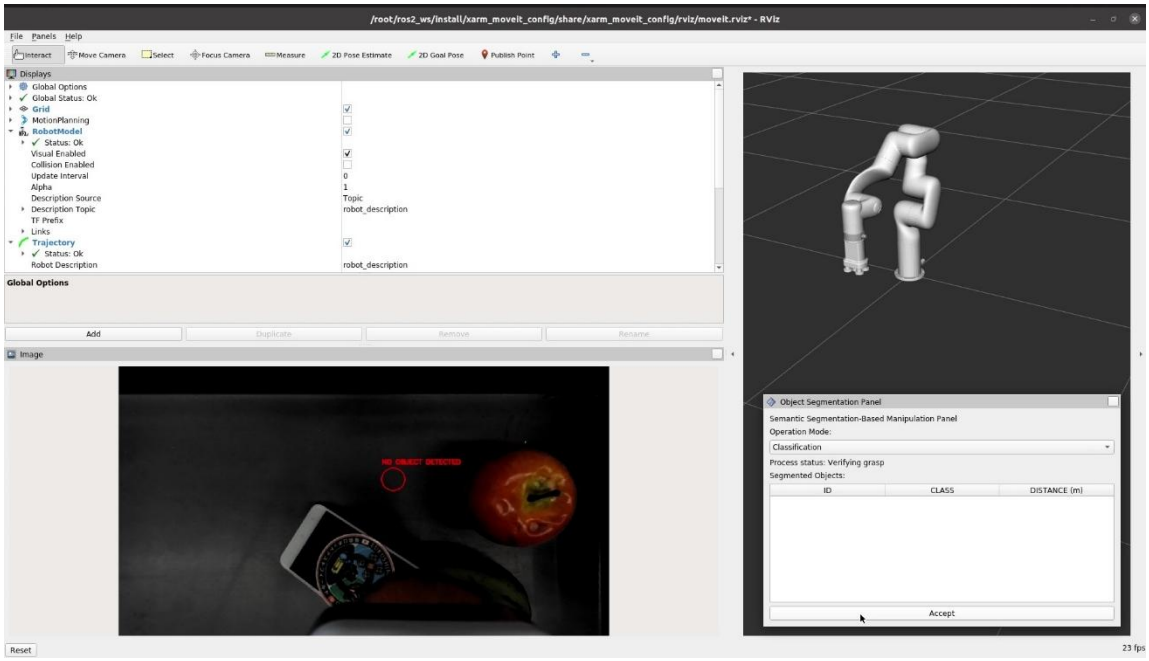


Figura 4.5. Comprobación de agarre mediante detección del objeto seleccionado

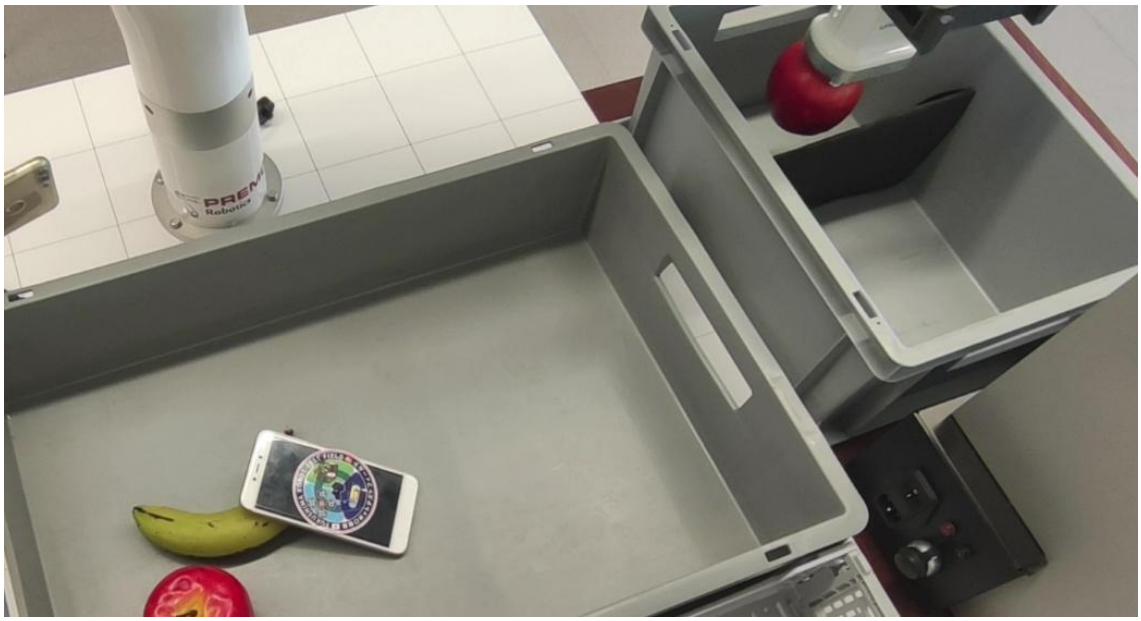


Figura 4.6. Colocación del objeto en la zona correspondiente a la clase manzana

Al finalizar el proceso del primer objeto, el sistema continuó automáticamente con el siguiente elemento de la lista. En este caso, el teléfono móvil. En la Figura 4.7, se detalla la maniobra de agarre, lo que implica que ya se había calculado su punto de agarre y orientación mediante el mismo procedimiento ya descrito.

Figura 4.7. Maniobra de agarre para objeto de clase móvil

