DESARROLLO DE UNA HERRAMIENTA SOFTWARE DE SUPERVISIÓN Y TELE-COMANDO DE UN ROBOT MANIPULADOR SIMULADO PARA UNA APLICACIÓN DE ALMACENAMIENTO, TRANSPORTE Y CARGA/DESCARGA DE BIENES EN UN AMBIENTE DE APRENDIZAJE PARA LA INDUSTRIA 4.0

LINA MARCELA SALCEDO MONTENEGRO

CÓDIGO: 202025519

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2026

DESARROLLO DE UNA HERRAMIENTA SOFTWARE DE SUPERVISIÓN Y TELE-COMANDO DE UN ROBOT MANIPULADOR SIMULADO PARA UNA APLICACIÓN DE ALMACENAMIENTO, TRANSPORTE Y CARGA/DESCARGA DE BIENES EN UN AMBIENTE DE APRENDIZAJE PARA LA INDUSTRIA 4.0

LINA MARCELA SALCEDO MONTENEGRO

Código:202025519

Documento presentado al programa de: Ingeniería Electrónica (3744) como requisito para optar por el título de: Ingeniera Electrónica

Director
Eval Bladimir Bacca Cortes, Ph.D.
Área informática industrial

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2026

Nota de aceptación:

_____
_____
_____
_____
_____
_____
_____

_____
Firma del director del trabajo

_____
Firma del evaluador

_____
Firma del evaluador

Santiago de Cali, 09 de febrero del 2026.

RESUMEN

La Industria 4.0, como expresión de la cuarta revolución industrial, plantea nuevos retos para la formación de los futuros profesionales, quienes deben adquirir competencias en tecnologías emergentes como la robótica, la programación, la automatización y las industrias digitales. Ante esta necesidad, los entornos educativos han evolucionado hacia la creación de espacios físicos de aprendizaje 4.0, orientados a la experimentación y al desarrollo de habilidades tecnológicas mediante experiencias prácticas.

En este contexto, en el presente trabajo de grado se desarrolla una herramienta de software orientada a un ambiente de aprendizaje para la Industria 4.0. Dicha herramienta consiste en una aplicación integral que permite el almacenamiento, transporte y procesos de carga y descarga, facilitando la supervisión y el tele-comando de un robot manipulador simulado. El sistema está diseñado para ofrecer a los usuarios una interacción directa con tecnologías similares a entornos industriales modernos, favoreciendo el aprendizaje práctico y significativo.

Los resultados obtenidos evidenciaron que la herramienta desarrollada contribuye al fortalecimiento de los conocimientos técnicos y habilidades tecnológicas de los estudiantes, apoyando los procesos de enseñanza-aprendizaje en escenarios alineados con las exigencias actuales de la Industria 4.0.

Palabras clave: ROS, laboratorios virtuales, robots manipuladores, industrias 4.0, robots para el aprendizaje.

ABSTRACT

Industry 4.0, as an expression of the fourth industrial revolution, poses new challenges for the training of future professionals, who must acquire competencies in emerging technologies such as robotics, programming, automation, and digital industries. In response to this need, educational environments have evolved toward the creation of 4.0 learning spaces, focused on experimentation and the development of technological skills through practical experiences.

In this context, the present thesis develops a software tool designed for a learning environment aligned with Industry 4.0. This tool consists of an integrated application that enables storage, transportation, and loading/unloading processes, facilitating the supervision and remote control of a simulated robotic manipulator. The system is designed to provide users with direct interaction with technologies similar to those found in modern industrial environments, promoting practical and meaningful learning.

The results obtained demonstrated that the developed tool contributes to strengthening students' technical knowledge and technological skills, supporting teaching-learning processes in scenarios aligned with the current demands of Industry 4.0.

Keywords: ROS, virtual laboratories, robot manipulators, Industry 4.0, educational robots.

TABLA DE CONTENIDO

1. CAPÍTULO I: INTRODUCCIÓN 11
1.1. INTRODUCCIÓN 11
1.2. PLANTEAMIENTO DEL PROBLEMA 13
1.3. OBJETIVO 14
1.3.1. OBJETIVO GENERAL 14
1.3.2. OBJETIVOS ESPECÍFICOS 14
1.4. SOLUCIÓN PROPUESTA 15
1.5. ESTRUCTURA DEL DOCUMENTO 16
2. CAPÍTULO II: antecedentes y marco teórico 17
2.1. INTRODUCCIÓN 17
2.2. ANTECEDENTES 18
2.3. MARCO TEÓRICO 21
2.3.1. Robots Manipuladores 21
2.3.2. Cinemática de los robos manipuladores 22
2.3.3. ROS (Robot Operating System) 23
2.3.4. Simulación robótica con Gazebo 24
2.3.5. Industrias 4.0 para robótica de manipuladores 25
2.3.6. Tareas de almacenamiento, transporte, carga y descarga 25
2.3.7. Técnicas de reconocimiento por forma 26
2.3.8. Técnicas de reconocimiento por color 26
2.3.9. RUP, Rational Unified Process 27
2.4. OBSERVACIONES FINALES 27
3. CAPÍTULO III: DESARROLLO DEL APLICATIVO EDU-ARM 29
3.1. Introducción 30
3.2. Diseño del aplicativo EDU-ARM 30
3.2.1. Requerimientos funcionales 30
3.2.2. Requerimientos no funcionales 31
3.2.3. Diagrama de concepto 32
3.2.4. Diagrama de nodos. 35
3.3. IMPLEMENTACIÓN DEL APLICATIVO EDU-ARM 35
3.3.1. Configuración y puesta en marcha del robot OpenManipulator-x 35
3.3.2. Interfaz EDU-ARM 42
3.4. IMPLEMENTACIÓN DE LA RUTA DE APRENDIZAJE 48
3.4.1. Entorno de trabajo 48

3.4.2. Ruta de Aprendizaje 50
3.4.3. Crazy Gripper 51
3.4.4. Arm-Eye 53
3.4.5. Arm-Commander 55
3.4.6. Paquetes verificadores 58
3.4.7. Proyecto Pickup-Máster 64
3.5. CONCLUSIONES 69
4. PRUEBAS Y RESULTADOS 70
4.1. Introducción 70
4.2. PRUEBAS DE INTEGRACIÓN 70
4.3. PRUEBAS DE CAMPO 81
4.4. CONCLUSIONES 84
5. CONCLUSIONES Y TRABAJOS FUTUROS 85
5.1. Conclusiones generales 85
5.2. Trabajos Futuros 86
6. REFERENCIAS 87
7. ANEXOS 90

LISTA DE TABLAS

Tabla 2.1 Tabla comparativa características relevantes de los antecedentes. .18
Tabla 3.1 Requisitos funcionales 31
Tabla 3.2 Requerimientos no funcionales 31
Tabla 3.3 Elementos y características del robot real. 38
Tabla 3.4 tópicos del Controlador 42
Tabla 3.5 Servicios del controlador del OpenManipulator-X 42
Tabla 3.6 Especificaciones articulares del robot 44
Tabla 4.1 Pruebas de Integración del módulo configuración de la Interfaz EDU-ARM. 71
Tabla 4.2 Pruebas de Integración del módulo Percepción de la Interfaz Edu-ARM 75
Tabla 4.3 Pruebas de Integración del módulo de Calibración de la Interfaz Edu-Arm. 77
Tabla 4.4 Preguntas de la encuesta 81
Tabla 4.5 Resumen encuesta a estudiantes en pruebas de campo. 83

LISTA DE FIGURAS

Figura 1.1 a) Robot Universal robots en la educación. Fuente: Tomado de [1]. b) Robot OpenMANIPULATOR-X real. Fuente: Tomado de [2]. 11
Figura 1.2 Proyecto “Imagine make” donde usan los robots para la enseñanza. Fuente: Tomado de [4]. 12
Figura 1.3 Robot en un entorno de aprendizaje. Fuente: Tomado de [5]. 12
Figura 1.4 Robots que almacenan, distribuyen y recolectan materiales. Fuente: Tomado de [3]. 13
Figura 1.5 Solución propuesta. Fuente: Elaboración propia. 15
Figura 2.1 (a) UR5; (b) Franka Panda; (c) Tampere RoboLab (center); (d) KUKA Iiwa; (e) Sensor rack. Fuente: Tomado de [9]. 17
Figura 2.2 Célula robótica reconfigurable para el reciclaje automatizado de residuos electrónicos. Fuente: Tomado de [10]. 21
Figura 2.3 a) Un brazo planar de tres vínculos los ángulos y posición de las articulaciones. b) Manipulador planar de tres vínculos con los ejes de posición. Fuente: Tomado de [15]. 22
Figura 2.4 Funcionamiento servicios ROS. Fuente: Elaboración propia. 23
Figura 2.5 Funcionamiento acciones ROS. Fuente: Elaboración propia. 24
Figura 2.6 Modelo de funcionamiento de ROS. Fuente: Elaboración propia. ... 24
Figura 2.7 Entorno simulado en Gazebo. Fuente: Tomado de [17]. 24
Figura 2.8 Detector de bordes Canny aplicado a una imagen. Fuente: Tomado de [22]. 26
Figura 2.9 Mask en imagen para detección de objetos color amarillo. Fuente: Tomado de [24]. 27
Figura 2.10 Gráfico de las fases de RUP. Fuente: Tomado de [27]. 27
Figura 3.1 portable “Mini-Factory”. Fuente: Tomada de [12]. 29
Figura 3.2 Diagrama conceptual aplicativo. Fuente: Elaboración propia 33
Figura 3.3 Diagrama de Nodos. Fuente: Elaboración propia. 34
Figura 3.4 Modelo URDF del Open Manipulator X visualizado en GAZEBO. Fuente: Tomado de[28]. 36
Figura 3.5 Open Manipulator X en Gazebo. Fuente: Elaboración propia. 36
Figura 3.6 Estructura jerárquica de Joints y Links del Open Manipulator X. Fuente: Elaboración propia. 37
Figura 3.7 Ventana de gestión de Firmware en Wizard 2.0 para el motor XM430-W350. Fuente: Elaboración propia. 39
Figura 3.8 Parámetros de configuración de la cámara. Fuente: Elaboración propia. 40
Figura 3.9 Configuración de red del robot. Fuente: Elaboración propia. 40
Figura 3.10 OpenManipulator-X ensamblado. Fuente: Elaboración propia. 41
Figura 3.11 (a) Diseño 3D de la carcasa de la tarjeta OpenCR y la Raspberry Pi. Fuente: Elaboración propia. 41
Figura 3.12 Módulo de Configuración. Fuente: Elaboración propia. 43
Figura 3.13 Movimiento Cartesiano. Fuente: Elaboración propia. 44
Figura 3.14 pestaña percepción. Fuente: Elaboración propia. 45
Figura 3.15 Pestaña calibración. Fuente: Elaboración propia. 47
Figura 3.16 Entorno físico ideal para actividades de almacenamiento, carga, descarga de bienes. Fuente: Elaboración propia. 48

Figura 3.17 Objetos físicos 49
Figura 3.18 a) Mundo en Gazebo b) Entorno de trabajo del robot. Fuente: Elaboración propia. 49
Figura 3.19 Ruta de Aprendizaje. Fuente: Elaboración propia. 50
Figura 3.20 Proyecto Pickup-Master. Fuente: Elaboración propia. 50
Figura 3.21 Ventana de Asistente de Moveit. Fuente: Elaboración propia. 51
Figura 3.22 Desplazamiento del robot a posición deseada. Fuente: Elaboración propia. 52
Figura 3.23 a) Posición inicial b) posición destino. Fuente: Elaboración propia. 52
Figura 3.24 a) Posición 1 b) Posición 2 c) Posición 3. Fuente: Elaboración propia 53
Figura 3.25 a) Posición de reconocimiento a objeto azul. Fuente: Elaboración propia b) Reconocimiento objeto azul. Fuente: Elaboración propia 53
Figura 3.26 Publicación del centroide en tópico /gamora/objeto/Centroide. Fuente: Elaboración propia 53
Figura 3.27 a) Objeto en escala de grises b) Contornos del objeto c) Objeto de forma redonda reconocido. Fuente: Elaboración propia 54
Figura 3.28 Centroide de objeto de forma redonda en tópico /gamora/objeto/Centroide. Fuente: Elaboración propia 54
Figura 3.29 a) Posición de reconocimiento b) Objeto verde y rojo reconocidos. Fuente: Elaboración propia 54
Figura 3.30 Tópico /gamora/rojo/Centroide para objeto rojo. Fuente: Elaboración propia 55
Figura 3.31 Tópico /gamora/verde/Centroide para objeto verde. Fuente: Elaboración propia 55
Figura 3.32 a) Posición de reconocimiento b) Reconocimiento objeto azul. c) Agarre manual del objeto d) d) Carga manual del objeto. Fuente: Elaboración propia 56
Figura 3.33 Terminal con TeleopKey. Fuente: Elaboración propia 56
Figura 3.34 a) Posición de reconocimiento b) Reconocimiento de objeto color verde. Fuente: Elaboración propia 57
Figura 3.35 a) Agarre objeto color verde b) Carga objeto color verde. Fuente: Elaboración propia 57
Figura 3.36 a) Posición de reconocimiento. b) Reconocimiento objetos color verde y rojo. Fuente: Elaboración propia 57
Figura 3.37 a) Agarre objeto color rojo b) Almacenamiento en estantería objeto color rojo. Fuente: Elaboración propia 58
Figura 3.38 a) Agarre objeto color verde. b) Almacenamiento en estantería objeto color verde. Fuente: Elaboración propia 58
Figura 3.39 Obtención del Token de Página web iBotAcademy [32]. 59
Figura 3.40 Funcionamiento procesal de paquete verificador. Fuente: Elaboración propia 59
Figura 3.41 Ejecución del paquete verificador sesión 1. Fuente: Elaboración propia 61
Figura 3.42 Ejecución de paquete verificador sesión 2. Fuente: Elaboración propia 62

Figura 3.43 Ejecución Paquete verificador sesión 3. Fuente: Elaboración propia 63
Figura 3.44 Diagrama de nodos Proyecto Pickup-Master. Fuente: Elaboración propia.....65
Figura 3.45 a) Reconocimiento objeto forma redonda. b) Carga objeto c) Descarga objeto en base del turtlebot. Fuente: Elaboración propia.....65
Figura 3.46 a) Reconocimiento objeto verde. b) Carga objeto verde desde estantería. c) Descarga objeto verde en base del turtlebot. Fuente: Elaboración propia.....66
Figura 3.47 Verificador fase 1. Fuente: Elaboración propia. 67
Figura 3.48 Verificador de la fase 2. Fuente: Elaboración propia. 68
Figura 3.49 Paquetes verificadores de Pickup-Master.....68
Figura 4.1 Escenario real IEP (Plataforma educativa innovadora). Fuente: Tomado de [33].....70
Figura 4.2 Pruebas de campo del aplicativo EDU-ARM en la Universidad del Valle.....81
Figura 4.3 Resumen de resultados de la encuesta de evaluación del ambiente de aprendizaje. 82
Figura 5.1 Robots disponibles en Laboratorio de robótica móvil de la Universidad del Valle. Fuente: Elaboración propia. 85
Figura 7.1 Estructura del Repositorio GitHub 90

1. CAPÍTULO I: INTRODUCCIÓN

"*La robótica abre la puerta a un micro-mundo de aprendizaje motivador y entretenido. El establecimiento de un vínculo entre el mundo digital y el mundo físico ayuda a presentar a los alumnos una tecnología clave para el futuro*". – Didier Roy

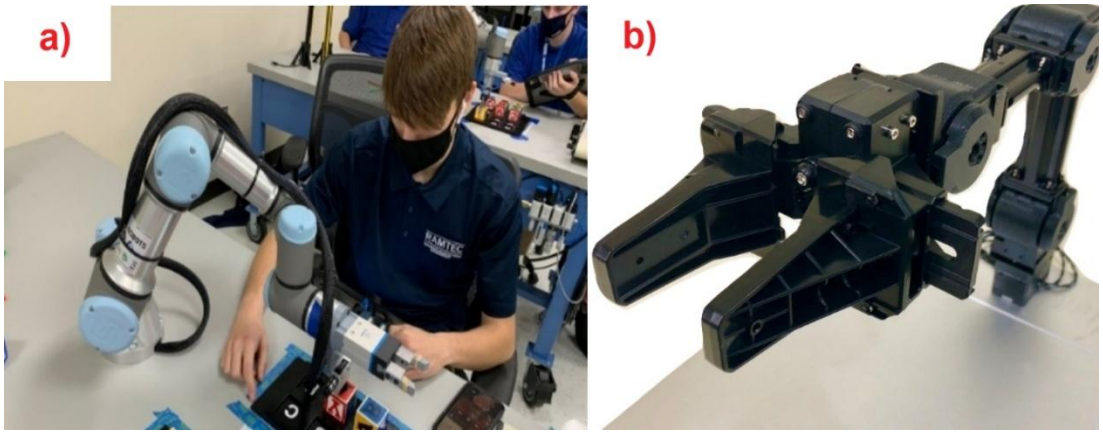


Figura 1.1 *a) Robot Universal robots en la educación. Fuente: Tomado de [1]. b) Robot OpenMANIPULATOR-X real. Fuente: Tomado de [2].*

1. CAPÍTULO I: INTRODUCCIÓN 11
1.1. INTRODUCCIÓN 11
1.2. PLANTEAMIENTO DEL PROBLEMA 13
1.3. OBJETIVO 14
1.3.1. OBJETIVO GENERAL 14
1.3.2. OBJETIVOS ESPECÍFICOS 14
1.4. SOLUCIÓN PROPUESTA 15
1.5. ESTRUCTURA DEL DOCUMENTO 16

1.1 INTRODUCCIÓN

La Industria 4.0, también conocida como la cuarta revolución industrial, se caracteriza por la integración de tecnologías avanzadas que transforman los procesos productivos mediante sistemas conectados, automatización y análisis de datos. Este paradigma, impulsado por el desarrollo de tecnologías como el Internet de las Cosas (IoT), la inteligencia artificial, la robótica avanzada y la fabricación inteligente, está redefiniendo el panorama de la producción de bienes y servicios. En este nuevo entorno, las fábricas tradicionales están evolucionando hacia fábricas inteligentes, donde las operaciones son monitoreadas y optimizadas en tiempo real [3].

A medida que esta revolución tecnológica sigue expandiéndose, la preparación de los futuros profesionales se vuelve una prioridad urgente. Es fundamental que los entornos educativos adapten sus métodos de enseñanza, proporcionando a los estudiantes oportunidades para interactuar con estas tecnologías de vanguardia, como en la Figura 1.2 se observa a los estudiantes interactuando directamente con un robot, lo cual evidencia la integración práctica de estas herramientas en el proceso formativo.



Figura 1.2 *Proyecto "Imagine make" donde usan los robots para la enseñanza. Fuente: Tomado de [4].*

Los robots manipuladores, equipados con software avanzado, representan una de las principales herramientas en estos entornos de aprendizaje. La inmersión práctica de los estudiantes con estas tecnologías no solo les permite aplicar los conceptos aprendidos en el aula, sino que también les dota de las habilidades necesarias para prosperar en un entorno laboral que está en proceso de digitalización acelerada. Se observa en la Figura 1.3 aulas adaptadas para simular entornos industriales, equipadas con robots manipuladores y móviles para fines académicos.



Figura 1.3 *Robot en un entorno de aprendizaje. Fuente: Tomado de [5].*

En los entornos industriales, los robots manipuladores son esenciales para realizar tareas repetitivas o de alta precisión, como el almacenamiento, transporte, carga y descarga de bienes como en la Figura 1.4 en dónde robots móvil-manipulador realizan este tipo de actividades.



Figura 1.4 *Robots que almacenan, distribuyen y recolectan materiales. Fuente: Tomado de [3].*

No obstante, la implementación eficiente de estas tecnologías enfrenta varios desafíos. Uno de los principales retos es optimizar la coordinación y ejecución de las tareas. Para que un robot manipulador funcione de manera eficiente, es necesario programar sus movimientos de forma precisa, adaptándose a cambios en la disposición de los bienes y reduciendo al máximo los tiempos de inactividad. La incapacidad de gestionar eficazmente estos factores puede resultar en una disminución significativa de la productividad, ineficiencias operativas y mayores costos de operación [3].

Con base en lo anterior, este trabajo de grado propone el desarrollo de una herramienta de software que permita la supervisión y teleoperación de robots manipuladores, en un ambiente educativo, tal que se pueda gestionar eficazmente este tipo de tecnologías orientada a aplicaciones industriales clave como el almacenamiento, transporte y la carga/descarga de bienes.

La herramienta será diseñada para integrarse en un Ambiente de Aprendizaje alineado con los principios de la Industria 4.0, ofreciendo a los usuarios una plataforma interactiva para mejorar su comprensión teórica y desarrollar habilidades prácticas en el manejo de estas tecnologías. Este ambiente está en el contexto de un proyecto de investigación de convocatoria interna identificado CI21257. Esta aplicación no solo será útil como un recurso didáctico, sino que también permitirá a los estudiantes simular y experimentar con situaciones reales del entorno industrial, preparándolos para enfrentar los desafíos del mundo laboral digitalizado.

En resumen, este proyecto busca cerrar la brecha entre la teoría y la práctica en la formación de futuros profesionales, proporcionando un entorno controlado y experimental en el cual los estudiantes puedan interactuar con robots manipuladores simulados. Esto, a su vez, contribuirá a que los usuarios adquieran competencias tecnológicas avanzadas, facilitando su inserción en industrias que demandan un alto nivel de conocimiento en automatización y robótica.

1.2 PLANTEAMIENTO DEL PROBLEMA

La necesidad de tecnología en la vida cotidiana es inevitable, lo que provoca el constante desarrollo de innovaciones tecnológicas que impactan en el sector

industrial día a día. Es por ello por lo que surge el concepto de Industria 4.0, que implica la transformación de fábricas tradicionales en fábricas inteligentes.