El siguiente objeto fue la otra manzana. El resultado de la maniobra de manipulación se muestra en la Figura 4.8.

Figura 4.8. Maniobra de agarre para el otro objeto de clase manzana

Finalmente, el último objeto a procesar es el plátano cuya maniobra de agarre puede observarse en la Figura 4.9.

Figura 4.9. Maniobra de agarre para el otro objeto de clase plátano

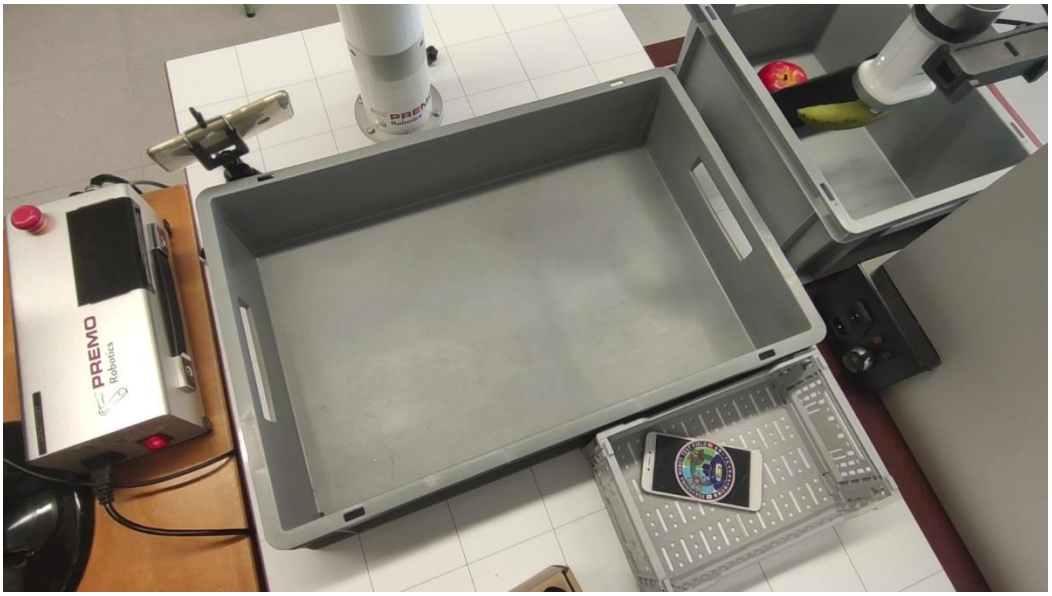


Figura 4.10. Colocación del objeto en la zona correspondiente a la clase plátano

Una vez clasificados todos los objetos (Figura 4.10), el sistema finalizó el ciclo y quedó a la espera de una nueva instrucción por parte del usuario. Con ello, el experimento concluyó.

4.3 Análisis de los resultados

El experimento realizado permitió evaluar el comportamiento del sistema en condiciones controladas y con diferentes tipos de objetos. A continuación, se presentan los resultados obtenidos en función de los aspectos evaluados.

- Durante todo el experimento, se observó una correcta comunicación entre los distintos nodos que conforman el sistema. La visualización ofrecida en el panel de control reflejó en todo momento el estado del sistema en tiempo real, incluyendo la lista de objetos identificados, la etapa del proceso de manipulación en curso y las imágenes procesadas por el sistema de visión. Esto validó el uso de la política de calidad de servicio seleccionada, que permitió mantener una comunicación fluida incluso ante múltiples publicaciones simultáneas.
- El sistema ejecutó de forma secuencial y coherente todas las etapas del ciclo de operación definido. Desde la inicialización hasta la verificación del agarre y el transporte del objeto a su destino correspondiente, todas las transiciones se realizaron de acuerdo con lo descrito en el diseño del sistema. Además, respondió correctamente a la configuración inicial seleccionada por el usuario mediante el panel: el modo de clasificación automática fue ejecutado sin errores lo que permitió verificar la lógica de repetición del ciclo sin necesidad de intervención del usuario.
- Los resultados obtenidos para los distintos tipos de geometrías confirman una buena precisión en el cálculo del punto de agarre y la orientación del efector final y la versatilidad del algoritmo ante geometrías superficiales diversas.

5 Conclusiones

A lo largo de este trabajo se han cumplido los objetivos planteados en el apartado inicial. En primer lugar, se ha logrado desarrollar un sistema capaz de detectar y segmentar objetos en un entorno desordenado mediante una cámara estéreo RGB-D, integrando dicha información para calcular de forma automática un punto de agarre y vector normal que permite al robot manipulador realizar con éxito la acción de recogida. Además, el sistema ha demostrado adaptabilidad a distintos tipos de geometría y clases de objetos, confirmando su viabilidad para entornos variables.

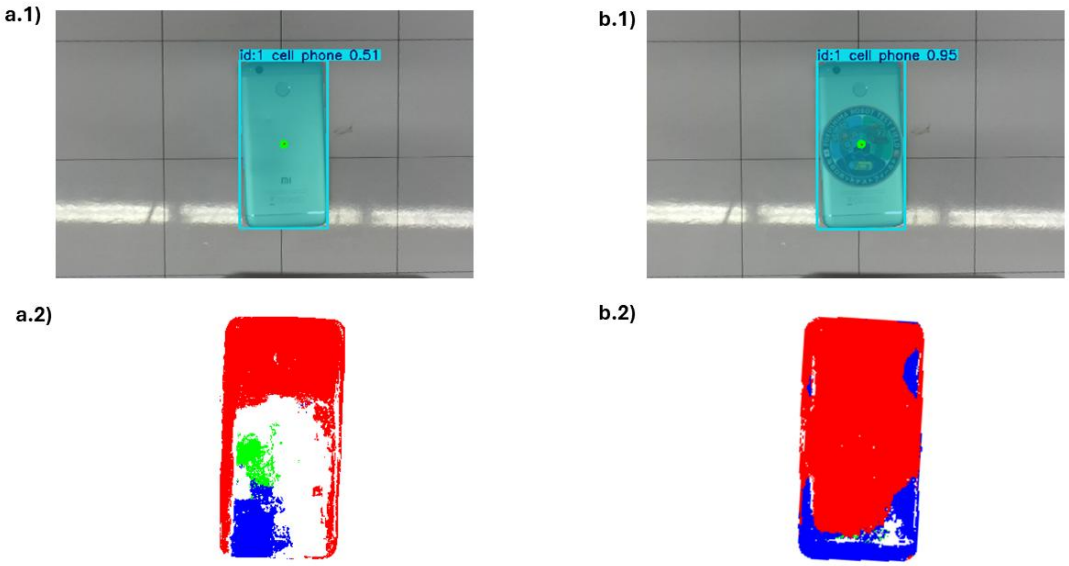
Como objetivo secundario, se ha implementado un sistema de monitorización en tiempo real, que permite visualizar tanto la segmentación como el estado actual del manipulador. El panel de control desarrollado facilita la interacción con el usuario, permitiendo seleccionar objetos, definir modos de operación y seguir el proceso en cada una de sus etapas. Asimismo, el sistema incorpora un mecanismo de verificación posterior al agarre, que actúa como tolerancia a errores y permite reiniciar la acción en caso de fallo, cumpliendo con el objetivo de robustez frente a errores de manipulación.