Existen perspectivas de que esta transformación afecta a los empleos en las industrias, ya que el trabajo manual tiende a ser reemplazado por sistemas automáticos [6]. Sin embargo, se olvida que, además de la revolución tecnológica, es necesario un cambio en la educación tradicional, ya que el conocimiento sobre el manejo y uso de estas nuevas tecnologías es limitado. Se hace evidente la necesidad de mejorar las habilidades humanas para adaptarse y fomentar el empleo. A esto se le denomina Aprendizaje 4.0 [1].

Por este motivo, en entornos educativos, surge una necesidad apremiante de crear ambientes de aprendizaje para formar empleados capacitados y competitivos en vista de las fábricas digitales del futuro[1]. En ejemplos concretos [3], [7] y [8], se utilizan robots móviles y manipuladores según escenarios típicos de fábricas para que los alumnos amplíen sus conocimientos y competencias en el campo de la industria 4.0 con aplicaciones de procesos específicos automatizados. Estos robots cuentan con un software que permiten al usuario manipularlo, tener un control y monitoreo efectivo de su estado.

Entre estas aplicaciones industriales está el proceso de bodegaje. Grandes empresas cuentan con bodegas cuyo proceso de transporte, y carga/descarga se hace de manera manual. Este procedimiento no siempre es eficiente en cuanto al tiempo, costos, y mayor riesgo de seguridad para quienes lo realizan. Ante esta problemática, se han implementado soluciones inteligentes haciendo uso de la robótica. Esta innovación se presenta como una oportunidad valiosa para fortalecer el aprendizaje en el contexto de las industrias 4.0. En este marco, se propone el desarrollo de un software que permita supervisar y tele-comandar remotamente un robot manipulador para una aplicación de Almacenamiento, Transporte y Carga/Descarga de bienes en un ambiente de aprendizaje humano en las industrias 4.0. Dado lo anterior, y teniendo en cuenta la necesidad surge la pregunta: ¿Cómo desarrollar una herramienta software de supervisión y tele-comando de un robot manipulador simulado para una aplicación de almacenamiento, transporte y carga/descarga de bienes en un ambiente de aprendizaje para la Industria 4.0?.

1.3 OBJETIVO

1.3.1 OBJETIVO GENERAL

Desarrollar una herramienta software de supervisión y tele-comando de un robot manipulador simulado para una aplicación de almacenamiento, transporte y carga/descarga de bienes en un ambiente de aprendizaje para la Industria 4.0

1.3.2 OBJETIVOS ESPECÍFICOS

1. Sintetizar información bibliográfica sobre los métodos y procedimientos existentes para el desarrollo de software con ROS para supervisar y tele-comandar robots manipuladores simulados con el fin de extraer los requerimientos funcionales del sistema.

2. Diseñar una herramienta Software que permita supervisar y telecomandar un robot manipulador simulado para la aplicación de almacenamiento, transporte, y carga/descarga.

3. Implementar una herramienta Software que permita supervisar y telecomandar un robot manipulador simulado para la aplicación de almacenamiento, transporte, y carga/descarga.

4. Realizar pruebas de integración y simulación de campo del funcionamiento para una Aplicación de almacenamiento, transporte y carga/descarga de bienes en un ambiente de aprendizaje para la industria 4.0.

1.4 SOLUCIÓN PROPUESTA

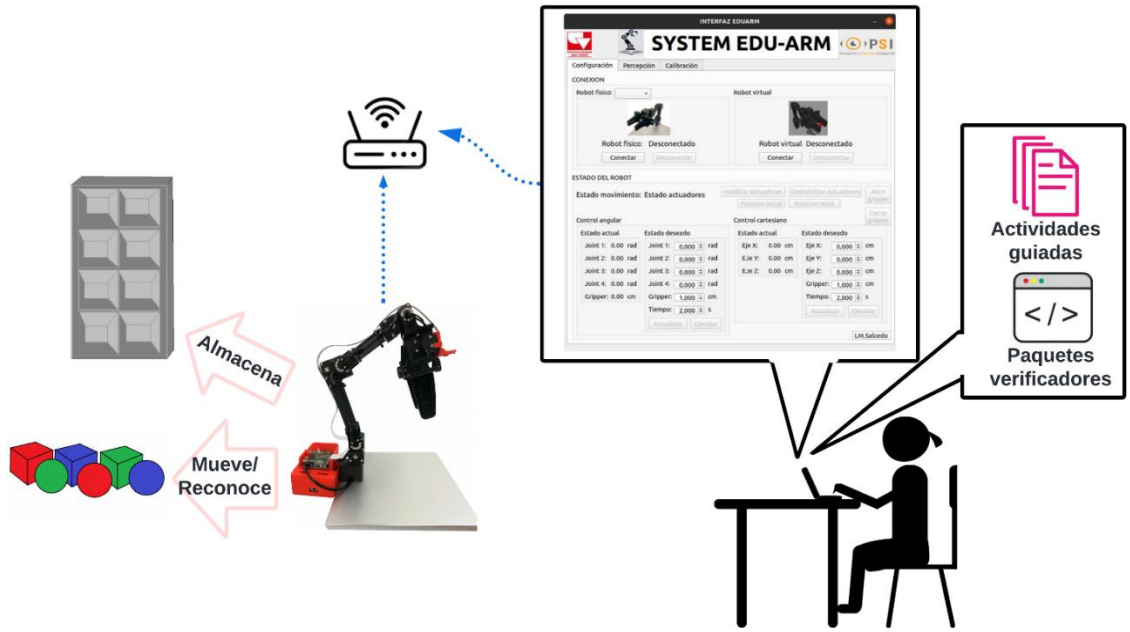


Figura 1.5 *Solución propuesta. Fuente: Elaboración propia.*

Como se muestra en la Figura 1.5, la solución planteada consiste en el desarrollo de un aplicativo interactivo que permita al usuario operar y teleoperar un robot manipulador de manera intuitiva. Este aplicativo incorpora una interfaz gráfica de usuario (GUI) diseñada para facilitar la interacción con el sistema, de forma que el usuario pueda familiarizarse rápidamente con las funciones principales del robot. Entre estas funciones se destacan el control de los movimientos articulares del manipulador y el reconocimiento de objetos a partir de sus características de color y forma. El propósito es ofrecer una experiencia accesible tanto para usuarios principiantes como para aquellos con conocimientos previos en robótica.

Además de las herramientas de operación directa, el aplicativo integra una ruta de aprendizaje estructurada para guiar al usuario a través de escenarios orientados a tareas logísticas, tales como el almacenamiento, transporte, carga y descarga de objetos. Esta ruta está concebida de manera progresiva, de modo que los conocimientos adquiridos en cada sesión sean acumulativos y permitan avanzar hacia retos de mayor complejidad. Cada sesión incluye un reto práctico, cuyo desarrollo es evaluado mediante un paquete verificador en ROS, encargado de suscribirse a los tópicos generados por el usuario, analizar la información y asignar una calificación automática que retroalimente el proceso de aprendizaje, todo este material está a disposición del estudiante.

1.5 ESTRUCTURA DEL DOCUMENTO

Con el objetivo de facilitar la comprensión del contenido presentado a lo largo del documento se explicará el contenido resumen para permitirle al lector tener una visión general del enfoque adoptado para desarrollar la solución propuesta.

- **Introducción:** Este capítulo tiene como propósito contextualizar al lector, presentando el problema abordado y las motivaciones que llevaron a plantear una solución específica.
- **Antecedentes y marco teórico:** Se realiza un análisis de trabajos previos, aplicaciones similares y conceptos clave que sirven de base para la elaboración de la propuesta actual.
- **Desarrollo del aplicativo:** Aquí se detalla el proceso de construcción de la solución, incluyendo la definición de los requisitos funcionales, el diseño de la interfaz y la implementación técnica del sistema.
- **Pruebas y validación**: En esta sección se describen los escenarios de prueba aplicados al aplicativo, junto con los resultados obtenidos.
- **Conclusiones y trabajos futuros:** Finalmente, se reflexiona sobre los aprendizajes y logros del proceso de desarrollo, y se proponen posibles mejoras o líneas de trabajo para ampliar el alcance de la solución en el futuro.

2. CAPÍTULO II: ANTECEDENTES Y MARCO TEÓRICO

"La robótica educativa permite que los estudiantes pasen de ser simples consumidores de tecnología a convertirse en creadores." – Silvia Libow Martínez



Figura 2.1 *(a) UR5; (b) Franka Panda; (c) Tampere RoboLab (center); (d) KUKA Iiwa; (e) Sensor rack. Fuente: Tomado de [9].*

2. CAPÍTULO II: antecedentes y marco teórico 17
2.1. INTRODUCCIÓN 17
2.2. ANTECEDENTES 18
2.3. MARCO TEÓRICO 21
2.3.1. Robots Manipuladores 21
2.3.2. Cinemática de los robos manipuladores 22
2.3.3. ROS (Robot Opeating System) 23
2.3.4. Industrias 4.0 para robótica de manipuladores 25
2.3.5. Tareas de almacenamiento, transporte, carga y descarga 25
2.3.6. Técnicas de reconocimiento por forma 26
2.3.7. Técnicas de reconocimiento por color 26
2.3.8. RUP, Rational Unified Process 27
2.4. OBSERVACIONES FINALES 27

2.1 INTRODUCCIÓN

Este capítulo tiene como objetivo situar al robot manipulador en el contexto de las plataformas tecnológicas usadas en la educación, explorando sus características clave en cuanto a su capacidad para integrarse con el entorno educativo, lenguajes de programación, actuadores, sensores y tecnologías firmware más modernas. Se realiza un análisis destacando sus ventajas y limitaciones, con la finalidad de escoger la tecnología más amigable, interactiva y funcional. Además, el marco teórico ofrece una visión general sobre los componentes esenciales de los robots manipuladores, y que términos serán aplicados sobre ROS para el desarrollo del proyecto.

2.2 ANTECEDENTES

Con el propósito de realizar una consulta bibliográfica del tema, se revisaron antecedentes relacionados con la utilización de software para el control de robots manipuladores como herramienta de aprendizaje en el marco de las industrias 4.0. El objetivo principal era adquirir un conocimiento detallado de las características y particularidades involucradas en la ejecución de proyectos similares. Para llevar a cabo esta revisión, se consultaron diversas bases de datos de renombre, como IEEE, el repositorio virtual Univalle, EbscoHost y Google Académico, empleando palabras clave como "Manipulador", "Robot Móvil", "Industrias 4.0", "Aprendizaje 4.0", "ROS" y "Robots para el Aprendizaje".

Los trabajos mostrados se comparan considerando las características siguientes.

- **Framework:** Marco de trabajo utilizado.
- **Lenguaje de programación**: Lenguajes de programación empleados.
- **Plataforma robótica**: Plataformas robóticas usadas.
- **CPU**: Plataformas de procesamiento de información en el ambiente de aprendizaje.
- **Simulación (Sim)**: Soporte o no de simulación.
- **Remoto / Presencial (R/P)**: Soporte de acceso remoto o no.
- **Sensores**: Sensores empleados por los robots involucrados.
- **Tipo educación (Tipo Ed)**: Soporte de educación formal o no formal.

En el análisis de la literatura disponible, se destaca la relevancia de las decisiones respecto a los marcos de trabajo en distintos proyectos, como se detalla en la Tabla 2.1. Específicamente, los proyectos [10], [11], [9] [5], [12] y [4] sobresalen al elegir ROS (o extensiones compatibles como FlexBe) como su framework principal. Esta elección responde a la necesidad de contar con un entorno modular, escalable y orientado a la simulación robótica. Por otro lado, proyectos como [8] recurren a herramientas de modelado (Petri-net, UML, BPMN, etc.), mientras que [7] trabaja con un entorno propietario de UR3. Este panorama refleja la variedad de enfoques y estrategias empleadas en la investigación en robótica, destacando la adaptabilidad de los desarrolladores a sus contextos y necesidades particulares. La toma de decisiones en torno a los entornos de desarrollo emerge como un componente esencial que influye significativamente en el desarrollo y éxito de los proyectos robóticos.

Tabla 2.1 *Tabla comparativa características relevantes de los antecedentes.*

Ref.	Framew.	R/P	Sensores	Leng. Prog.	Tipo Ed.	Platf.	CPU	Sim
[3]	No	Presencial	LIDAR, Ultrasónico, IMU, Cámara, proximidad, etc.	No	Formal	Robot manipulador, robot móvil.	Raspberry Pi 3B+, computer, dispositivo móviles(Tablets, movil).	Sí

[4]	Spot SDK	Presencial	Cámaras	Python	Formal y no formal	Spot de Boston Dynamics	PC	Si
[5]	ROS	Presencial	Sensor de visión.	C++	Formal	UR10 + pinza Robotica, Turtlebot 2	PCs embebidos	Si
[7]	Propietario UR3	Presencial	Cámara global	No	Formal	UR3	PC, sist. embebido RPY	No
[8]	Petri-net, UML, EER, BPMN, EPC	Ambos	Cámara global	No	Formal	Dobot Magician	PC	Sí
[9]	ROS	Ambos	2D/3DToF cámaras, LiDAR, GNSS, IMU, Pos.	Python, C++, MATLAB,etc.	Formal y no formal	UR5, Panda, Kuka, Turtlebot, Mir100	PC, embebidos, RPY, TX2	Si
[10]	ROS, FlexBe	Presencial	Torque, Posición.	Python, c++	Formal	Franka robot arms	PC, sist. embebido RPY	Si
[13]	No	Presencial	Proximidad, cámara	Python	Formal	Brazos robóticos, robot móvil	PC, PLC, Sist. Embebido RPY	No
[11]	ROS	Presencial	LiDAR.	Python	Formal	Ufactory xArm 6, DashGo B1	PC, PLC	Si
[12]	Diversos	Presencial	cámaras (RGB/ RGB-D)	Varios	Formal	15 robots industriales distribuidos en mini-factorias	PC	Si

En el contexto actual, donde las metodologías de aprendizaje se transforman hacia modalidades remotas, la Tabla 2.1 evidencia que la mayoría de los antecedentes mantienen una modalidad presencial en sus procesos formativos, como ocurre en [10], [7], [13], [5], [12] [4] y [3] .Esto se justifica debido a la interacción directa requerida con robots manipuladores, sensores físicos y plataformas embebidas. Aunque algunos estudios, como [8] y [9], adoptan modalidades mixtas, la predominancia de la presencialidad revela la necesidad de manipular equipos reales para asegurar un aprendizaje práctico adecuado. Estos antecedentes evidencian la complejidad inherente en la interacción con tecnologías que requieren una presencia física para su correcto manejo y funcionamiento. A pesar de la creciente importancia de la educación remota, estos hallazgos subrayan la persistencia de enfoques tradicionales en algunos

contextos educativos, resaltando la necesidad de explorar soluciones que puedan integrar de manera efectiva tanto la presencialidad como la accesibilidad remota en el proceso de aprendizaje.

La interacción de los robots con su entorno de aprendizaje se manifiesta mediante una amplia variedad de sensores, como se observa en la Tabla 2.1. La mayoría de los proyectos integran cámaras para la percepción del entorno o sensores de visión, presentes en [3], [7], [8], [10], [13] [9],[5], [12] y [4]. Asimismo, varios antecedentes incorporan LiDAR, como [3],[11] y [9] mientras que sensores de proximidad se encuentran en [3], [13] y [9]. Proyectos como [10] destacan por el uso de sensores de torque y posición, fundamentales para aplicaciones con robots colaborativos como Franka Emika. Otros estudios, como [9], amplían sus capacidades mediante IMU, GNSS y cámaras 2D/3D ToF. Esta variedad de sensores desempeña un papel esencial al proporcionar datos detallados y multidimensionales sobre el entorno y el propio robot, permitiendo una comprensión más completa de su comportamiento y rendimiento. La combinación de estas tecnologías sensoriales contribuye a la mejora continua de la interacción robot-entorno en el ámbito del aprendizaje y la investigación robótica.

Una consideración clave al interactuar con robots manipuladores es la elección del lenguaje de programación. Según la Tabla 2.1, los lenguajes más utilizados son Python y C++, presentes en proyectos como [10], [11], [9], [5], [12] y [4], debido a su compatibilidad con ROS y su flexibilidad para el desarrollo robótico. En contraste, proyectos como [8], [7] y [3] no especifican un lenguaje de programación, ya sea por utilizar herramientas de modelado, entornos propietarios o plataformas sin programación explícita. Esto refleja un claro predominio de Python y C++ como lenguajes estándar en sistemas robóticos actuales.
Aunque el enfoque principal de los estudios se desarrolla en contextos de educación formal, como se observa en [5], [7], [8], [3], [10], [11], y [12] también se reconoce la presencia de experiencias en educación no formal, especialmente en proyectos [9] y [4], donde los robots se integran a actividades de formación continua o capacitación industrial. Este fenómeno refleja una percepción más amplia de la educación, donde el aprendizaje no se limita únicamente a las aulas universitarias, sino que se extiende a un espectro más amplio de aprendices, incluyendo aquellos que buscan mejorar y actualizar sus habilidades en el ámbito laboral.

Ahora bien, teniendo una visión amplia de los desarrollos e implementaciones que se han llevado a cabo en el tema, es posible destacar las propiedades más relevantes de este tipo de sistemas. En primer lugar, se encuentra la importancia del lenguaje de programación, el cual determina la complejidad al desarrollar la herramienta software, así como su nivel de intuición y facilidad de uso para los usuarios. Además, es fundamental considerar el tipo de sensores del robot que permitan identificar, clasificar y ubicar productos para la aplicación de almacenamiento. Estos sensores, como los de torque y posición, ofrecen monitoreo del estado actual del robot y brindan precisión. Del mismo modo, la

elección del controlador es crucial para garantizar un rendimiento preciso y eficiente, teniendo en cuenta la compatibilidad con los sensores y la capacidad de procesamiento. Finalmente, se destaca la selección de la modalidad de funcionamiento del robot. Si su objetivo es participar en un ambiente de aprendizaje humano, la modalidad remota no debe ser descartada.

2.3 MARCO TEÓRICO

En esta sección se presentan los conceptos fundamentales que facilitan la comprensión del funcionamiento del sistema. Dichos conceptos pertenecen al ámbito de la tecnología, con un enfoque específico en el campo de la robótica.

2.3.1 Robots Manipuladores

Un robot manipulador es un tipo específico de robot diseñado para realizar tareas de manipulación física, es decir, para mover y controlar objetos en su entorno. Estos robots son comúnmente utilizados en entornos industriales para realizar diversas tareas, como ensamblaje, soldadura, paletización, carga y descarga, entre otras [3]. Como se observa en la Figura 2.2 dónde robots manipuladores son usados para el reciclaje automatizado de residuos electrónicos.

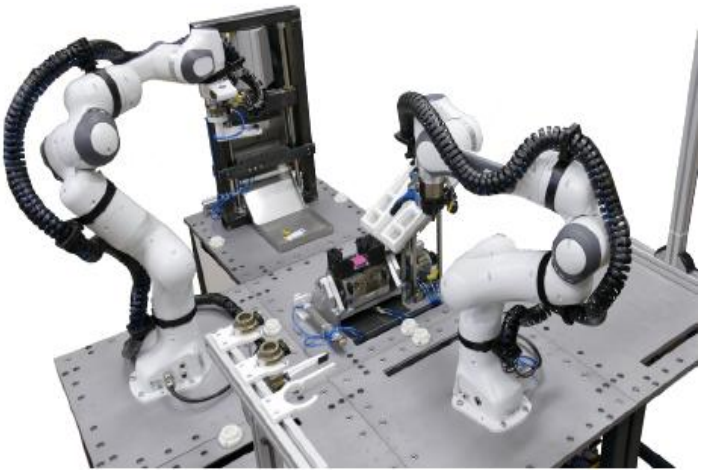


Figura 2.2 *Célula robótica reconfigurable para el reciclaje automatizado de residuos electrónicos. Fuente: Tomado de [10].*

Características clave de los robots manipuladores:

- **Brazo Mecánico**: Los robots manipuladores tienen un brazo mecánico compuesto por articulaciones y segmentos, imitando la estructura y movimientos de un brazo humano. Estas articulaciones permiten al robot manipular objetos en diferentes direcciones y alcanzar diferentes posiciones.
- **Actuadores**: Los actuadores son componentes que generan el movimiento en las articulaciones del brazo. Pueden ser motores eléctricos, hidráulicos o neumáticos, dependiendo del diseño y las necesidades del robot.
- **Garras o Herramientas de Sujeción**: En el extremo del brazo, los robots manipuladores suelen tener garras, pinzas u otras herramientas especializadas que les permiten agarrar y manipular objetos de diversas

formas y tamaños.

- **Sensores**: Algunos robots manipuladores están equipados con sensores, como cámaras, sensores de proximidad o sensores táctiles, para detectar objetos, ajustar su posición y evitar colisiones.
- **Controladores**: Un controlador electrónico supervisa y coordina los movimientos del robot. Los programas informáticos o controladores específicos permiten programar las acciones que debe realizar el robot.
- **Programación**: Los robots manipuladores pueden ser programados para realizar tareas específicas mediante programación manual, programación de trayectorias o mediante el uso de sistemas de visión artificial.

Estos robots son esenciales en la automatización industrial, ya que pueden realizar tareas repetitivas con precisión y eficiencia, mejorando la productividad y liberando a los trabajadores humanos de trabajos monótonos o peligrosos.

2.3.2 Cinemática de los robos manipuladores

La cinemática de los robots manipuladores se refiere al estudio de los movimientos y posiciones de los diferentes elementos (articulaciones, eslabones, extremos efectores) en un robot manipulador. En otras palabras, la cinemática se ocupa de la geometría y las relaciones matemáticas que describen cómo se mueve y se posiciona un robot manipulador en el espacio [14].

La cinemática es esencial para diseñar algoritmos de control, planificación de trayectorias y programación de robots manipuladores, ya que permite comprender y predecir cómo se moverá el robot en función de las entradas que recibe. Además, es clave para evitar colisiones, garantizar movimientos eficientes y lograr la precisión necesaria en diversas aplicaciones industriales y de investigación.

Existen dos aspectos principales en la cinemática de los robots manipuladores: cinemática inversa y cinemática directa.