A partir de los resultados experimentales y del desarrollo del sistema, pueden extraerse algunas observaciones adicionales relevantes. Por ejemplo, el método utilizado para calcular el punto de agarre se adapta correctamente en situaciones con solape parcial de objetos, ya que opera sobre la porción visible y segmentada. Sin embargo, esto no garantiza que el punto elegido sea el más estable para el agarre. Además, se ha observado que cuando el robot alcanza un estado de error (por ejemplo, debido a una interrupción inesperada), el sistema es capaz de reiniciarse de forma automática, volviendo a la etapa inicial, lo que mejora la autonomía del ciclo operativo.

Durante la implementación también se detectaron limitaciones importantes derivadas del uso de visión estereoscópica. Tal como se describe en la sección 2.3, este sistema se basa en la comparación entre las imágenes captadas por dos cámaras físicas separadas, que observan la escena desde perspectivas ligeramente distintas. En las pruebas con el mapa de profundidad, se observa cierta

inestabilidad en la estimación de la distancia de puntos pertenecientes a superficies homogéneas y poco texturizadas.

Esta limitación puede observarse comparando el mapa de profundidad de una superficie homogénea con el de esa misma superficie tras añadir un elemento que introduzca variabilidad y textura (Figura 5.1). La presencia de características visuales adicionales facilita al algoritmo estéreo la identificación de correspondencias entre las imágenes captadas por ambas cámaras, lo que se traduce en una estimación de profundidad más estable y precisa.



a.1) Móvil segmentado
a.2) Píxeles agrupados del mapa de profundidad del móvil
b.1) Móvil con pegatina segmentado
b.2) Píxeles agrupados del mapa de profundidad del móvil con pegatina

Figura 5.1. Problema de cálculo de profundidad para superficies homogéneas.

Del mismo modo, se ha identificado que un exceso de brillo en las escenas puede afectar negativamente a la identificación. Como se describe en la sección 3.1, actualmente se aplica una atenuación global del brillo mediante la reducción del canal V en el espacio de color HSV, lo que permite mitigar parcialmente la sobreexposición general en la imagen.

No obstante, esta solución global no distingue entre zonas correctamente iluminadas y zonas sobreexpuestas, lo que puede degradar detalles relevantes para la identificación. Como mejora futura, se plantea aplicar técnicas de realce local, que preserven texturas significativas en regiones de interés sin afectar el

contraste en áreas ya adecuadas. En particular, podrían explorarse métodos como CLAHE (*Contrast Limited Adaptive Histogram Equalization*), una forma adaptativa de ecualización de histograma que actúa por regiones, o técnicas de normalización fotométrica para compensar variaciones de iluminación no uniformes.

En resumen, el sistema desarrollado cumple con los objetivos iniciales, mostrando un comportamiento robusto y versátil. No obstante, la experiencia obtenida en su desarrollo permite identificar posibles mejoras tanto a nivel de percepción como de planificación de movimientos.