2.3.2.1 CINEMÁTICA DIRECTA

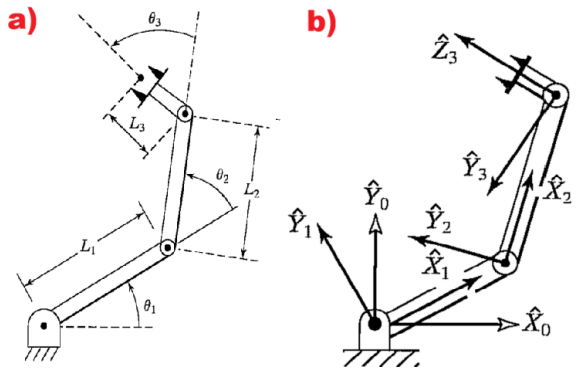


Figura 2.3 *a) Un brazo planar de tres vínculos los ángulos y posición de las articulaciones. b) Manipulador planar de tres vínculos con los ejes de posición. Fuente: Tomado de [15].*

La cinemática directa se refiere al cálculo de la posición y orientación final del

extremo del robot (extremo efector) en función de las variables de las articulaciones. En otras palabras, dadas las posiciones y ángulos de todas las articulaciones, la cinemática directa te dice dónde se encuentra la herramienta final del robot en el espacio tridimensional [14] como se observa en la Figura 2.3a para un robot manipulador de 3 grados de libertad (3-DOF).

2.3.2.2 CINEMÁTICA INVERSA

La cinemática inversa es el proceso opuesto. Aquí, se busca determinar las posiciones y ángulos de las articulaciones necesarios para colocar el extremo efector del robot en una posición y orientación específicas en el espacio. En otras palabras, dado un punto en el espacio tridimensional, la cinemática inversa calcula las configuraciones de las articulaciones requeridas para llegar a esa posición, como se aprecia en la Figura 2.3b.

2.3.3 ROS (Robot Operating System)

ROS, un sistema operativo de código abierto diseñado para robots, ofrece a los desarrolladores las herramientas esenciales para construir aplicaciones robóticas. Actuando como un metasistema operativo, ROS brinda servicios que van más allá de la simple abstracción de hardware, incluyendo el control de dispositivos de bajo nivel, la implementación de funciones comunes, la transmisión de mensajes entre procesos y la gestión de paquetes [16].

En su enfoque, para facilitar el desarrollo de aplicaciones robóticas, ROS introduce conceptos clave:

Tópicos: Estos son canales de comunicación de una sola dirección que posibilitan a los nodos enviar y recibir mensajes, siendo los mensajes unidades de datos para compartir información entre nodos.

Servicios: Ofrecen canales de comunicación bidireccionales, permitiendo a los nodos solicitar y recibir información. Los servicios son esenciales para tareas que requieren interacciones entre dos nodos. Como se muestra en la Figura 2.4, el nodo cliente envía una petición al nodo servidor a través de un servicio en ROS, y el nodo servidor envía la respuesta a través del canal.

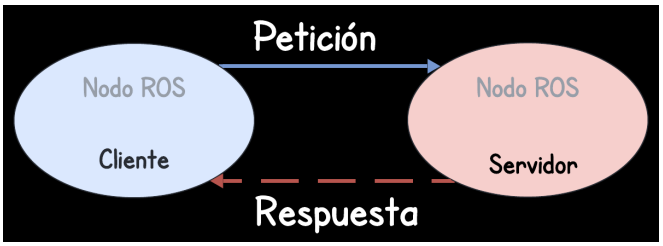


Figura 2.4 *Funcionamiento servicios ROS. Fuente: Elaboración propia.*

Acciones: Representan una extensión de los servicios, habilitando la ejecución de tareas complejas que involucran seguimiento o progreso. Como se muestra en la Figura 2.5, la interfaz de acciones en ROS permite una comunicación más compleja entre el cliente y el servidor mediante múltiples tópicos.

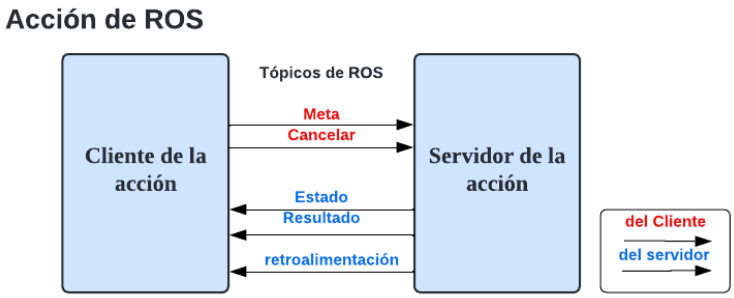


Figura 2.5 *Funcionamiento acciones ROS. Fuente: Elaboración propia.*

La programación en ROS es políglota, admitiendo lenguajes como C++, Python, Octave, MATLAB, Java, Lua, Ruby, .NET y JavaScript. Adoptando un modelo basado en nodos, cada uno de estos representa un proceso específico que realiza una tarea determinada. La comunicación entre nodos se realiza mediante tópicos, servicios y acciones, permitiendo un desarrollo modular y colaborativo de aplicaciones robóticas[10]. Como se observa en la Figura 2.6, los nodos en ROS se comunican mediante la publicación y suscripción a tópicos.

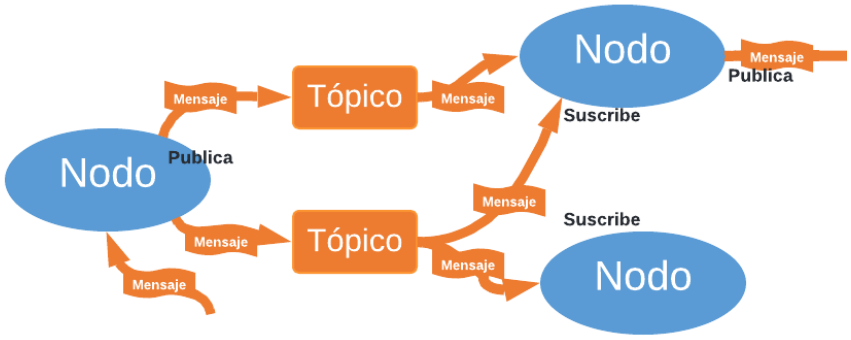


Figura 2.6 *Modelo de funcionamiento de ROS. Fuente: Elaboración propia.*

2.3.4 Simulación robótica con Gazebo

Gazebo es un entorno de simulación robótica que puede ser utilizado en sistemas operativos Ubuntu, el cual permite modelar y visualizar robots en entornos tridimensionales con un motor de física realista [17], como se observa en la Figura 2.7. Además, es compatible con ROS, lo que facilita el desarrollo, prueba y validación de aplicaciones robóticas en contextos académicos sin necesidad de utilizar hardware real.

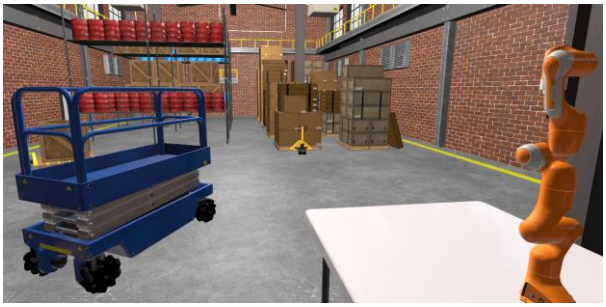


Figura 2.7 *Entorno simulado en Gazebo. Fuente: Tomado de [17].*

2.3.5 Industrias 4.0 para robótica de manipuladores

En el contexto de la Industria 4.0, la robótica de manipuladores se ha beneficiado de la integración de tecnologías digitales en los procesos de producción. Esto ha permitido la creación de robots manipuladores más precisos, eficientes y seguros.

Por ejemplo, la integración de la inteligencia artificial (IA) ha permitido la creación de robots manipuladores que pueden aprender de su entorno y realizar tareas de forma autónoma.

La Industria 4.0 también ha permitido la creación de robots colaborativos, que son robots diseñados para trabajar en estrecha colaboración con los trabajadores humanos. Los robots colaborativos pueden realizar tareas que son peligrosas o incómodas para los trabajadores humanos, lo que permite a los trabajadores humanos centrarse en tareas que requieren habilidades cognitivas y emocionales.

Además, la robótica de manipuladores también se ha beneficiado de la integración de la realidad aumentada (RA). Esto ha permitido a los trabajadores humanos interactuar con los robots manipuladores de una manera más intuitiva y natural [8].

2.3.6 Tareas de almacenamiento, transporte, carga y descarga

En un ambiente industrial, un brazo robótico utilizado para participar en una aplicación de almacenamiento, transporte, carga y descarga de bienes en un ambiente de aprendizaje, es importante resaltar que debe ejecutar las siguientes tareas:

- **Almacenamiento**: El robot manipulador en un entorno de almacenamiento se encarga de identificar, clasificar y ubicar productos de manera eficiente, utilizando sensores y sistemas de visión. Además, gestiona el inventario de forma precisa, asegurando una actualización constante.

- **Transporte**: La función de transporte se limitaría al movimiento de objetos desde un punto a otro dentro del alcance del brazo. En este caso, el enfoque estaría más en las tareas de manipulación y colocación precisa de objetos en distintas ubicaciones, asegurando una gestión eficiente en el entorno de almacenamiento.

- **Carga**: En las tareas de carga, el robot realiza la manipulación segura y eficiente de cargas, coordinando movimientos precisos para cargar productos en áreas designadas. La verificación de cargas garantiza que estén correctamente colocadas y aseguradas durante el transporte.

- **Descarga**: En el proceso de descarga, el robot realiza descargas precisas y seguras en áreas designadas, minimizando los tiempos de inactividad durante las transiciones entre áreas.

2.3.7 Técnicas de reconocimiento por forma

El reconocimiento de objetos por forma es una técnica de visión por computador que permite identificar y clasificar objetos en función de sus características geométricas, independientemente del color o la textura. Algunas técnicas más usadas son las siguientes:

- **Detección de bordes**: mediante operadores Canny [18], se resaltan los bordes que delimitan la forma del objeto [19], como se observa en la Figura 2.8.
- **Aproximación poligonal**: Se analizan los contornos extraídos y se aproximan a polígonos, permitiendo identificar figuras como triángulos, rectángulos o círculos a partir del número de vértices [20].
- **Transformada de Hough:** esta técnica permite detectar formas geométricas específicas como líneas, círculos o elipses a partir de los puntos obtenidos en la etapa de detección de bordes. Es especialmente útil para el reconocimiento de estructuras regulares en imágenes complejas [21].

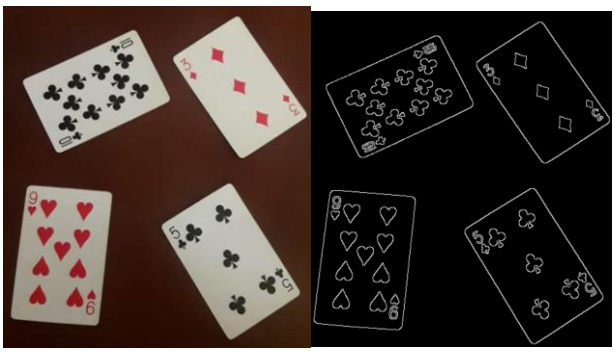


Figura 2.8 *Detector de bordes Canny aplicado a una imagen. Fuente: Tomado de [22].*

2.3.8 Técnicas de reconocimiento por color

El reconocimiento de objetos por color es una técnica de visión por computador que permite identificar y localizar objetos dentro de una imagen o escena a partir de las características cromáticas que presentan. Las técnicas más usadas en algoritmos de programación son las siguientes:

- **Umbralización en espacios de color**: consiste en definir rangos de valores para cada canal de color y generar una máscara binaria que aísle los píxeles que cumplen con dichos rangos [23], como se observa en la Figura 2.9.

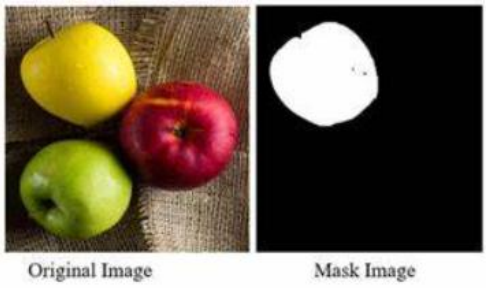


Figura 2.9 *Mask en imagen para detección de objetos color amarillo. Fuente: Tomado de [24].*

- **Histogramas y retroproyección**: se construye un histograma de los colores que caracterizan al objeto y se utiliza la retroproyección para asignar a cada píxel una probabilidad de pertenencia, localizando las regiones que mejor coinciden con el modelo cromático [25]
- **Agrupamiento de colores (clustering)**: algoritmos como *k-means* permiten separar la imagen en grupos de colores dominantes, destacando aquellos que corresponden al objeto [26].

2.3.9 RUP, Rational Unified Process

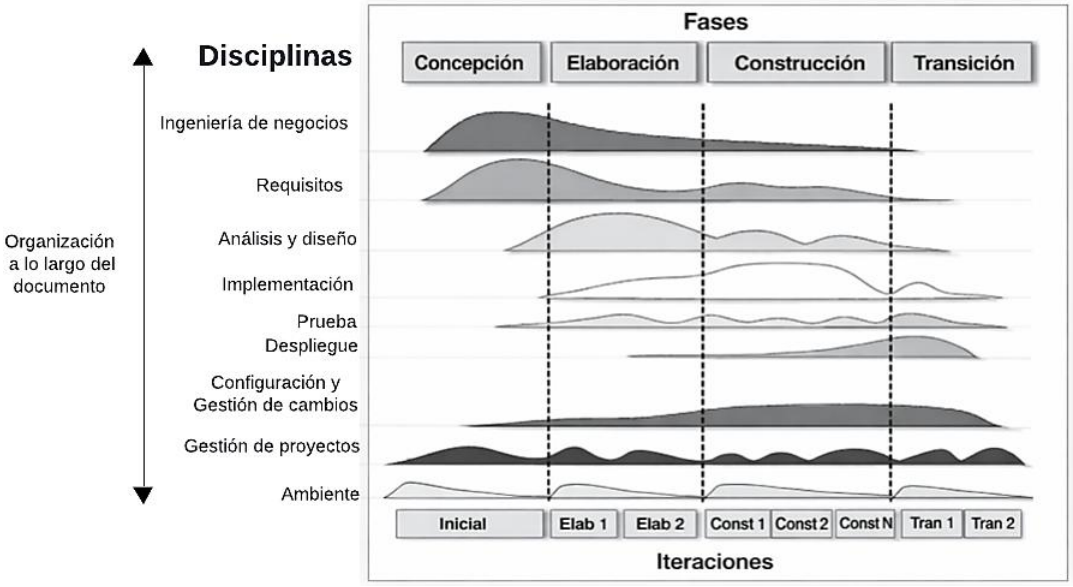


Figura 2.10 *Gráfico de las fases de RUP. Fuente: Tomado de [27].*

La metodología RUP (Rational Unified Process) es un enfoque de desarrollo de software iterativo e incremental que organiza el ciclo de vida del proyecto en cuatro fases: Inicio, Elaboración, Construcción y Transición. Se centra en la definición temprana de la arquitectura del sistema, emplea técnicas de modelado visual para representar diferentes aspectos del software y asigna roles y responsabilidades específicos a los miembros del equipo de desarrollo. RUP promueve la adaptabilidad, permitiendo ajustes según las necesidades del proyecto, y busca mejorar la calidad del software mediante una gestión disciplinada y estructurada del desarrollo [27]. Como se muestra en la Figura 2.10, las disciplinas del proceso de desarrollo se organizan tanto a lo largo del tiempo (fases e iteraciones) como del contenido (actividades y disciplinas), lo que refleja un enfoque iterativo e incremental.

2.4 OBSERVACIONES FINALES

La revisión de los antecedentes permitió identificar que la elección del entorno de desarrollo constituye un factor clave en el diseño y el éxito de los proyectos robóticos. La marcada predominancia de frameworks como ROS y sus

extensiones se justifica por su carácter modular, escalable y su fuerte orientación a la simulación, características especialmente relevantes en contextos de investigación y formación. Sin embargo, la presencia de enfoques alternativos, como el uso de herramientas de modelado o entornos propietarios, pone de manifiesto la diversidad de estrategias adoptadas en función de los objetivos, recursos y restricciones de cada proyecto, evidenciando la capacidad de adaptación existente en el ámbito de la robótica.

Por otro lado, el análisis de los antecedentes revela una clara inclinación hacia la modalidad presencial en los procesos de aprendizaje, principalmente debido a la necesidad de interacción directa con robots, sensores físicos y plataformas embebidas. Si bien algunos estudios incorporan modalidades mixtas, la presencialidad continúa siendo un componente fundamental para garantizar un aprendizaje práctico efectivo. Este escenario evidencia los desafíos que aún persisten para integrar de manera equilibrada la educación remota en entornos robóticos, al mismo tiempo que resalta la oportunidad de desarrollar soluciones que combinen accesibilidad, simulación y manipulación remota de sistemas reales.

Asimismo, se destaca la importancia de los elementos técnicos que conforman estos sistemas, tales como la selección del lenguaje de programación, los sensores y el controlador del robot. El predominio de Python y C++ se explica por su compatibilidad con los principales frameworks robóticos y por su flexibilidad para el desarrollo de aplicaciones complejas. De igual manera, la integración de diversos sensores resulta esencial para mejorar la percepción, precisión y control del robot en su entorno. En este contexto, la consideración de modalidades de operación remota surge como un aspecto relevante, especialmente cuando los sistemas están orientados a entornos de aprendizaje, ampliando las posibilidades de uso y acceso sin comprometer su funcionalidad.

Finalmente, en este capítulo se desarrolló el marco teórico del proyecto, el cual permitió establecer los conceptos y la terminología necesarios para su correcta comprensión y desarrollo. Dicho marco incluyó la metodología RUP, fundamentos de robótica relacionados con robots manipuladores y su cinemática directa e inversa, así como las funcionalidades de ROS su compatibilidad con entornos de simulación como Gazebo y técnicas de reconocimiento de objetos por color y forma. Estos elementos teóricos constituyen la base conceptual sobre la cual se sustenta el diseño y la implementación de la solución propuesta, la cual se abordará en el siguiente capítulo.

3. CAPÍTULO III: DESARROLLO DEL APLICATIVO EDU-ARM

"La educación debe preparar a los estudiantes no solo para el mundo que conocemos, sino para el que aún está por venir." — José Luis Sánchez Perdomo

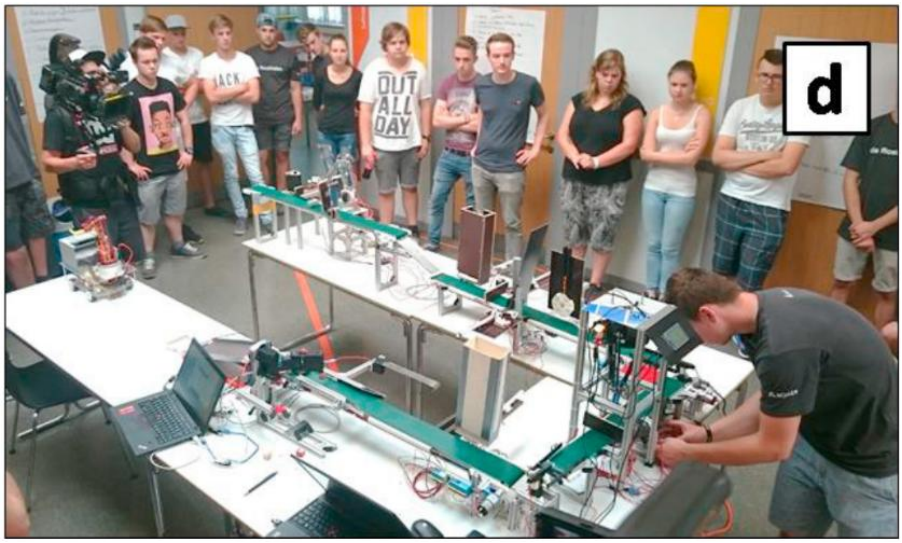


Figura 3.1 *Ambiente portable Mini-Factory. Fuente: Tomada de [12].*

3. CAPÍTULO III: DESARROLLO DEL APLICATIVO EDU-ARM 29
3.1. Introducción 30
3.2. Diseño del aplicativo EDU-ARM 30
3.2.1. Requerimientos funcionales 30
3.2.2. Requerimientos no funcionales 31
3.2.3. Diagrama de concepto 32
3.2.4. Diagrama de nodos. 35
3.3. IMPLEMENTACIÓN DEL APLICATIVO EDU-ARM 35
3.3.1. Configuración y puesta en marcha del robot OpenManipulator-x 35
3.3.2. Interfaz EDU-ARM 42
3.4. IMPLEMENTACIÓN DE LA RUTA DE APRENDIZAJE 47
3.4.1. Entorno de trabajo 48
3.4.2. Ruta de Aprendizaje 50
3.4.3. Crazy Gripper 51
3.4.4. Arm-Eye 53
3.4.5. Arm-Commander 55
3.4.6. Paquetes verificadores 58
3.4.7. Proyecto Pickup-Máster 64
3.5. CONCLUSIONES 69

3.1 INTRODUCCIÓN

El aplicativo EDU-ARM se plantea como una herramienta para la supervisión y el tele-comando de un robot manipulador simulado, orientada a aplicaciones de almacenamiento, transporte y carga/descarga de bienes dentro de un entorno de aprendizaje enfocado en la Industria 4.0. Su objetivo es fomentar el uso de la robótica como una herramienta educativa en el ámbito universitario, especialmente para los estudiantes de Ingeniería, mediante la creación de un entorno de aprendizaje interactivo.

Este capítulo se enfoca en el desarrollo del aplicativo EDU-ARM. La construcción del sistema se llevó a cabo utilizando la metodología RUP, a través de la cual se definieron los requerimientos funcionales y no funcionales que establecen las características necesarias para cumplir con las necesidades del usuario y garantizar la calidad del sistema, incluyendo aspectos como seguridad, disponibilidad y usabilidad. A continuación, se presentan dichos requerimientos junto con un diagrama conceptual que representa la relación entre las funciones del sistema EDU-ARM. Esto se complementa con el diagrama de clases, que define el rol específico de cada clase, y el diagrama de nodos, que muestra el flujo de información entre los distintos componentes del sistema.

El desarrollo del aplicativo se organiza en cuatro aspectos fundamentales: la configuración y puesta en marcha del robot, el módulo de configuración, el módulo de percepción, los cuales se integran y visualizan adecuadamente a través de la interfaz de EDU-ARM. Finalmente, todos estos desarrollos se condensan en las rutas de aprendizaje, que permitirán al estudiante explorar, retarse y poner en práctica cada una de las funcionalidades del aplicativo. Los anexos correspondientes a este proyecto están disponibles en el repositorio de GitHub, en la ruta */linasalcedo01/Open_Manipulator*, y sus respectivas carpetas serán referenciadas a lo largo del desarrollo de este capítulo según sea necesario.

3.2 DISEÑO DEL APLICATIVO EDU-ARM

Para llevar a cabo el desarrollo del aplicativo, se optó por la metodología RUP debido a su estructura basada en ciclos iterativos e incrementales, lo que garantiza la construcción de un producto final robusto y de alta calidad. Esta metodología permitió identificar y definir tanto los requisitos funcionales como los no funcionales, lo cual sirvió de base para la elaboración de los diagramas de conceptos y de clases, esenciales para una planificación eficiente y una entrega estructurada del sistema. Además, como resultado de este proceso, se elaboraron casos de uso y diagramas de secuencia, los cuales están disponibles en la carpeta RUP dentro del repositorio del proyecto.

3.2.1 Requerimientos funcionales

Los requerimientos funcionales se desarrollan para determinar las características necesarias para cumplir con las necesidades del usuario. En el caso del aplicativo los requerimientos funcionales son todos los requisitos que harán al sistema una herramienta para la enseñanza.

Con esto en mente, entonces se diseñan los siguientes requerimientos funcionales, y se categorizan en dos vertientes: Esenciales (E) y Opcionales (O). Esto se observa en la Tabla 3.1.

Tabla 3.1 *Requisitos funcionales*

Ref. #	Software	Categoría
1.0	El aplicativo dispondrá de una interfaz gráfica de usuario (GUI) que le permita controlar manualmente la posición y orientación del efector final del robot y visualizar los datos de los sensores.	E
2.0	El aplicativo dispondrá de una herramienta de filtrado de objetos basado en patrones de color utilizando imágenes captadas por la cámara del robot.	E
3.0	El aplicativo dispondrá de un algoritmo de filtrado de objetos que permita a los usuarios identificar objetos específicos basados en diferentes formas geométricas detectadas en las imágenes captadas por la cámara del robot	E
4.0	El aplicativo permitirá agarrar manualmente un objeto específico utilizando el efector final del robot.	E
5.0	El aplicativo dispondrá de una funcionalidad para grabar posiciones en el plano con el fin de ubicar objetos en ellas.	E
6.0	El aplicativo contará con una funcionalidad que permita a los usuarios tomar objetos desde la parte superior del robot móvil.	E
7.0	El aplicativo contará con una funcionalidad que permita a los usuarios almacenar objetos en una ubicación específica de la estantería.	E
8.0	El software de control del robot manipulador contará con una interfaz de comunicación WEB para enviar y recibir datos desde una aplicación externa.	E
9.0	El aplicativo dispondrá de un ambiente de trabajo simulado utilizando Gazebo	E

3.2.2 Requerimientos no funcionales

Los requerimientos no funcionales permiten establecer las características necesarias para asegurar la calidad del sistema, considerando aspectos como la seguridad, la disponibilidad y la usabilidad. Asimismo, estos requerimientos son fundamentales para la selección de la tecnología, ya que abordan elementos que no están directamente relacionados con las funcionalidades específicas del software y que harán del aplicativo EDU-ARM un sistema ideal para el uso en la academia. Estos aspectos se detallan en la Tabla 3.2.

Tabla 3.2 *Requerimientos no funcionales*

Ref #	Descripción	Categoría

1.0	El lenguaje de programación usado será Python.	E
2.0	El programa para diseñar la interfaz GUI será QtDesigner	O
3.0	La versión de Ubuntu en la que se desarrollará el aplicativo será Ubuntu 20.04.5	E
4.0	La versión de ROS a utilizar para desarrollar el aplicativo es ROS Noetic	E
5.0	Se usará Gazebo como simulador para ejecutar el entorno virtual	E
6.0	El modelo del robot será Open Manipulator X de Robotis	E

3.2.3 Diagrama de concepto

En la sección 1.4 se presentó la solución propuesta y, en la Figura 3.2, se muestra el diagrama conceptual que resume los componentes fundamentales del sistema. El robot OpenManipulator-X es el encargado de ejecutar las tareas de manipulación: mover, cargar, transportar y descargar objetos. Este robot está equipado con una cámara que le permite capturar información visual de su entorno, elemento clave para integrar la percepción en las actividades planteadas.

Por otra parte, el sistema incluye un aplicativo denominado EDU-ARM, conformado por una interfaz gráfica de usuario (GUI) que actúa como herramienta de acompañamiento durante todo el proceso. A través de la interfaz es posible supervisar y tele operar al robot manipulador, ofreciendo tres funcionalidades principales:

1. Módulo de movimiento, para la configuración y control de trayectorias.
2. Módulo de percepción, orientado al reconocimiento de objetos por color y forma.
3. Módulo de supervisión, que permite monitorear en tiempo real el estado del robot y su entorno.

Finalmente, el aplicativo incorpora una ruta de aprendizaje, diseñada como estrategia pedagógica para guiar al estudiante en el desarrollo de competencias relacionadas con la tele operación y supervisión de robots. Esta ruta está estructurada en sesiones con actividades guiadas y un ejercicio propuesto, el cual es evaluado automáticamente mediante un paquete verificador que se suscribe a los tópicos generados por el estudiante, extrae la información necesaria y asigna una calificación objetiva.

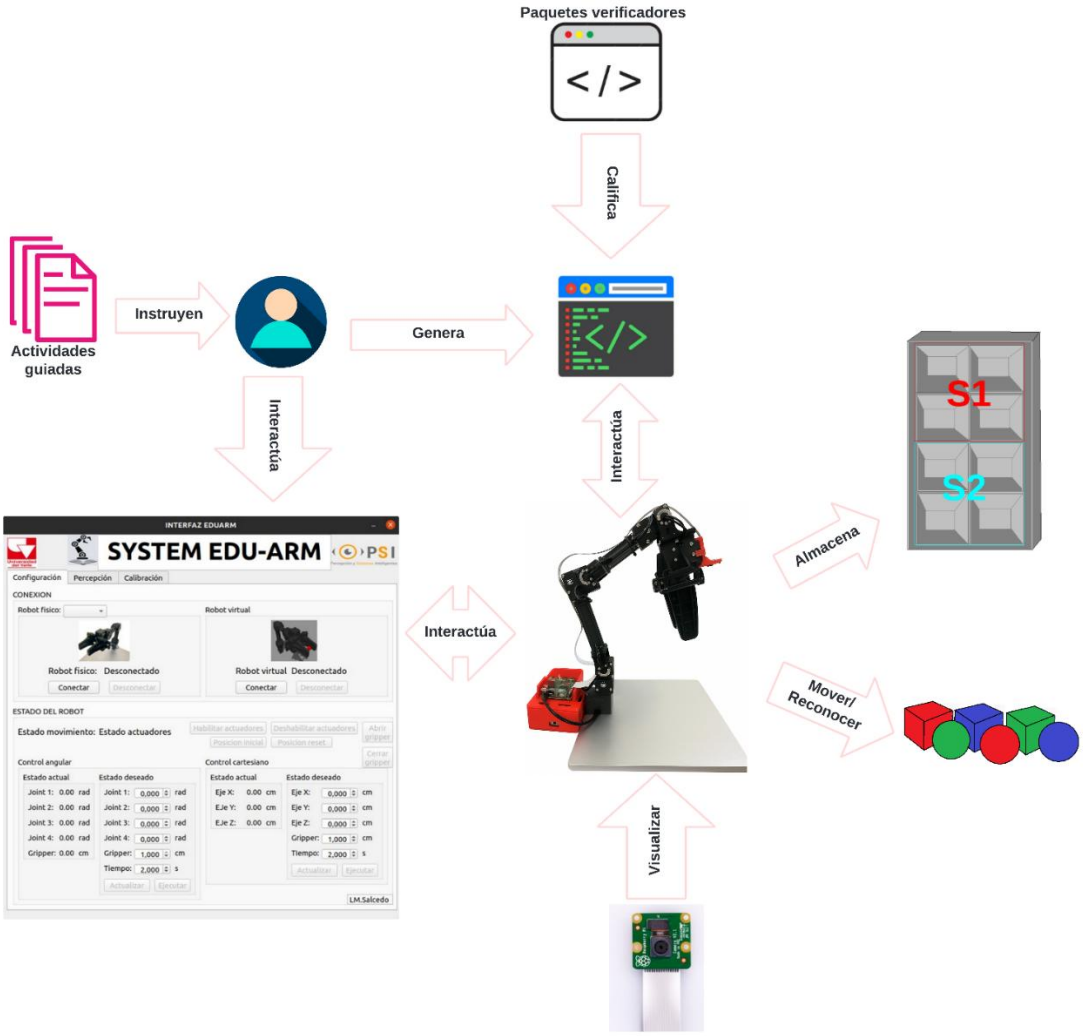


Figura 3.2 Diagrama conceptual aplicativo. Fuente: Elaboración propia

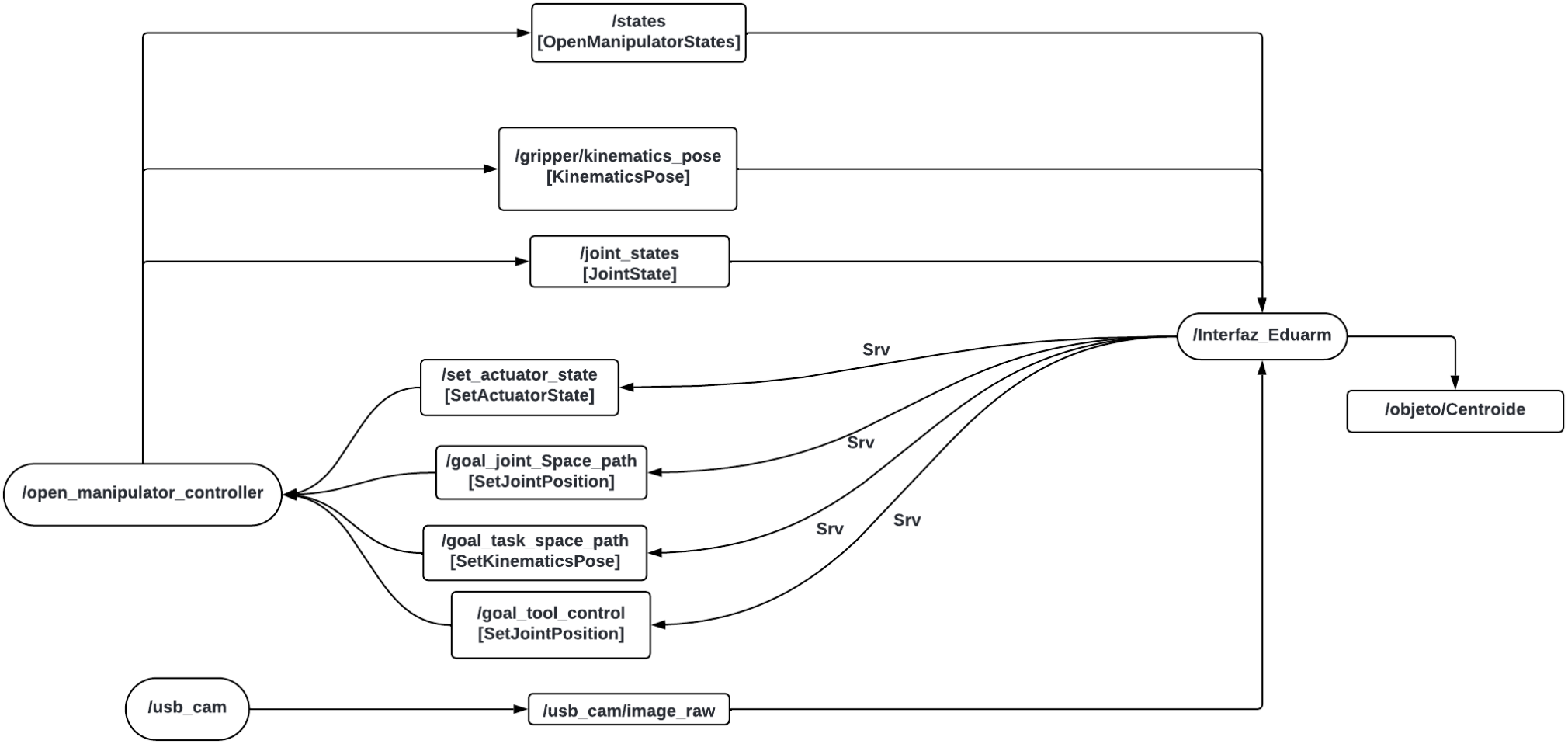


Figura 3.3 Diagrama de Nodos. Fuente: Elaboración propia.

3.2.4 Diagrama de nodos.

El diagrama de nodos de la Figura 3.3 describe la arquitectura de comunicación del aplicativo EDU-ARM en el marco de ROS. Este diagrama corresponde al funcionamiento interno de la interfaz gráfica que acompaña al estudiante durante el desarrollo de las sesiones.
En él se distinguen los siguientes tres nodos:

- **/open_manipulator_controller:** Este nodo se encarga de gestionar el movimiento del robot OpenMANIPULATOR-X, permitiendo controlarlo tanto en el espacio articular como en el espacio cartesiano. Su funcionamiento se puede dividir en dos grandes tareas: por un lado, informar el estado del robot y, por otro, ejecutar sus movimientos. Para la supervisión, publica en el tópico /states el estado general del robot, indicando si está detenido o en movimiento; en el tópico /gripper/kinematics_pose transmite las coordenadas x, y, z y la orientación del efector final; y en el tópico /joint_states informa los valores en radianes de cada articulación junto con su velocidad y esfuerzo. En cuanto al control, el nodo expone una serie de servicios que permiten operar directamente el robot: /set_actuator_state, para habilitar o deshabilitar los actuadores; /goal_joint_space_path, para planificar y ejecutar trayectorias en el espacio articular; /goal_task_space_path, para mover el efector final a una posición específica en el espacio cartesiano; y /goal_tool_control, para abrir o cerrar el efector final.

- **/usb_node:** Este nodo es el encargado de recibir los datos de la cámara, procesar esa información y publicarla en el tópico /usb_cam/image_raw.

- **/Interfaz_Eduarm:** Este nodo ejecuta la aplicación interfaz_edu_arm y se encarga de mostrar en la interfaz gráfica el estado actual del robot. Para ello, se suscribe al tópico /states, mediante el cual identifica si el robot se encuentra detenido o en movimiento; al tópico /joint_states, donde obtiene y visualiza los valores articulares; y al tópico /gripper/kinematics_pose, que le permite representar la posición y orientación del efector final en el espacio cartesiano. Además de la visualización, este nodo gestiona los movimientos del robot a través de la invocación de los servicios que ofrece el nodo controlador. Asimismo, se suscribe al tópico /usb_cam/image_raw para mostrar en la interfaz la imagen captada por la cámara.

3.3 IMPLEMENTACIÓN DEL APLICATIVO EDU-ARM

En esta sección se explicará cómo se configura y se pone en marcha el robot, así como la manera en que las funcionalidades de movimiento y de percepción se configuran para supervisar y teleoperar el robot usando la GUI.

3.3.1 Configuración y puesta en marcha del robot OpenManipulator-x

El OpenManipulator-X de ROBOTIS es un brazo robótico diseñado específicamente para aplicaciones educativas y de investigación. Este robot destaca por ser modular, compacto y completamente compatible con ROS (Robot Operating System). El OpenManipulator-X cuenta con cuatro grados de libertad (DOF) y un efector final tipo gripper, lo que lo convierte en una opción ideal para tareas básicas de manipulación, como agarrar, mover y soltar objetos. Este puede utilizarse en dos modalidades principales: mediante un entorno simulado en Gazebo o a través del robot físico real.

En ambos casos, la operación se lleva a cabo utilizando el controlador oficial de ROBOTIS, el cual permite establecer la comunicación y ejecutar las acciones necesarias para el control del manipulador. La información relacionada con la configuración y uso de estos modos se desarrolla en el presente apartado.

3.3.1.1 Robot OpenManipulator-X simulado (URDF/Xacro)

En la Figura 3.4 se muestra el robot manipulador OpenManipulator-X. Las articulaciones (joints) son los elementos mecánicos que permiten el movimiento relativo entre dos partes rígidas del robot, llamadas eslabones (links). En este robot, se encuentran cuatro articulaciones de revolución, cuya numeración inicia desde la base de manera ascendente hasta el efector final (gripper), identificadas como Joint1, Joint2, Joint3 y Joint4.

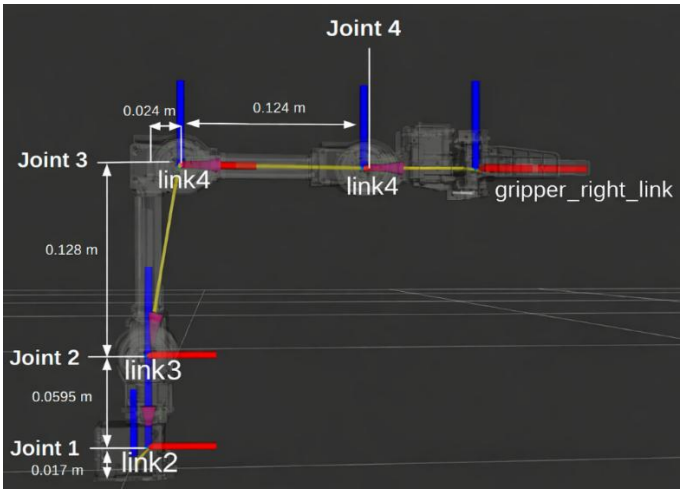


Figura 3.4 *Modelo URDF del Open Manipulator X visualizado en GAZEBO. Fuente: Tomado de[28].*

El archivo URDF (Unified Robot Description Format) del robot manipulador define su estructura física, especificando cada eslabón (link) y articulación (joint) que lo componen. También incluye propiedades físicas como dimensiones, masas, colisiones e inercias. Para facilitar la organización del modelo, se utiliza una versión en formato Xacro, que permite estructurar el archivo de manera más flexible y modular. Con el fin de lograr una simulación más realista y adaptada a las necesidades del proyecto, fue necesario adicionar tres elementos adicionales al URDF: la cámara que se empleará para la percepción visual, la base metálica sobre la que se encuentra montado el robot, y la carcasa que contiene tanto la controladora OpenCR 1.0 como la Raspberry Pi 4B, responsables del procesamiento y control del sistema. Esto se observa en la Figura 3.5.

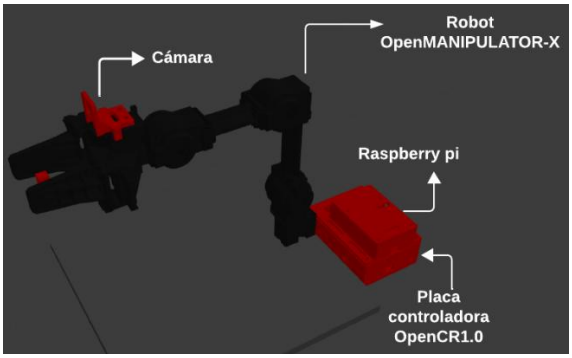


Figura 3.5 *Open Manipulator X en Gazebo. Fuente: Elaboración propia.*

Se integra una cámara al modelo del OpenManipulator-X mediante la incorporación de un joint de tipo fixed, denominado camera_joint, el cual establece una conexión rígida entre el

link5 del brazo y un nuevo eslabón denominado camera_link. Esta configuración permite que la cámara quede acoplada de forma fija al extremo del manipulador. La posición y orientación relativa del sensor se especifica utilizando la etiqueta <origin>, mediante coordenadas xyz para la traslación y rpy para los ángulos de rotación. El nuevo enlace incluye una sección <visual>, donde se define la geometría tridimensional de la cámara a partir del archivo STL asm_cam_v1.stl, junto con un factor de escala que ajusta su tamaño al entorno del robot. Para la detección de colisiones en simulación, la sección <collision> reutiliza el mismo modelo geométrico. Además, se incorpora un bloque <inertial> que define una masa y matriz de inercia simplificadas, lo que permite una simulación física básica de la cámara. El archivo URDF con estos cambios se muestra en los anexos.

Con esta configuración, la cámara queda integrada tanto a nivel visual como físico dentro del modelo del robot. Como se observa en el modelo jerárquico de transformadas de ROS en la Figura 3.6.