5.1 Futuras mejoras

A partir de la experiencia obtenida durante la implementación y prueba del sistema desarrollado, se han identificado diversas líneas de mejora que podrían mejorar sus características:

- **Mejora del sistema de visión**: Una posible mejora sería sustituir la cámara estéreo RGB-D por una cámara basada en infrarrojos, como Intel RealSense, que ofrece mayor estabilidad ante superficies homogéneas o escenas con iluminación compleja. También se propone considerar configuraciones en las que el sistema de visión no se sitúe en el extremo del manipulador, lo que permitiría mantener una vista constante de la escena durante la manipulación que facilite el seguimiento de los objetos.
- **Planificación y control más avanzados**: Aunque MoveIt2 no se ha utilizado de forma directa en este trabajo, su integración futura permitiría implementar algoritmos de planificación de trayectorias más complejos, adaptados a geometrías irregulares o espacios reducidos. Asimismo, podría explorarse el uso de control visual basado en imagen en lugar de operar únicamente con consignas de posición en el espacio operacional.
- **Adaptación a distintas herramientas de agarre**: El sistema podría extenderse fácilmente para operar con otros tipos de efectores finales, como pinzas paralelas, en función del tipo de objeto a manipular.
- **Entrenamiento personalizado de la red neuronal**: Para maximizar la eficacia en entornos industriales concretos, podría entrenarse la red de

segmentación con un conjunto de datos específico de los objetos de interés, mejorando la precisión en la detección y clasificación.

- **Mejoras en la verificación del agarre**: Aunque el mecanismo actual aporta robustez, podría mejorarse mediante técnicas de visión que permitan mantener la identificación del objeto durante todo el proceso de manipulación. Por ejemplo, se podrían usar métodos que comparen características visuales más específicas del objeto, en lugar de basarse únicamente en la clase, para verificar de forma más precisa si el objeto sigue presente en la zona de agarre. Estas técnicas podrían basarse en descriptores visuales o análisis de vectores de características que permitan diferenciar objetos similares y confirmar la presencia del objeto deseado.

Estas líneas abren la posibilidad de evolucionar el sistema hacia un entorno más autónomo, versátil y robusto, capaz de adaptarse a distintas condiciones industriales.

6 Referencias bibliográficas

[1] P. Corke, Robotics, Vision and Control: Fundamental Algorithms in Python, 3.ª ed. Cham, Switzerland: Springer, 2023.

[2] D. Martín Godoy, «MANIPULATOR-ROBOT-CONTROL-THROUGH-VISION», GitHub. Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://github.com/Deglox/MANIPULATOR-ROBOT-CONTROL-THROUGH-VISION/tree/main

[3] S. Hutchinson, G. D. Hager, y P. I. Corke, «A tutorial on visual servo control», IEEE Trans. Robot. Autom., vol. 12, n.º 5, pp. 651-670, oct. 1996, doi: 10.1109/70.538972.

[4] Shirai, Y. y Inoue, H., «Guiding a Robot by Visual Feedback in Assembling Tasks», Pattern Recognition, vol. 5, pp. 99-108, 1973.

[5] Burlacu, Adrian y Lazăr, Corneliu, «Image Based Controller for Visual Servoing Systems», Buletinul Institutului Politehnic din Iași, vol. LIV, 2008.

[6] Hill, J. y Park, W. T., «Real time control of a robot with a mobile camera», presentado en 9th International Symposium on Industrial Robots (ISIR), Washington, D.C., mar. 1979, pp. 233-246.

[7] Makhlin, A. G., «Stability and sensitivity of servo vision systems», presentado en 5th International Conference on Robot Vision and Sensory Controls – RoViSeC 5, Amsterdam, 1985, pp. 79-89.

[8] Weiss, L. E., «Dynamic visual servo control of robots: An adaptive image-based approach», Ph.D. thesis, Carnegie Mellon University, 1984.

[9] Kelly, R., «Robust asymptotically stable visual servoing of planar robots», vol. 12, n.º 5, pp. 759-766, 1996.