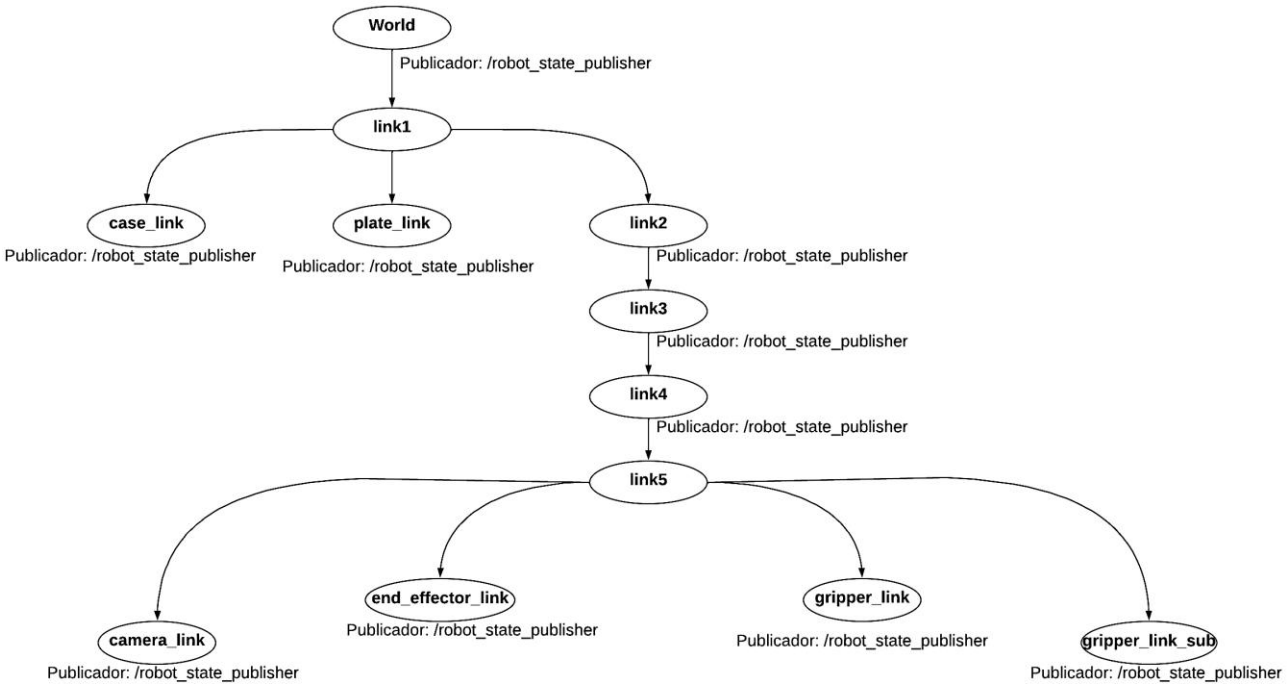


Figura 3.6 *Estructura jerárquica de Joints y Links del Open Manipulator X. Fuente: Elaboración propia.*

Considerando la Figura 3.6, el OpenManipulator-X está compuesto por una secuencia de links (eslabones) rígidos conectados mediante joints (articulaciones) que permiten el movimiento. El enlace raíz del sistema es world o base_link, al que se conecta el primer brazo (joint1) como enlace hijo y cuyo enlace padre es *base_link*. A partir de ahí, cada nueva articulación (como joint2, joint3, joint4) conecta un eslabón con el siguiente, formando una cadena cinemática en la que cada joint tiene un enlace padre (el eslabón anterior) y un enlace hijo (el siguiente eslabón móvil). Por ejemplo, joint2 conecta link1 (padre) con link2 (hijo). Finalmente, al último eslabón (gripper_link) se le añade la cámara y la pinza, unidos mediante un joint fijo que también sigue la relación de padre-hijo para integrarse al árbol del robot. Así, el OpenManipulator-X forma una estructura jerárquica clara y secuencial desde la base hasta el efector final.

3.3.1.2 Robot OpenManipulator-X real

Para la implementación del robot manipulador real, es fundamental comprender que el sistema se compone de varios elementos físicos condensados en la Tabla 3.3. En esta

sección se explicará cómo se configuraron estos elementos para poner a punto el robot real.

Tabla 3.3 *Elementos y características del robot real.*

Ítem	UN	Tipo	Descripción	Imagen	Especificaciones técnicas	Función principal
Motores	5	Actuador	DYNAMIXEL XM430-W350-T, motores inteligentes con control de posición, velocidad y torque.		12V, TTL, 360° continuo, 0.29 Nm	Movimiento de articulaciones del manipulador
OpenCR 1.0	1	Placa Controladora	OpenCR1.0 compatible con ROS y DYNAMIXEL.		Entrada 12V, múltiples puertos TTL y sensores	Control de bajo nivel y comunicación con actuadores
Raspberry Pi 4B	1	Computadora embebida	Microcomputadora para procesamiento general.		8 GB RAM, 4 núcleos, GPIO, USB 3.0, Wi-Fi	Procesamiento de alto nivel, interfaz ROS, visión
Cámara Raspberry pi	1	Sensor	Cámara RGB para visión artificial y detección de objetos.		Módulo de cámara raspberry pi v2.1	Captura de imágenes para reconocimiento visual

3.3.1.3 Configuración de la placa controladora OpenCR 1.0

Para la implementación del robot manipulador en su versión física, se parte de la configuración de la placa controladora OpenCR 1.0, la cual se realiza utilizando el entorno de desarrollo Arduino IDE. Esta placa actúa como una interfaz de comunicación entre la computadora y los actuadores del robot. La OpenCR se conecta al computador vía USB, y su preparación incluye la instalación de las librerías oficiales proporcionadas por ROBOTIS, así como la selección del firmware adecuado desde el gestor de placas del Arduino IDE. Una vez realizado este paso, se carga el sketch usb_to_dxl, el cual permite establecer un puente de comunicación entre la computadora y los motores mediante el protocolo TTL explicado en la web de ROBOTIS [26].

3.3.1.4 Configuración de los motores Dynamixel XM430-W350-T

Después de configurar la OpenCR como interfaz de comunicación, se procede a la configuración individual de cada motor. Los cinco motores DYNAMIXEL XM430-W350-T que conforman el robot se conectan mediante cables TTL al puerto correspondiente en la placa. Al ser actuadores inteligentes, cada motor requiere una configuración previa mediante el software DYNAMIXEL Wizard 2.0 como se observa en la Figura 3.7, también proporcionado por ROBOTIS. Esta herramienta permite detectar, identificar y ajustar parámetros clave en los motores. Es esencial asignar un ID único a cada uno, ya que todos comparten el mismo bus de comunicación. Si dos o más motores tuvieran el mismo ID, responderían simultáneamente a los mismos comandos, generando conflictos operativos. El orden establecido desde la base hasta el efector final (gripper) es: ID 11, ID 12, ID 13, ID 14 e ID 15. Tras asignar los ID, se realiza un escaneo con la herramienta para verificar que

todos los motores fueron correctamente identificados y configurados tal como explica la web de ROBOTIS [28].

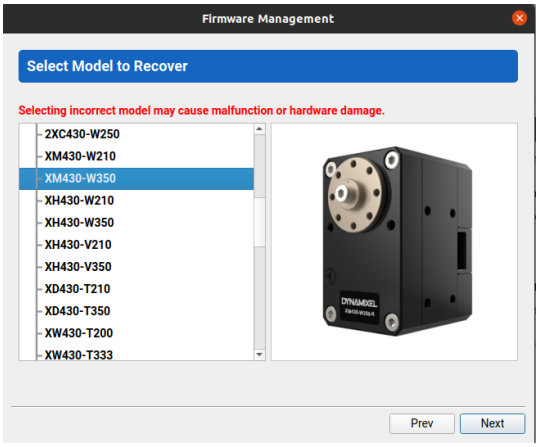


Figura 3.7 *Ventana de gestión de Firmware en Wizard 2.0 para el motor XM430-W350. Fuente: Elaboración propia.*

3.3.1.5 Configuración de la Raspberry Pi

En paralelo, se configura la microcomputadora Raspberry Pi 4B (modelo de 8 GB de RAM), que funcionará como cerebro auxiliar del sistema. Para ello, se emplea el programa Raspberry Pi Imager desde un sistema Ubuntu, eligiendo el sistema operativo Ubuntu 20.04.5 LTS (Focal Fossa) por su compatibilidad con ROS Noetic. Esta imagen se instala en una tarjeta microSD de 32 GB, que servirá como medio de almacenamiento principal para el sistema. Una vez finalizada la instalación, se inserta la microSD en la Raspberry Pi, se alimenta la placa y se conectan periféricos básicos como pantalla (puerto micro-HDMI), teclado y ratón (puertos USB) para continuar con la configuración inicial del sistema [29].

A continuación, se instala ROS Noetic desktop-full (versión 1.15) desde la terminal. Una vez instalado, se crea el entorno de trabajo *catkin_ws/src*, donde se incluirán los paquetes funcionales del robot manipulador. Entre los paquetes necesarios se encuentran: *open_manipulator_msgs, open_manipulator_dependencies y open_manipulator,* los cuales contienen configuraciones predefinidas, parámetros de control, y limitaciones cinemáticas y dinámicas para cada articulación del robot [28].

3.3.1.6 Configuración de la Cámara Raspberry Pi

En cuanto a la configuración del módulo de cámara, se conecta físicamente el Raspberry Pi Camera Module al puerto CSI (Camera Serial Interface) de la placa. En sistemas basados en Raspberry Pi OS, esta configuración se habilita mediante la utilidad *raspi-config*. No obstante, al utilizar Ubuntu 20.04.5 LTS, esta herramienta no está disponible, por lo que la habilitación se realiza de forma manual. Para ello, se edita el archivo /boot/firmware/config.txt añadiendo las líneas necesarias para activar la cámara como se observa en la Figura 3.8. Adicionalmente, se instalan los paquetes libcamera y v4l2-utils, herramientas fundamentales para el manejo de cámaras en sistemas Linux. Se verifica la correcta detección del dispositivo con comandos como *ls /dev/video0 y v4l2-ctl --list-devices*. Finalmente, para integrar la cámara con ROS, se instala el paquete *usb_cam*, que permite publicar las capturas de video en un tópico de ROS, haciéndolas accesibles para procesos posteriores como visión artificial o reconocimiento de objetos [30].

```
start_x=1
gpu_mem=128
disable_camera_led=1
```

Figura 3.8 *Parámetros de configuración de la cámara. Fuente: Elaboración propia.*

3.3.1.7 Configuración de la conexión a Internet

Para realizar la teleoperación del robot se estableció una conexión a través de la Raspberry Pi de este. Para que el robot pueda compartir el mismo espacio ROS con el computador del usuario, es necesario que ambos equipos estén conectados a la misma red y configurados con las variables de entorno correspondientes:

- **ROS_MASTER_URI**: señala en qué dirección se encuentra corriendo el maestro ROS, que en este caso corresponde a la IP de la Raspberry Pi.
- **ROS_HOSTNAME o ROS_IP**: define la dirección IP que ROS utilizará para que otros nodos puedan establecer comunicación con el dispositivo.

En el Laboratorio de Robótica Móvil de la Escuela de Ingeniería Eléctrica y Electrónica se cuenta con un enrutador de red local denominado *robotsrouter*. Para integrar el robot a esta red, se configuró la Raspberry Pi asignándole una IP estática mediante la edición del archivo *50-cloud-init.yaml*, ubicado en el directorio */etc/netplan*. Esta configuración garantiza que la Raspberry Pi mantenga siempre la misma dirección IP al conectarse, asegurando así una comunicación estable y continua con el computador del usuario dentro del entorno ROS.

En este proyecto se configuraron dos robots manipuladores open manipulator-X que, como parte del proceso de identificación dentro del entorno de trabajo, los robots se denominaron *Nebula* y *Gamora*, asignándoles estos nombres con fines de distinción y facilitación en la documentación, interacción y monitoreo durante las actividades de prueba y desarrollo. De esta manera cada robot tiene configurada una red estática definida (Ver Figura 3.9)

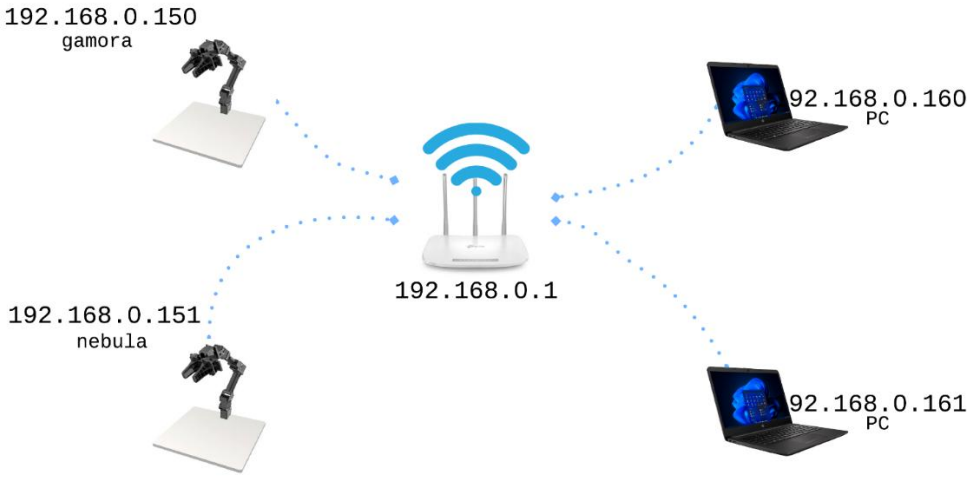


Figura 3.9 *Configuración de red del robot. Fuente: Elaboración propia.*

El computador del usuario puede conectarse a la red del laboratorio configurando su dirección IP en función de la puerta de enlace del enrutador, de manera que se establezca una configuración única y bidireccional. Con ambos dispositivos vinculados al enrutador, es posible establecer la comunicación mediante el protocolo SSH (Secure Shell), lo que permite el acceso remoto al sistema de la Raspberry Pi desde el computador del usuario.

Esto facilita la ejecución y el control de los nodos del robot dentro del entorno ROS. La guía detallada para orientar al usuario en el proceso de configuración y conexión se encuentra disponible en las actividades guiadas del repositorio del proyecto.

3.3.1.8 Ensamble del robot físico Open Manipulator-X

Consecuentemente, una vez los motores estén configurados el paso siguiente es ensamblar el robot físico OpenMANIPULATOR-X siguiendo el manual suministrado por ROBOTIS, obteniendo el resultado de la Figura 3.10 donde se observan los dos robots físicos Nebula y Gamora.



Figura 3.10 *OpenManipulator-X ensamblado. Fuente: Elaboración propia.*

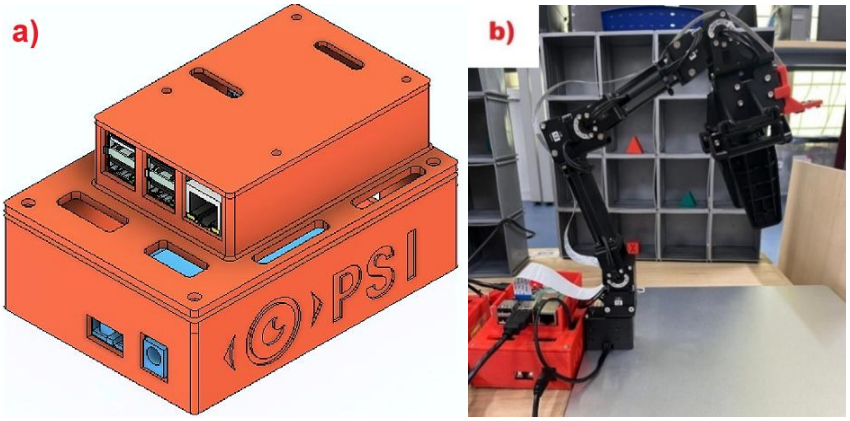


Figura 3.11 *(a) Diseño 3D de la carcasa de la tarjeta OpenCR y la Raspberry Pi. Fuente: Elaboración propia.*

(b) Implementación del robot OpenManipulator X en el entorno físico. Fuente: Elaboración propia.

Finalmente, con todos los elementos del sistema físico correctamente configurados a nivel de hardware y software, el robot queda preparado para integrarse en el entorno de desarrollo y ejecución, replicando las funcionalidades previamente definidas en el entorno simulado, pero ahora con interacción directa en el mundo real. No obstante, dado que se emplean placas embebidas como la Raspberry Pi y la OpenCR 1.0, las cuales son componentes sensibles a factores ambientales como la humedad y a posibles daños por manipulación inadecuada, se considera fundamental proteger estos dispositivos. Con el objetivo de mejorar la durabilidad, seguridad y portabilidad del sistema, así como de facilitar su uso en ambientes educativos, se diseñó una carcasa compacta mediante el software Fusion 360 [31]. Esta carcasa 3D está especialmente adaptada para albergar ambas placas y presenta orificios estratégicamente ubicados para permitir el acceso a sus respectivos puertos de alimentación, comunicación y expansión. La Figura 3.11a muestra el diseño final de dicha carcasa, optimizado para su integración en entornos de aprendizaje de robótica, y la Figura 3.11b ilustra el resultado final del robot físico completamente terminado.

3.3.1.9 Controlador del robot Open Manipulator-X

Para la operación del robot manipulador OpenManipulator-X dentro del entorno de ROS, se hace uso del controlador de ROBOTIS, el cual permite la interacción entre el hardware (o el modelo simulado) y los comandos generados desde otros componentes del sistema. El controlador principal utilizado es el proporcionado por la interfaz ROS del paquete open_manipulator_controller, que define los nodos encargados de recibir trayectorias, calcular cinemática inversa y enviar comandos a los motores del robot. Este nodo interactúa con los modelos físicos definidos en URDF/Xacro y visualizados en Gazebo, permitiendo realizar operaciones de manipulación en un entorno virtual.

Los tópicos usados para ejecutar la comunicación ROS con el robot simulado son los enlistados en la Tabla 3.4.

Tabla 3.4 *tópicos del Controlador*

Tópicos	Descripción
/joint_states	Publica el estado actual de las articulaciones (posición, velocidad y esfuerzo).
/open_manipulator/states	Informa el estado general del manipulador (habilitado, detenido, error, etc.).
/open_manipulator/kinematics_pose	Publica la posición y orientación actuales del efector final (pose cartesiana).

El sistema también ofrece los servicios ROS enlistados en la Tabla 3.5, lo que permite a otros nodos o interfaces de usuario enviar comandos al robot de manera directa. Estos servicios son fundamentales para integrar el módulo de supervisión y tele-comando, permitiendo que las decisiones tomadas por el usuario o por otros algoritmos se traduzcan en acciones del robot dentro del entorno simulado.

Tabla 3.5 *Servicios del controlador del OpenManipulator-X*

Servicio	**Descripción**
/open_manipulator/set_actuator_state	Activa o desactiva los actuadores del manipulador.
/open_manipulator/goal_joint_space_path	Envía un objetivo en espacio articular (posiciones de cada articulación).
/open_manipulator/goal_task_space_path	Envía un objetivo en espacio cartesiano (pose del efector final).
/open_manipulator/goal_tool_control	Controla la herramienta (gripper), permitiendo abrir o cerrar la pinza.

3.3.2 Interfaz EDU-ARM

La interfaz gráfica del aplicativo Edu-Arm fue desarrollada en Qt Designer, herramienta perteneciente al framework Qt que permite construir interfaces de usuario mediante un sistema visual de arrastrar y soltar componentes. La elección de esta herramienta se basó en su compatibilidad con Python a través de la librería PyQt5, su facilidad de integración con sistemas robóticos basados en ROS (Robot Operating System) y la posibilidad de crear entornos gráficos flexibles y escalables para tareas de teleoperación y supervisión.

El diseño de la interfaz se estructuró en tres módulos principales: Configuración, Percepción y Calibración, cada una orientada a un conjunto específico de funcionalidades. A continuación, se detalla la comunicación entre nodos, tópicos y servicios en cada sección.

1.1.1.1. Módulo de Configuración

En la guía de uso de la interfaz de usuario Edu-Arm ubicada en la carpeta Documentos Guía del repositorio del proyecto, se presenta la guía de la interfaz, cuyo objetivo es instruir al usuario en la operación del robot mediante su utilización.

En el Componente 1 de la Figura 3.12, correspondiente a la configuración de la conexión del robot, el usuario puede vincular la interfaz tanto con el controlador del robot real como con el del robot simulado. La validación de esta conexión se realizó verificando el tópico /joint_states, el cual publica en tiempo real las posiciones articulares del OpenManipulator X. La presencia de datos en este tópico confirma que el controlador se encuentra activo y que el robot está listo para su uso.

El Componente 2 de la Figura 3.12 permite visualizar el estado operativo del robot. Si el manipulador está en movimiento, la interfaz informa al usuario sobre dicho estado. Además, mediante los botones “Posición inicial” y “Posición reset”, es posible ubicar al robot en posiciones predefinidas. De igual manera, se incluyen botones específicos para realizar de forma sencilla las acciones de apertura y cierre del gripper.

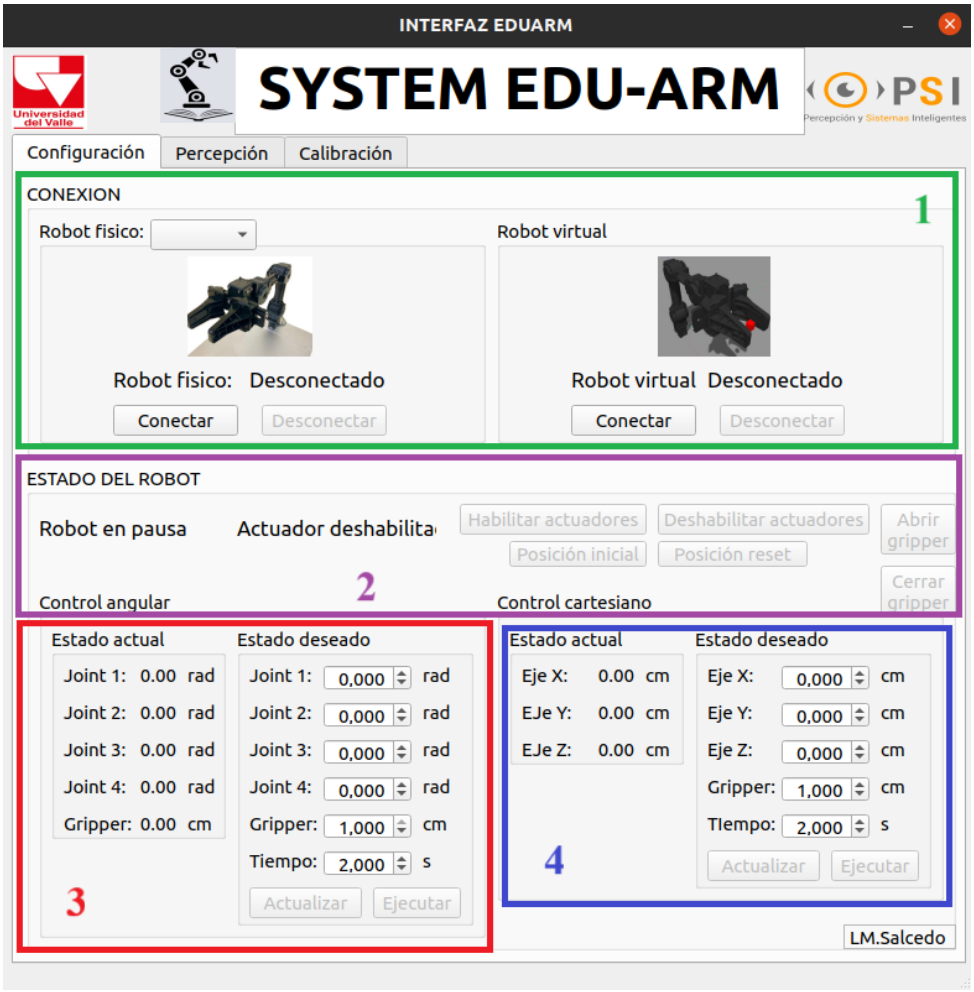


Figura 3.12 *Módulo de Configuración. Fuente: Elaboración propia.*

Para estas funciones, se implementó la publicación en el servicio /goal_joint_space_path, encargado de enviar las posiciones predefinidas (como la inicial y la de reinicio). El control del gripper se gestionó a través del servicio /goal_tool_control, indicando el estado de apertura o cierre de la garra. Adicionalmente, se utilizó el tópico /states para conocer si el manipulador estaba en ejecución o detenido.

En el Componente 3 de la Figura 3.12 se encuentra el control angular del robot. En el lado izquierdo se muestran, en tiempo real, los valores de las articulaciones (Joint1, Joint2, Joint3 y Joint4), expresados en radianes. En el lado derecho, el usuario puede ingresar valores numéricos para mover el robot en el plano articular, es decir, controlar directamente cada articulación hacia un ángulo específico.

Para esta sección, se utilizó la suscripción al tópico /joint_states, con el propósito de mostrar en la interfaz la información de cada articulación en tiempo real. Asimismo, los movimientos articulares se generaron enviando comandos al servicio /goal_joint_space_path, estableciendo trayectorias específicas en el espacio articular. Cabe destacar que el controlador define límites operativos para cada articulación, los cuales se presentan en la Tabla 3.6.

Tabla 3.6 *Especificaciones articulares del robot*

Articulación (Joint)	Nombre ROS	Tipo	Límite mínimo rad(grados)	Límite máximo rad (grados)
Joint 1	joint1 (base)	Revoluta	-1.57 (−90°)	1.57 (90°)
Joint 2	joint2 (hombro)	Revoluta	-1.05 (−60°)	1.57 (90°)
Joint 3	joint3 (codo)	Revoluta	-2.0 (−114.6°)	1.53 (87.7°)
Joint 4	joint4 (muñeca)	Revoluta	-1.8 (−103.1°)	1.8 (103.1°)
Gripper	gripper	Prismatic	0.010 m (abierto)	0.015m (cerrado)

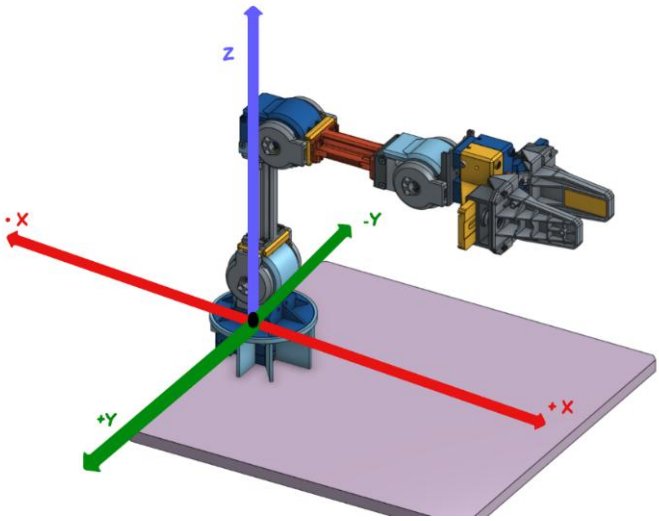


Figura 3.13 *Movimiento Cartesiano. Fuente: Elaboración propia.*

Finalmente, el Componente 4 de la Figura 3.12 corresponde al control cartesiano del robot, enfocado en la manipulación del efector final (gripper). En este apartado, el usuario puede especificar de manera directa la posición y orientación en el espacio tridimensional, mediante coordenadas X, Y y Z como se observa en la Figura 3.13. En lugar de mover manualmente cada articulación, se define un punto exacto en el espacio al que debe llegar el efector, y el sistema calcula automáticamente los ángulos necesarios de las articulaciones mediante cinemática inversa. Este tipo de control es especialmente útil cuando se requiere precisión espacial, como en la toma de un objeto en una ubicación específica o en la ejecución de trayectorias dentro del espacio de trabajo. En este módulo, el estado actual del robot en el espacio cartesiano se visualiza en el lado derecho, mientras que en el lado izquierdo se encuentran las casillas destinadas a la entrada de datos por parte del usuario para llevar al robot a la posición deseada. Se gestionaron movimientos en

el espacio cartesiano publicando en el servicio /goal_task_space_path, encargado de ejecutar trayectorias del efector final a partir de coordenadas en el espacio. En paralelo, se suscribió al tópico /kinematics_pose, que reporta la posición cartesiana actual del gripper, y cuyos datos se mostraron en la interfaz. Cada dato obtenido mediante la suscripción a un tópico es visualizado en tiempo real en la interfaz, garantizando retroalimentación continua al usuario.

1.1.1.2. Módulo de Percepción

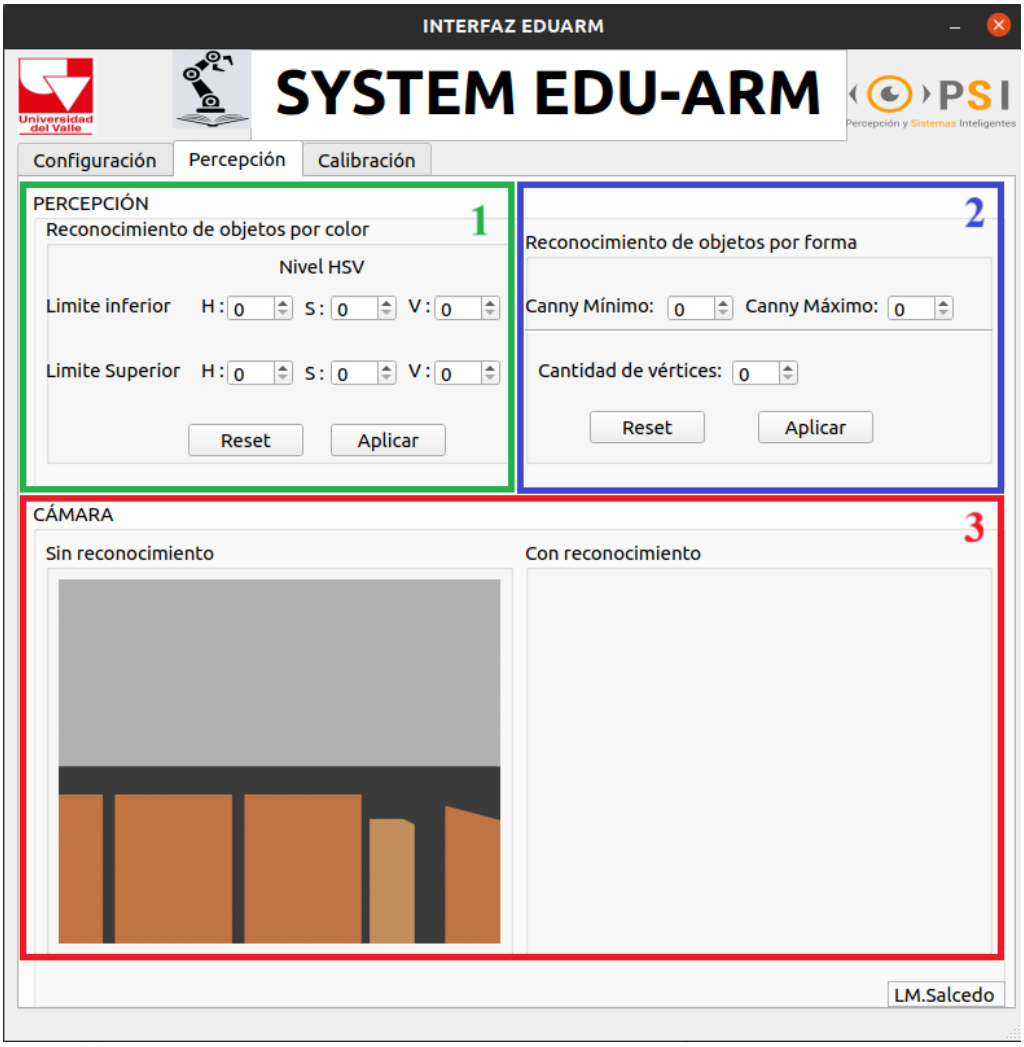


Figura 3.14 *Pestaña percepción. Fuente: Elaboración propia.*

El módulo de percepción estuvo dedicada al procesamiento de imágenes y al reconocimiento de objetos. Para ello, se integró la cámara USB explicado en la sección 3.3.1.6, cuyos datos son publicados en el tópico /usb_cam/image_raw como se observa en el diagrama de nodos de la Figura 3.3.

El componente 1 de la Figura 3.14 se dedica a la funcionalidad de reconocimiento de objetos por color, el usuario debe ingresar el HSV mínimo y HSV máximo para ejecutar el reconocimiento cuyo funcionamiento se desarrolla de acuerdo con el Algoritmo 3.1a.

El Componente 2 mostrado en la Figura 3.14 se encarga del reconocimiento de objetos a partir de su forma. Para realizar esta tarea, la interfaz se conecta al tópico /usb_cam, desde donde recibe en tiempo real las imágenes captadas por la cámara. Una vez que el usuario especifica el rango de valores del filtro de Canny y el número de vértices correspondiente a la figura que desea identificar, el sistema ejecuta el proceso de detección de formas

siguiendo las etapas descritas en el Algoritmo 3.1b. De esta manera, el componente permite analizar las imágenes en vivo y reconocer figuras geométricas según los parámetros definidos por el usuario.

En el componente 3 de la Figura 3.14 en la sección “Imagen sin reconocimiento” se muestra la imagen en tiempo real captado por la cámara del robot y en la sección “Imagen con reconocimiento” con los algoritmos aplicados según corresponda.

```
01 // INICIALIZAR
02    // Capturar imagen desde la cámara del robot
03    IMAGEN ← CapturarImagen()
04    // Convertir imagen de formato BGR a HSV
05    HSV_IMAGEN ← ConvertirBGRaHSV(IMAGEN)
06    // Definir rangos HSV para el color azul
07    HSV_MIN ← [H_MIN, S_MIN, V_MIN]
08    HSV_MAX ← [H_MAX, S_MAX, V_MAX]
09    // Definir tópico para publicar el centroide
10    TOPICO_CENTROIDE ← "/Objeto/Centroide"
11    // Aplicar máscara binaria para filtrar el color azul
12    MASCARA ← InRange(HSV_IMAGEN, HSV_MIN, HSV_MAX)
13    // Encontrar los contornos de los objetos detectados
14    [CONTORNOS, JERARQUIA] ← FindContours(MASCARA, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
15    // Recorrer cada contorno encontrado
16    PARA CADA CONTORNO EN CONTORNOS HACER
17        // Dibujar el contorno sobre la imagen original
18        DibujarContorno(IMAGEN, CONTORNO, COLOR_BLANCO, GROSOR = 9)
19        // Calcular los momentos del contorno
20        M ← CalcularMomentos(CONTORNO)
21        // Verificar que el área del contorno sea diferente de cero
22        SI M["m00"] ≠ 0 ENTONCES
23            CX ← ENTERO(M["m10"] / M["m00"])
24            CY ← ENTERO(M["m01"] / M["m00"])
25            // Dibujar un círculo en el centroide
26            DibujarCirculo(IMAGEN, (CX, CY), RADIO = 5, COLOR_VERDE, RELLENO)
27            // Crear y publicar el mensaje del centroide en el tópico /Objeto/Centroide
28            MENSAJE_CENTROIDE ← CrearMensaje((CX, CY))
29            Publicar(TOPICO_CENTROIDE, MENSAJE_CENTROIDE)
30        FIN SI
31    FIN PARA
32    // Mostrar la imagen procesada
33    MostrarImagen(IMAGEN)
34 FIN
```

```
01 // INICIALIZAR
02    // Capturar la imagen desde la cámara del robot
03    IMAGEN ← CapturarImagen()
04    // Convertir la imagen de formato BGR a escala de grises
05    GRAY_IMAGEN ← ConvertirBGRaGRIS(IMAGEN)
06    // Aplicar el filtro Canny para detectar bordes
07    LEER CANNY_MIN
08    LEER CANNY_MAX
09    LEER LADOS
10    CANNY ← AplicarFiltroCanny(GRAY_IMAGEN, CANNY_MIN, CANNY_MAX)
11    // Encontrar los contornos en la imagen filtrada
12    CONTORNOS ← FindContours(CANNY, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
13    // Recorrer cada contorno encontrado
14    PARA CADA CONTORNO EN CONTORNOS HACER
15        // Aproximar la forma del contorno para identificar la figura
16        APROX ← AproximarContorno(CONTORNO, 0.02 * CalcularPerimetro(CONTORNO), VERDADERO)
17        // Comprobar si los lados encontrados es igual a lados proporcionado
18        SI Longitud(APROX) == LADOS
19            // Calcular los momentos del contorno
20            M ← CalcularMomentos(CONTORNO)
21            // Verificar que el área del contorno sea diferente de cero
22            SI M["m00"] ≠ 0 ENTONCES
23                // Calcular el centroide (cx, cy)
24                CX ← ENTERO(M["m10"] / M["m00"])
25                CY ← ENTERO(M["m01"] / M["m00"])
26                // Dibujar el centroide sobre la imagen
27                DibujarCirculo(IMAGEN, (CX, CY), RADIO = 5, COLOR_AZUL, RELLENO)
28                // Dibujar el contorno del objeto
29                DibujarContorno(IMAGEN, CONTORNO, COLOR_AMARILLO, GROSOR = 9)
30            FIN SI
31        FIN SI
32    FIN PARA
33    // Mostrar la imagen con el contorno y el centroide
34    MostrarImagen(IMAGEN)
35 FIN
```

Algoritmo 3.1 *a) Reconocimiento de objetos por color. Fuente: Elaboración propia. b) Reconocimiento de objetos por forma. Fuente: Elaboración propia.*

1.1.1.3. Módulo de Calibración

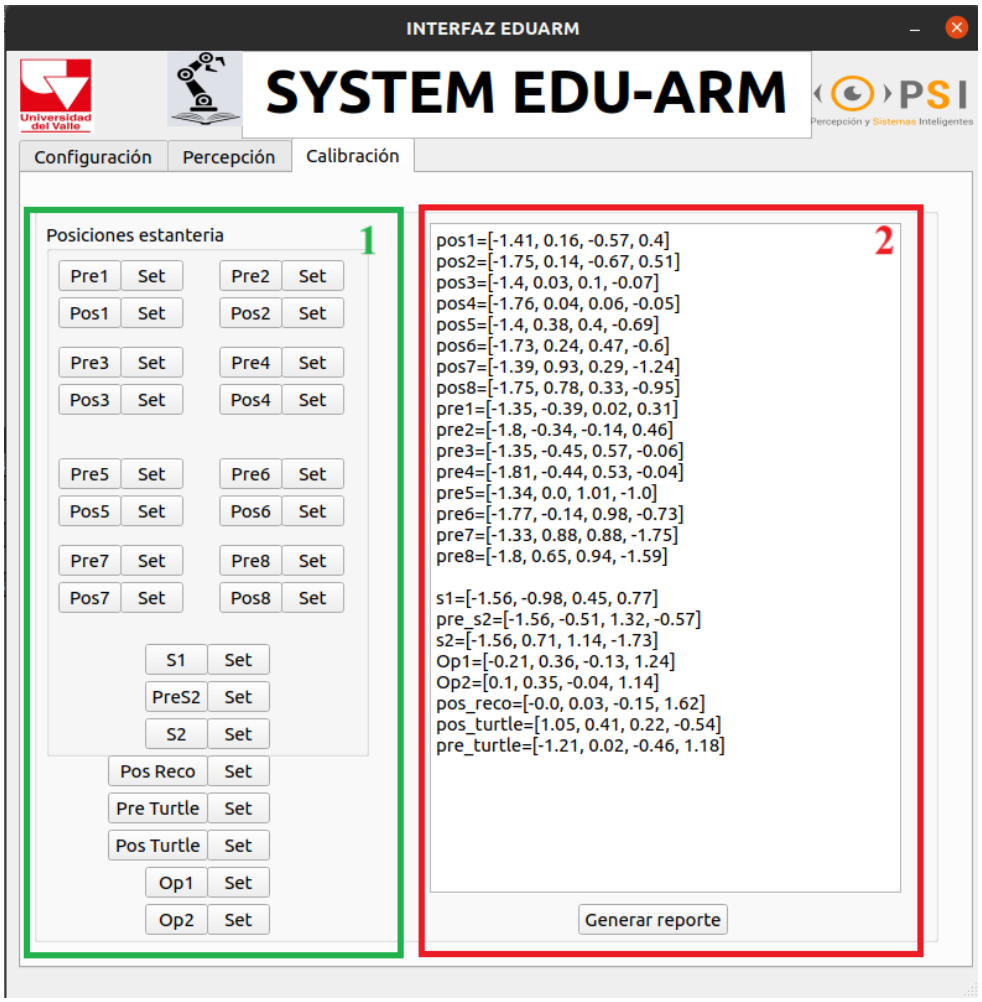


Figura 3.15 *Pestaña calibración. Fuente: Elaboración propia.*

El módulo de calibración fue diseñado como una herramienta práctica que permite al usuario interactuar directamente con el robot y explorar su espacio de trabajo. El objetivo de esta sección es facilitar la configuración de posiciones de prueba y fomentar la familiarización con los movimientos del manipulador.

En el Componente 1 de la Figura 3.15 se encuentran definidos los ocho botones correspondientes a las posiciones de la estantería, así como las ocho posiciones previas asociadas a cada una de ellas. Adicionalmente, se incluyen las posiciones S1 y S2, que permiten visualizar el primer y segundo conjunto de recuadros de la estantería, respectivamente. También se disponen las posiciones Op1 y Op2, que representan ubicaciones de objetos situados en el área del piso del robot.
A través de estos botones, vinculados al servicio /goal_joint_space_path, se ejecutan posiciones articulares predefinidas que representan configuraciones relevantes dentro del entorno de aprendizaje orientado a la Industria 4.0. Al presionar cada botón, se envían los valores articulares correspondientes y el robot se desplaza hacia la postura indicada.

En el Componente 2 de la Figura 3.15, se incorporaron botones Set, encargados de registrar el estado actual de las articulaciones mediante la lectura del tópico /joint_states. Esta información se almacena temporalmente y, al seleccionar la opción “Generar reporte”,

se extraían los valores registrados para mostrarlos en la interfaz, proporcionando así un registro detallado de la configuración articular alcanzada.

3.4 IMPLEMENTACIÓN DE LA RUTA DE APRENDIZAJE

En este apartado se explicará cómo se implementó la ruta de aprendizaje del robot manipulador. Inicia con el desarrollo del entorno de trabajo para la aplicación de almacenamiento, transporte, carga y descarga de bienes en el marco de la industria 4.0. Posteriormente se explicará las partes que la componen.

3.4.1 Entorno de trabajo

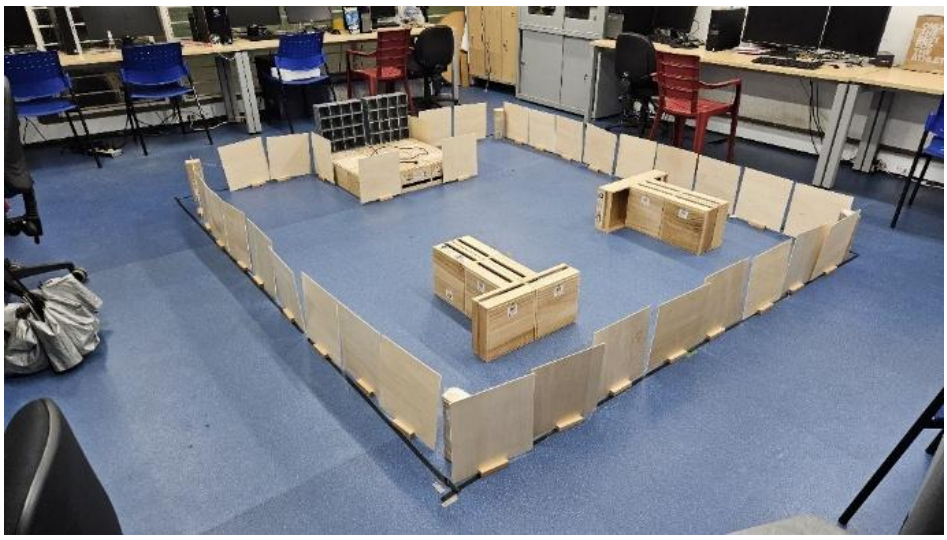


Figura 3.16 *Entorno físico ideal para actividades de almacenamiento, carga, descarga de bienes. Fuente: Elaboración propia.*

Esta sección describe el entorno de trabajo diseñado específicamente para el desarrollo de actividades educativas en el marco del aprendizaje de la robótica industrial. Este espacio debe garantizar las condiciones adecuadas para el correcto desempeño del robot en tareas como almacenamiento, transporte, carga y descarga de bienes. Para el almacenamiento, se definió el uso de una estantería con una configuración de 4x2 (cuatro filas por dos columnas), la cual ha sido dimensionada de forma que permita el ingreso del gripper del robot sin colisiones ni interferencias con la estructura. Además, se determinó que el robot manipulador esté ubicado sobre una plataforma elevada, compuesta por cajas de madera. Esta decisión se toma por dos razones principales: la primera, por motivos de seguridad y visibilidad dentro del entorno educativo; y la segunda, porque se prevé que el robot deba interactuar con una base móvil durante algunas actividades de transporte, carga y descarga por lo que posicionarlo a un nivel superior facilita el acoplamiento y la ejecución eficiente de dicha tarea. En la Figura 3.16 se evidencia el entorno de trabajo del robot real implementado en el laboratorio de robótica móvil de la escuela de Ingeniería eléctrica y electrónica de la Universidad del Valle.