[10] A.-P. Botezatu y A. Burlacu, «A Short Review of Deep Learning Methods in Visual Servoing Systems», Bull. Polytech. Inst. Iași Electr. Eng. Power Eng. Electron. Sect., vol. 69, n.º 3, pp. 113-136, sep. 2023, doi: 10.2478/bipie-2023-0018.

[11] G. Hou, H. Chen, M. Jiang, y R. Niu, «An Overview of the Application of Machine Vision in Recognition and Localization of Fruit and Vegetable Harvesting Robots», Agriculture, vol. 13, n.º 9, p. 1814, sep. 2023, doi: 10.3390/agriculture13091814.

[12] Q. Guo et al., «Progress, challenges and trends on vision sensing technologies in automatic/intelligent robotic welding: State-of-the-art review»,

Robot. Comput.-Integr. Manuf., vol. 89, p. 102767, oct. 2024, doi: 10.1016/j.rcim.2024.102767.

[13] «ZED Mini Stereo Camera», Stereolabs. Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://www.stereolabs.com/en-es/store/products/zed-mini

[14] Stereolabs, «ZED Mini Datasheet v1.1». [En línea]. Disponible en: https://cdn.sanity.io/files/s18ewfw4/staging/ebcd46896092d1ee6212b7f4d81aaa1c479c2440.pdf/ZED%20Mini%20Datasheet%20v1.1.pdf

[15] Stereolabs, «ZED SDK - Python API», Stereolabs. [En línea]. Disponible en: https://www.stereolabs.com/docs/api/python

[16] UFactory, «xArm 6 – Robot Arm», UFactory. Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://www.ufactory.us/product/ufactory-xarm-6

[17] UFactory, «xArm 6 User Manual». [En línea]. Disponible en: https://www.ufactory.cc/wp-content/uploads/2023/05/xArm-User-Manual-V2.0.0.pdf

[18] UFactory, «xArm ROS2». Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://github.com/xArm-Developer/xarm_ros2

[19] Zach Wendt, «NVIDIA Jetson Xavier: una NPI que acelera la inteligencia artificial en el borde», Arrow Electronics. Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://www.arrow.com/es-mx/research-and-events/articles/nvidia-jetson-xavier-npi

[20] NVIDIA, «L4T JetPack Container Image», NVIDIA NGC. [En línea]. Disponible en: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-jetpack

[21] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», en 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA: IEEE, jun. 2016, pp. 779-788. doi: 10.1109/CVPR.2016.91.

[22] Ultralytics, «YOLO11 Overview», Ultralytics Docs. Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://docs.ultralytics.com/models/yolo11/#overview

[23] MoveIt Maintainers, «moveit2 (foxy branch)». Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://github.com/ros-planning/moveit2/tree/foxy

[24] ROS Perception Project, «vision_opencv (foxy branch)». Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://github.com/ros-perception/vision_opencv/tree/foxy

[25] ros-drivers, «nmea_msgs». Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://github.com/ros-drivers/nmea_msgs

[26] OpenCV, «Camera Calibration and 3D Reconstruction — OpenCV documentation», OpenCV. Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[27] Covenant Turtle, Basic Intro to Quaternions for 3D Rotations, (4 de mayo de 2022). [YouTube]. Disponible en: https://youtu.be/xoTqBqQtrY4

[28] Qt Company, «Qt – Cross-platform software development». Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://www.qt.io/

[29] COCO Dataset, «COCO - Common Objects in Context». Accedido: 15 de junio de 2025. [En línea]. Disponible en: https://cocodataset.org/#home

Apéndice A: Código fuente y material audiovisual

En este apéndice se incluyen los enlaces al repositorio en GitHub, que contiene todo el código fuente desarrollado para este trabajo, y al vídeo demostrativo del experimento analizado en el capítulo cuarto.

- **Repositorio GitHub:** Contiene los tres paquetes desarrollados en ROS 2, junto con documentación técnica y las instrucciones para la instalación del entorno de desarrollo.

  https://github.com/PedroMM03/semantic-segmentation-adaptive-manipulation.git

- **Vídeo demostrativo:** https://youtu.be/WOYQpVIyCkI