Además, para el diseño de los objetos que representan los bienes a manipular, se emplean cubos y esferas de 0.02 m de diámetro. Los objetos se imprimieron usando impresora 3D en material TPU (Poliuretano Termoplástico) como se observa en la Figura 3.17 que permite una sujeción precisa por parte del gripper del robot OpenManipulator-X. La inclusión de ambos tipos de geometrías busca aportar diversidad funcional, lo cual resulta especialmente útil durante las pruebas de reconocimiento del aplicativo.

Ahora bien, para implementar este diseño en una herramienta de simulación como Gazebo, se utiliza el software de modelado CAD Fusion 360. En esta plataforma, cada uno de los

componentes del entorno es modelado siguiendo fielmente sus dimensiones reales, lo que permite una representación precisa para propósitos de simulación. Una vez finalizado el modelado, las piezas se exportan en formato *.dae* (Collada), el cual es compatible con Gazebo y facilita su incorporación al entorno virtual.



Figura 3.17 *Objetos físicos*

La integración del entorno en Gazebo se lleva a cabo mediante un archivo *.world*, en el cual se especifican las rutas de acceso a los modelos 3D previamente exportados y se define la disposición espacial del escenario simulado. Este archivo permite representar con precisión el entorno físico diseñado, facilitando así su análisis y validación virtual. Además, dentro del mismo archivo se incorporan los objetos que representan los bienes a manipular, definiendo colisiones e inercias compatibles tanto con el mundo creado como con el robot manipulador simulado.

Finalmente, el robot se incorpora al entorno simulado a través de un archivo *roslaunch*, que carga tanto el modelo URDF del robot como el entorno definido en el archivo *.world*. De esta manera, al ejecutar el comando *roslaunch open_manipulator_gazebo open_manipulator_gazebo.launch* , se inicia automáticamente la simulación en Gazebo, con el entorno virtual completamente configurado y el robot posicionado en su ubicación inicial, listo para realizar las pruebas de validación funcional.

En la Figura 3.18a se observa el robot OpenManipulator-X ubicado en su posición inicial dentro del entorno simulado, interactuando con los objetos definidos. Asimismo, en la Figura 3.18b se presenta una vista panorámica del entorno, permitiendo apreciar en su totalidad la disposición del ambiente replicado a partir del laboratorio físico.

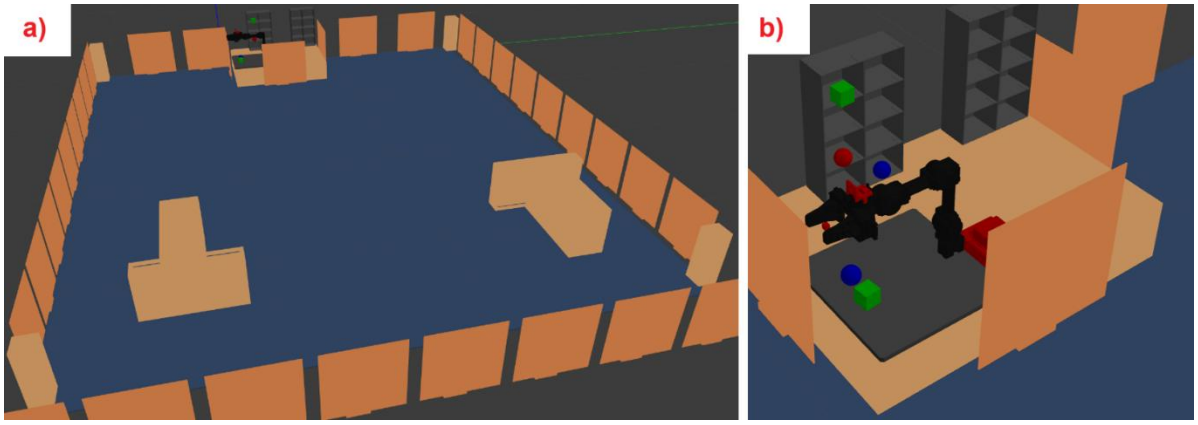


Figura 3.18 *a) Mundo en Gazebo b) Entorno de trabajo del robot. Fuente: Elaboración propia.*

Los paquetes necesarios para la ejecución del mundo simulado en Gazebo se encuentran en la carpeta Paquetes robot simulado dentro del repositorio del proyecto.

3.4.2 Ruta de Aprendizaje

En esta sección se describe la ruta de aprendizaje que el estudiante debe seguir para interactuar con el aplicativo. Tal como se observa en la Figura 3.19, dicha ruta está organizada en tres sesiones, cada una de ellas conformada por dos actividades guiadas y un ejercicio propuesto.

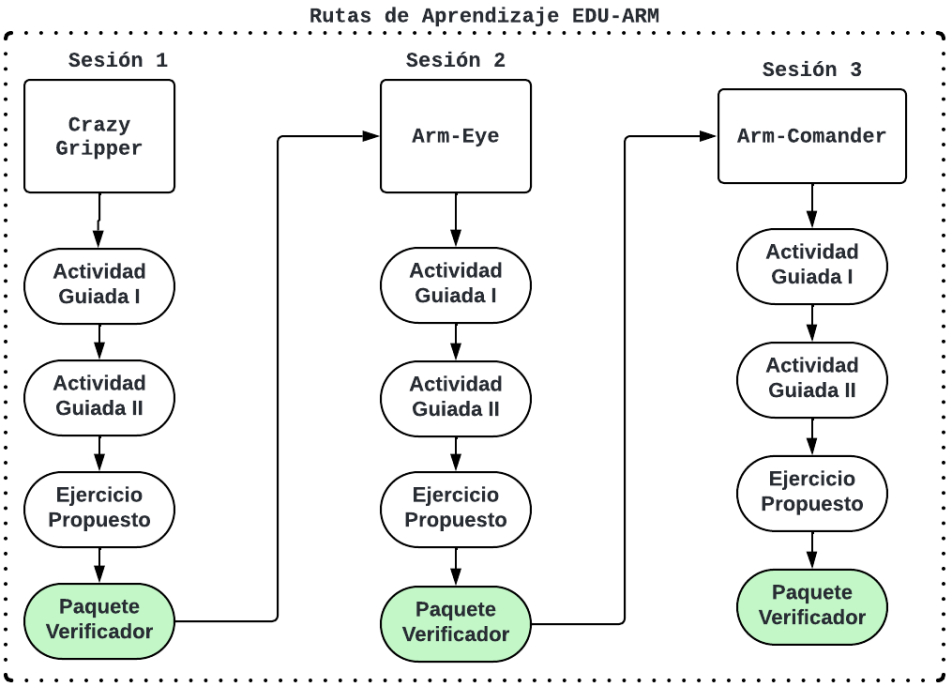


Figura 3.19 *Ruta de Aprendizaje. Fuente: Elaboración propia.*

Las actividades guiadas constituyen el material educativo proporcionado al estudiante. En estas se incluyen instrucciones detalladas que abarcan desde la instalación de los paquetes y dependencias necesarias para el robot, hasta la creación de paquetes y la comprensión de la estructura de los algoritmos aplicados en cada actividad. El objetivo de cada actividad guiada será descrito en esta misma sección, con el fin de resaltar su aporte en la construcción progresiva del conocimiento.

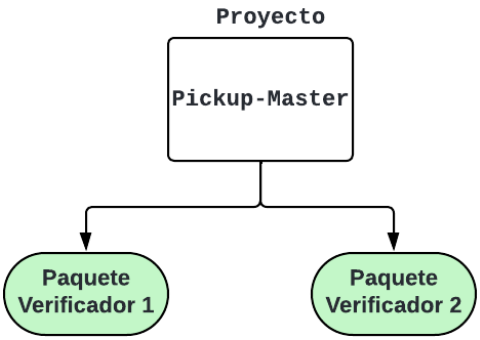


Figura 3.20 *Proyecto Pickup-Master. Fuente: Elaboración propia.*

Por su parte, el ejercicio propuesto plantea al estudiante el desafío de desarrollar un paquete propio en ROS que dé solución a los requerimientos establecidos. Una vez completado, el paquete verificador se encarga de validar automáticamente el cumplimiento de dichos requisitos. Para ello, inicializa un nodo que se suscribe a los tópicos

correspondientes, extrae la información generada, la contrasta con los valores esperados y, en caso de coincidencia total, asigna una calificación del 100%. Posteriormente, el sistema organiza los resultados en formato JSON, lo que facilita su transferencia o almacenamiento. Cabe señalar que los paquetes verificadores desarrollados para este aplicativo se encuentran disponibles en el repositorio de GitHub del proyecto, específicamente en la carpeta “Paquetes verificadores”.

Finalmente, la Figura 3.20 presenta el proyecto Pickup-Master, el cual corresponde a la fase final que complementa la ruta de aprendizaje. En esta etapa, los estudiantes pueden culminar su proceso formativo en supervisión y tele-comando del robot manipulador mediante la práctica en el proyecto final. Esta experiencia integra y consolida todos los conocimientos adquiridos a lo largo de la ruta de aprendizaje, constituyendo la aplicación práctica de los conceptos trabajados en las sesiones anteriores. El desarrollo del proyecto Pickup-Master, Crazy Gripper, Arm-Eye y Arm-Comander se explicarán en esta sección.

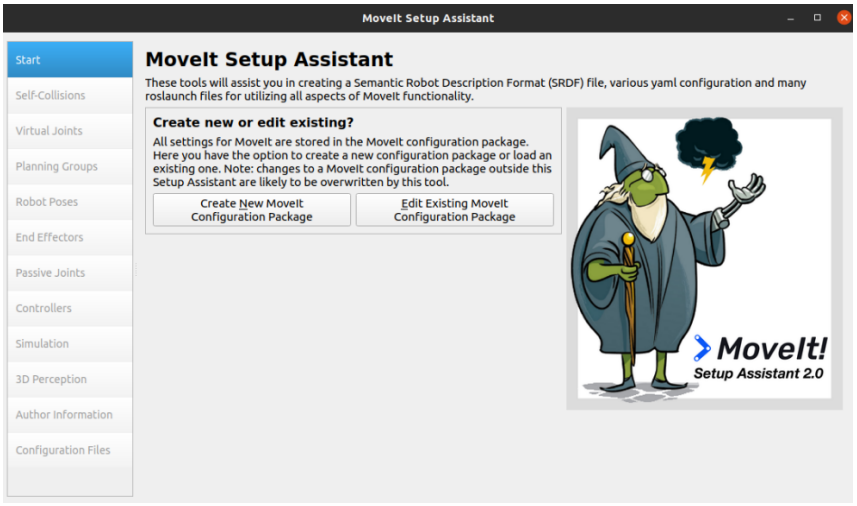


Figura 3.21 *Ventana de Asistente de Moveit. Fuente: Elaboración propia.*

3.4.3 Crazy Gripper

Esta sesión reta al estudiante para que se familiarice con los movimientos básicos del robot manipulador y comprenda cómo utilizar herramientas de planificación y control, como *MoveIt* y el controlador integrado del robot, para posicionar el efector final (gripper) en un punto específico dentro del espacio de trabajo. Esta sesión se desarrolla a través de las siguientes dos actividades guiadas (AG1, AG2) y un ejercicio propuesto (EP):

Planificación de trayectorias con MoveIt (AG1): En esta actividad, el usuario aprenderá a utilizar la herramienta MoveIt, como se observa en la Figura 3.21 para definir posiciones específicas en el espacio de trabajo del robot. El instructivo guía paso a paso al usuario en la creación de un paquete de configuración mediante el asistente de MoveIt, incluyendo la configuración de parámetros esenciales como la planificación de trayectorias, la detección de colisiones y la integración del modelo del robot.

Una vez completada esta configuración, el usuario podrá emplear MoveIt para generar trayectorias de manera automática, permitiendo que el robot manipulador se desplace de forma segura y precisa hacia las posiciones deseadas como se observa en la Figura 3.22.

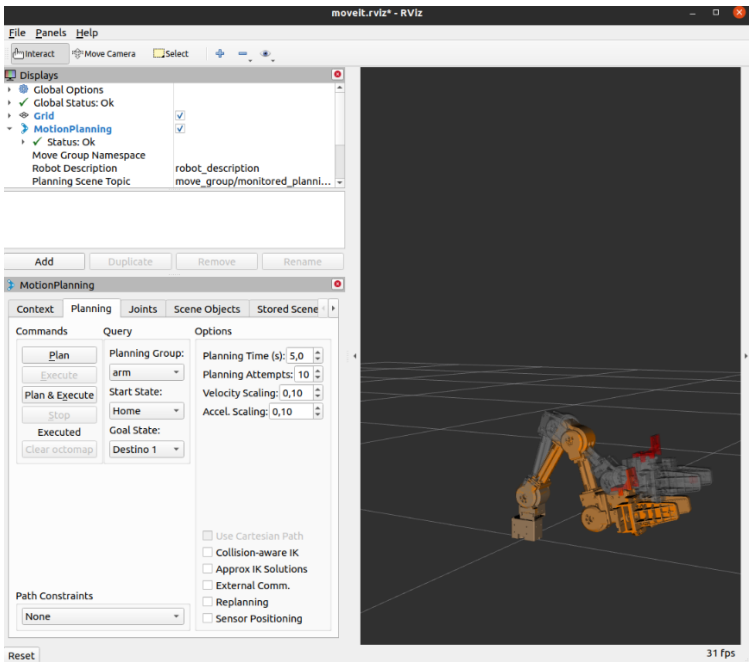


Figura 3.22 *Desplazamiento del robot a posición deseada. Fuente: Elaboración propia.*

Movimiento 1 posición definida (AG2): En esta actividad, el usuario aprenderá a mover el efector final del robot hacia una posición específica dentro del espacio cartesiano. El instructivo ofrece una guía detallada que incluye la creación del paquete necesario, la implementación de los nodos en ROS, la suscripción a los tópicos relevantes y la generación de clientes para los servicios del controlador del robot.

A lo largo de este proceso, el usuario podrá observar el comportamiento cinemático del manipulador mientras se desplaza desde su posición inicial (Figura 3.23a) hasta la posición final (Figura 3.23b), analizando la respuesta del sistema frente a los comandos definidos. De esta manera, se consolida la comprensión del control del robot en el espacio cartesiano.

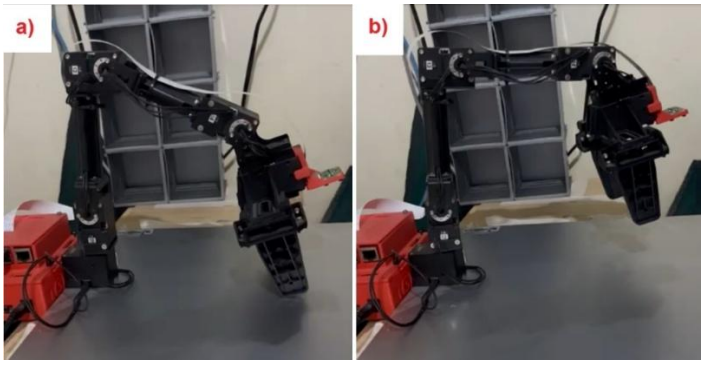


Figura 3.23 *a) Posición inicial b) posición destino. Fuente: Elaboración propia.*

Movimiento a tres posiciones definidas (EP): En esta actividad, el usuario deberá poner en práctica los conocimientos adquiridos en las actividades guiadas 1 y 2. El ejercicio propuesto reta al estudiante a emplear el controlador integrado del robot para desplazar el efector final hacia tres posiciones previamente definidas, ejecutando la secuencia de forma continua y sin interrupciones hasta completar el recorrido: primero hacia la posición 1 (Figura 3.24a), luego hacia la posición 2 (Figura 3.24b) y, finalmente, hacia la posición 3 (Figura 3.24c).

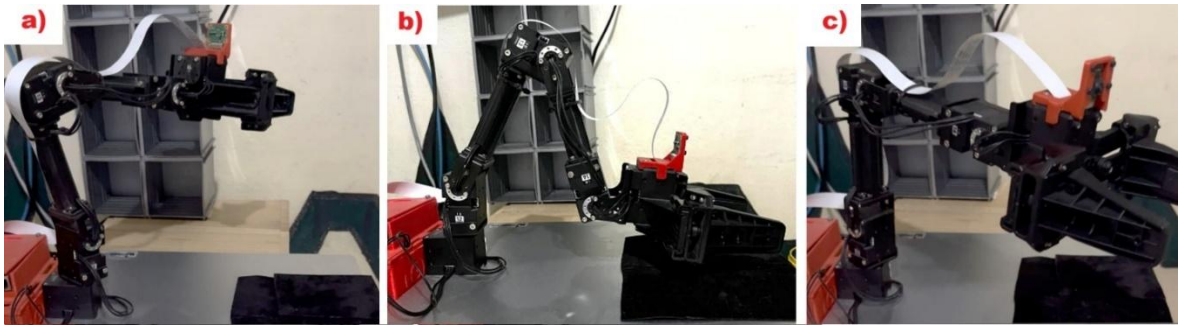


Figura 3.24 *a) Posición 1 b) Posición 2 c) Posición 3. Fuente: Elaboración propia*

3.4.4 Arm-Eye

Esta sesión reta al estudiante a que se familiarice con la cámara del robot y los algoritmos de reconocimiento de objetos por color y por forma. La finalidad de la sesión 2 es introducir al usuario en el uso del robot para el reconocimiento de objetos, utilizando tanto el método basado en color como el basado en forma. Esta sesión también cuenta con dos actividades guiadas (AG1, AG2) y un ejercicio propuesto EP:

Segmentación de objetos de color azul (AG1): En esta actividad, se guía al estudiante en la creación de un paquete en ROS para procesar las imágenes captadas por la cámara del robot. El instructivo detalla el procedimiento para suscribirse al tópico de la cámara, convertir las imágenes al formato de OpenCV mediante cv_bridge, y aplicar una máscara en el espacio de color HSV para detectar objetos de color azul. Para ello, el robot utilizará sus articulaciones para desplazarse a la posición de reconocimiento (Figura 3.25a). La imagen resultante, en la que se resalta el área correspondiente al color identificado (Figura 3.25b), se publica en un nuevo tópico para su observación en /gamora/objeto/Centroide, como se muestra en la Figura 3.26.

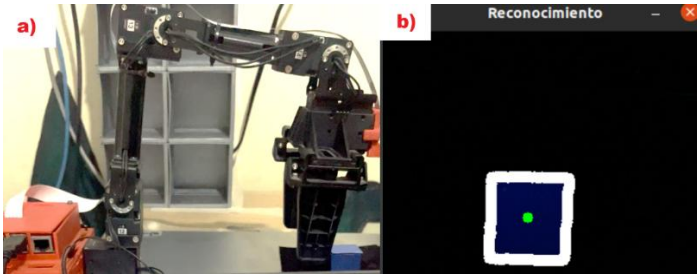


Figura 3.25 *a) Posición de reconocimiento a objeto azul. Fuente: Elaboración propia b) Reconocimiento objeto azul. Fuente: Elaboración propia*

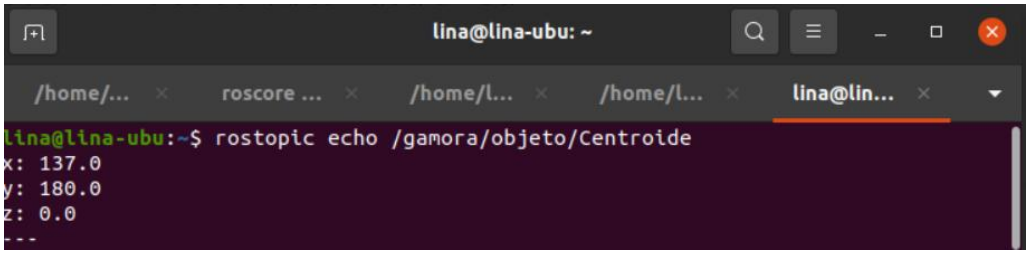


Figura 3.26 *Publicación del centroide en tópico /gamora/objeto/Centroide. Fuente: Elaboración propia*

Segmentación de objetos de forma redonda (AG2): En esta actividad, se instruye al usuario en la aplicación de técnicas de procesamiento de imágenes para la identificación automática de objetos con forma circular. El instructivo guía al usuario en la creación de un paquete en ROS y en el desarrollo de un algoritmo de segmentación que permite reconocer la forma del objeto. Al ejecutar el algoritmo, se muestra inicialmente la imagen capturada

del objeto (Figura 3.27a), seguida de la imagen procesada con el filtro Canny para la detección de bordes (Figura 3.27b). Posteriormente, se emplean funciones de la librería OpenCV para la identificación. Finalmente, se presenta el reconocimiento del objeto con su centroide dibujado (Figura 3.27c) y el centroide publicado en el tópico gamora/objeto/Centroide en la Figura 3.28.

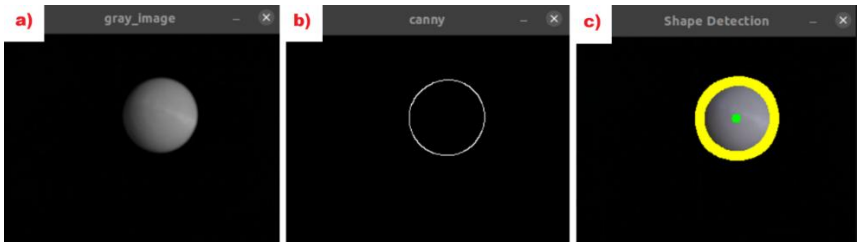


Figura 3.27 *a) Objeto en escala de grises b) Contornos del objeto c) Objeto de forma redonda reconocido. Fuente: Elaboración propia*

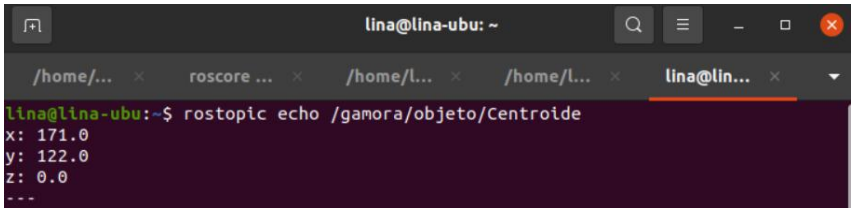


Figura 3.28 *Centroide de objeto de forma redonda en tópico /gamora/objeto/Centroide. Fuente: Elaboración propia*

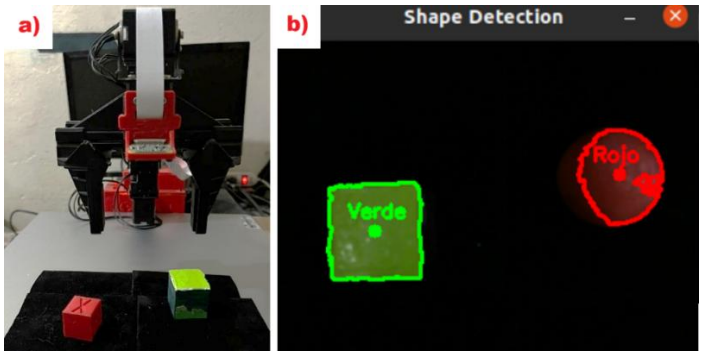


Figura 3.29 *a) Posición de reconocimiento b) Objeto verde y rojo reconocidos. Fuente: Elaboración propia*

Segmentación Objetos color ROJO y VERDE (EP): En esta actividad, el usuario se enfrentará al desafío de aplicar las técnicas de reconocimiento de imagen aprendidas previamente, implementando máscaras de filtrado para dos colores simultáneamente: rojo y verde. En primer lugar, el robot debe desplazarse a la posición de reconocimiento (Figura 3.29a) para identificar ambos objetos. Luego, el sistema debe contornearlos y dibujar sus centroides, como se muestra en la Figura 3.29b. Finalmente, las coordenadas de los centroides deben ser publicadas en los tópicos correspondientes, tal como se observa en la Figura 3.30 y Figura 3.31.

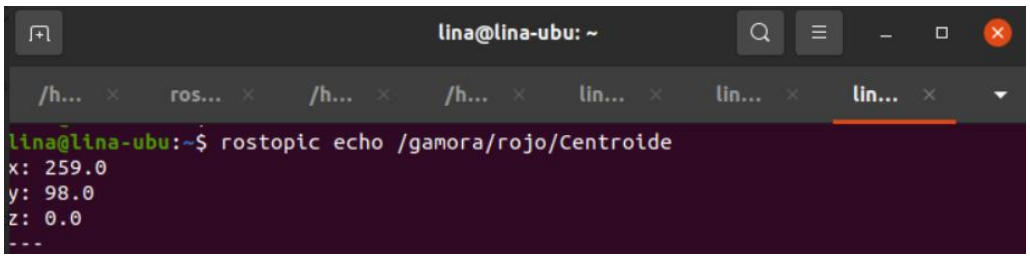


Figura 3.30 *Tópico /gamora/rojo/Centroide para objeto rojo. Fuente: Elaboración propia*

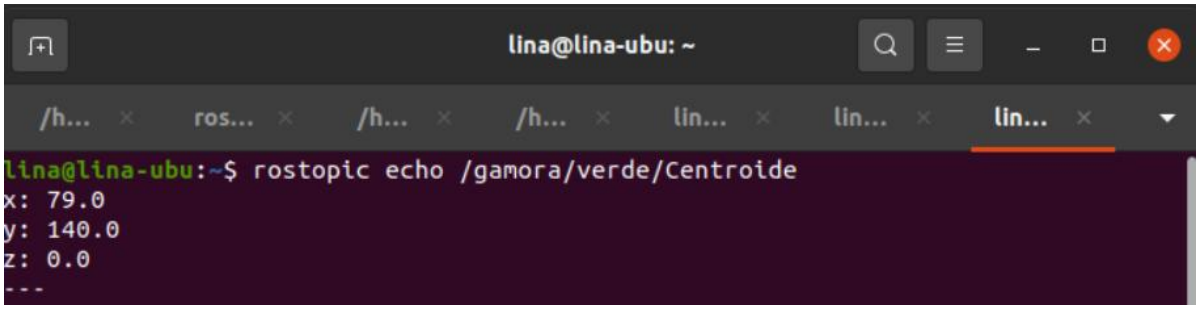


Figura 3.31 *Tópico /gamora/verde/Centroide para objeto verde. Fuente: Elaboración propia*

3.4.5 Arm-Commander

En esta sesión, se reta al estudiante a aplicar los conocimientos adquiridos en la Sesión 1 (Crazy Gripper) y la Sesión 2 (Arm-Eye). El objetivo principal consiste en mover al robot hacia una posición específica donde se encuentren los objetos deseados, los cuales pueden haber sido reconocidos por su color o por su forma, para luego transportarlos a otra ubicación o almacenarlos en un espacio de la estantería.

Esta sesión también cuenta con dos actividades guiadas (AG1, AG2) y un ejercicio propuesto EP:

Agarre manual de un objeto de color específico en una posición definida (AG1): En esta actividad, se instruye al estudiante en la creación de un paquete que permita reconocer y agarrar manualmente un objeto de color azul usando el paquete de Teleopkey. Para ello, es necesario posicionar el robot en una postura de reconocimiento (Ver Figura 3.32a), activar los algoritmos de detección y realizar el reconocimiento del objeto, como se muestra en la Figura 3.32b. Posteriormente, el estudiante utiliza el paquete TeleopKey (Figura 3.33) para mover manualmente el robot y ejecutar la acción de agarre del objeto, tal como se observa en las Figura 3.32c y Figura 3.32d.

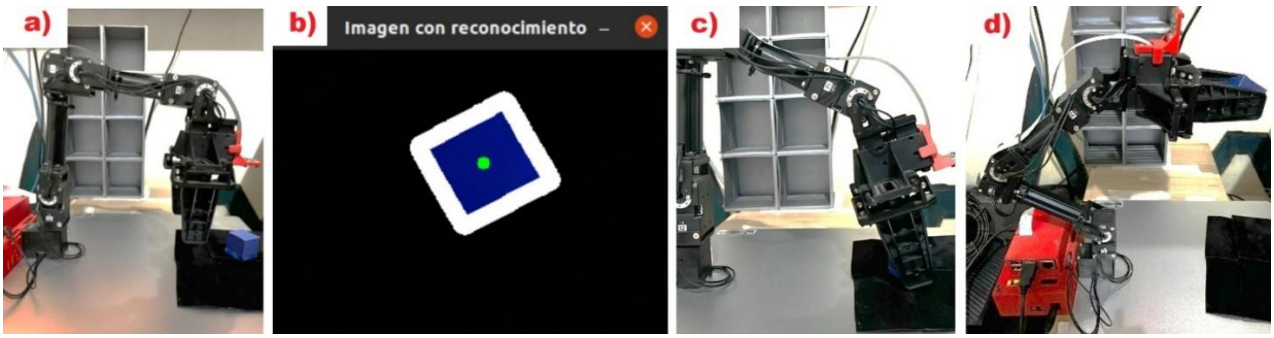


Figura 3.32 *a) Posición de reconocimiento b) Reconocimiento objeto azul. c) Agarre manual del objeto d) d) Carga manual del objeto. Fuente: Elaboración propia*

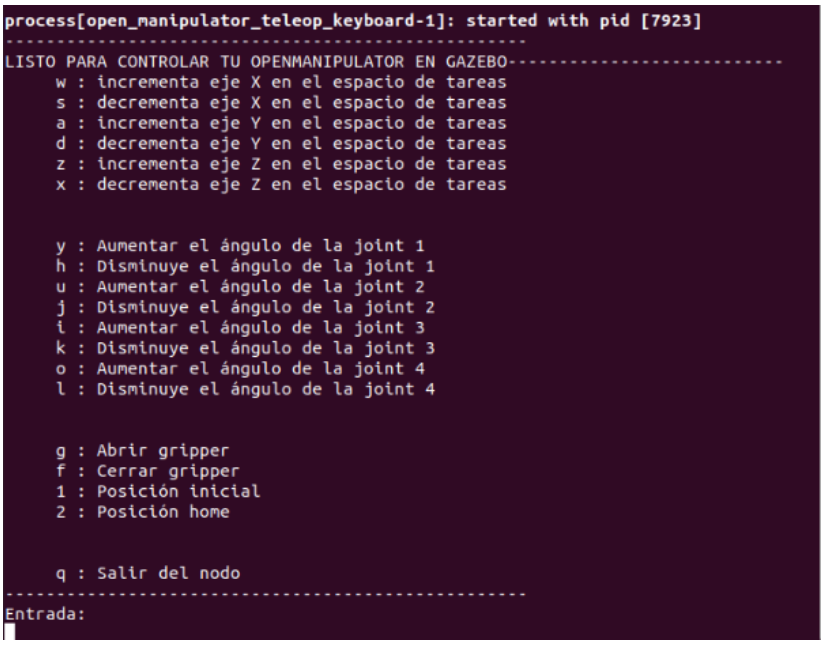


Figura 3.33 *Terminal con TeleopKey. Fuente: Elaboración propia*

Agarre automático de un objeto de color específico en una posición definida (AG2): Esta actividad guía al usuario en el desarrollo de un paquete que permita reconocer un objeto de color verde, estimar si su ubicación coincide con el punto de agarre y ejecutar automáticamente la acción de agarre. Para ello, el usuario debe desarrollar un paquete que primero posicione el robot en la postura de reconocimiento (Figura 3.34a), luego active el algoritmo de detección por color (Figura 3.34b), realice el agarre del objeto en el punto identificado y finalmente lo traslade a otra posición, como se muestra en la Figura 3.35a y Figura 3.35b.

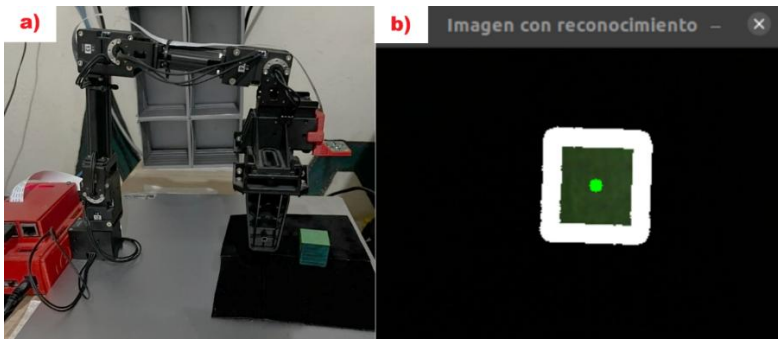


Figura 3.34 *a) Posición de reconocimiento b) Reconocimiento de objeto color verde. Fuente: Elaboración propia*

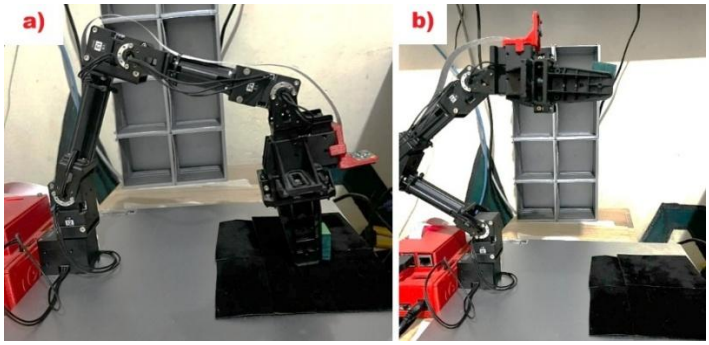


Figura 3.35 *a) Agarre objeto color verde b) Carga objeto color verde. Fuente: Elaboración propia*

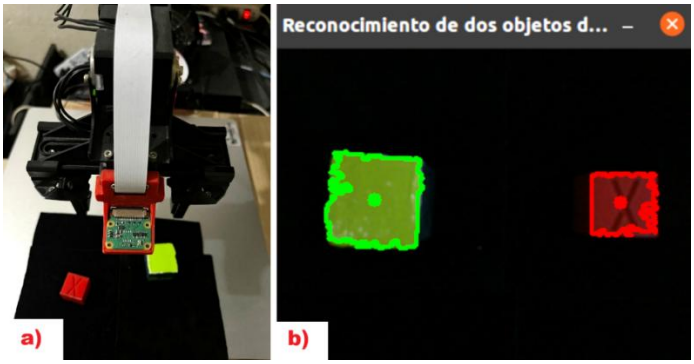


Figura 3.36 *a) Posición de reconocimiento. b) Reconocimiento objetos color verde y rojo. Fuente: Elaboración propia*

Segmentación Objetos color ROJO y VERDE (EP): En esta actividad, el estudiante enfrenta el desafío de aplicar las estrategias de agarre aprendidas, estimando la posición de los objetos reconocidos y utilizando los conocimientos adquiridos en las actividades anteriores. El usuario debe desarrollar un paquete en ROS que permita mover un objeto de color rojo y otro de color verde, trasladándolos hacia una posición definida en la estantería. Para ello, el robot primero debe ubicarse en una posición de reconocimiento (Figura 3.36a), identificar los objetos (Figura 3.36b), ejecutar el agarre del primer objeto (Figura 3.37a) y trasladarlo a la estantería (Figura 3.37b). Posteriormente, debe realizar el agarre del segundo objeto (Figura 3.38a) y desplazarlo igualmente a la posición correspondiente en la estantería (Figura 3.38b).

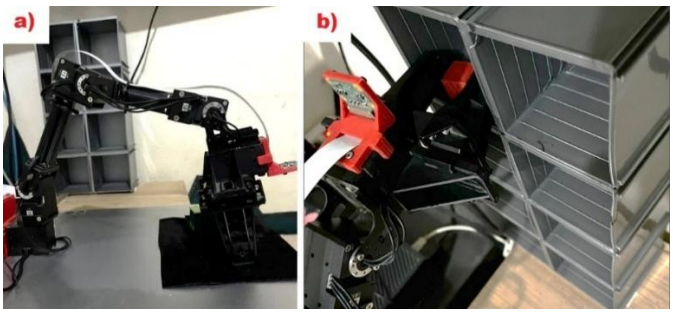


Figura 3.37 *a) Agarre objeto color rojo b) Almacenamiento en estantería objeto color rojo. Fuente: Elaboración propia*

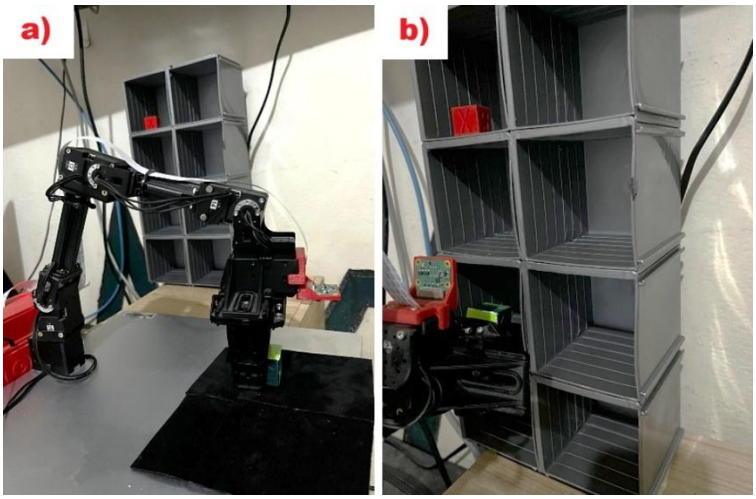


Figura 3.38 *a) Agarre objeto color verde. b) Almacenamiento en estantería objeto color verde. Fuente: Elaboración propia*

3.4.6 Paquetes verificadores

En la Figura 3.19 se explicó que cada sesión de la ruta de aprendizaje cuenta con un ejercicio propuesto. La sesión 1, denominada Crazy-Gripper, presenta su propio ejercicio; la sesión 2, denominada Arm-Eye, y la sesión 3, denominada Arm-Eye 2 también incluyen su ejercicio específico. Cada uno de estos ejercicios debe ser desarrollado por el estudiante aplicando los conocimientos adquiridos a lo largo de la ruta de aprendizaje. Los algoritmos implementados por el estudiante se evalúan mediante los paquetes verificadores los cuales necesitan dos parámetros para funcionar, estos son:

- **TOKEN:** Es el número o clave que permite a la página web recibir los datos del ejercicio y publicar la calificación correspondiente. Si el código es incorrecto, el estudiante no recibirá su calificación. En la Figura 3.39 se muestra el proceso de obtención del token.

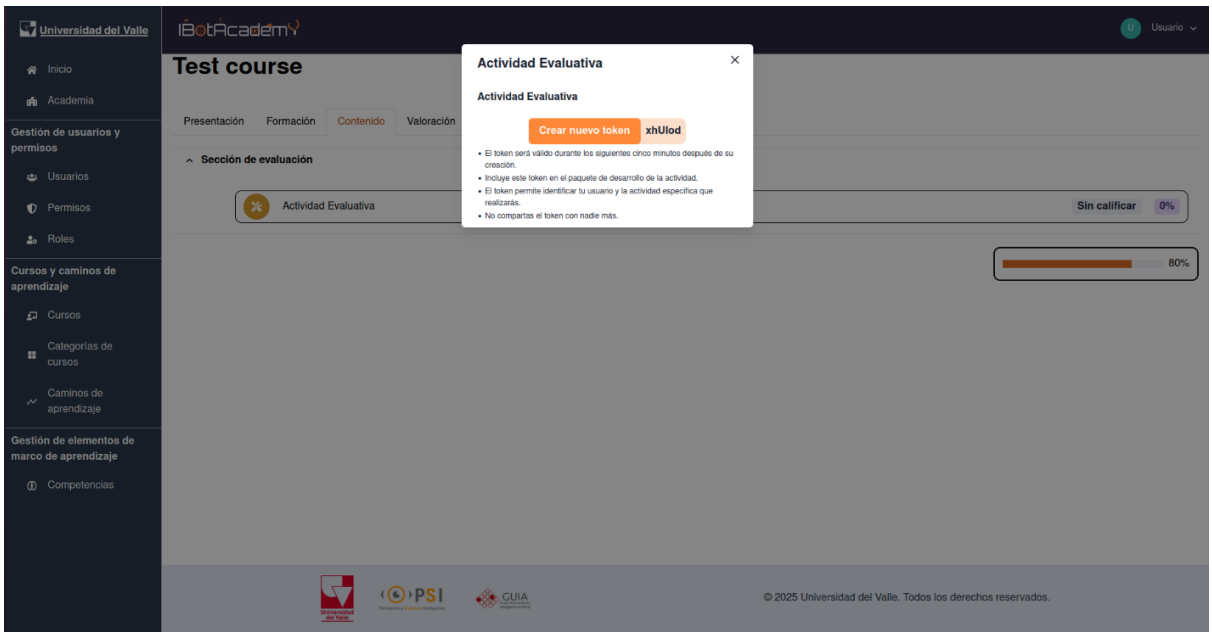


Figura 3.39 *Obtención del Token de Página web iBotAcademy [32].*

- **NAMESPACE:** Corresponde al nombre del robot que se va a utilizar. Este namespace corresponde al nombre que tendrá al inicio los tópicos a emplear. Si se emplea el robot Gamora, debe escribirse /gamora y los tópicos siempre inician con ese nombre; si se utiliza el robot nebula, debe escribirse */nebula*; y si el robot se ejecuta en Gazebo, no se debe especificar ningún *namespace*, los tópicos serán originales. Para este proyecto se explicará usando el robot gamora.

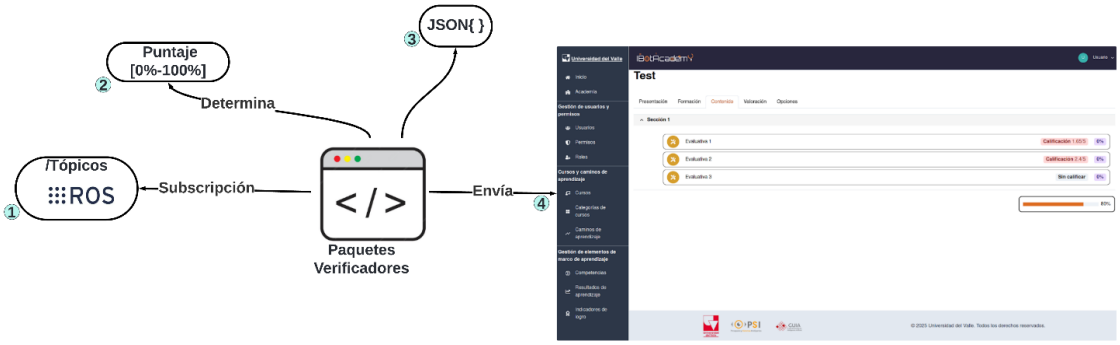


Figura 3.40 Funcionamiento procesal de paquete verificador. Fuente: Elaboración propia

Tal como se muestra en el diagrama conceptual de la Figura 3.2, el paquete verificador tiene como propósito evaluar el trabajo desarrollado por el estudiante a partir del desempeño del robot durante la ejecución del ejercicio. En Figura 3.40 se presenta el funcionamiento general de un paquete verificador: primero, el paquete se suscribe a los tópicos de ROS para obtener información relevante sobre el comportamiento del robot, según los requerimientos establecidos en el ejercicio. En segundo lugar, con base en esta información, el sistema determina una calificación entre 0 y 100%, la organiza en formato JSON y, finalmente, envía los resultados a la plataforma web iBotAcademy mediante el protocolo HTTP.

Cada paquete verificador correspondiente a las distintas sesiones se explica en detalle en esta sección.

3.4.6.1 Paquete verificador de la Sesión 1

```
01 INICIAR clase Verificador_Paquetes_ActividadGuiada1
02    FUNCIÓN __init__():
03       INICIAR nodo ROS "verificador_ejercicio_propuesto_sesion1"
04       OBTENER parámetros:
05          - namespace
06          - token
07       DEFINIR nombre del paquete a evaluar
08       INICIALIZAR calificación_total = 0
09       // Ejecutar verificaciones
10      cal1 ← verificar_existencia_paquete()
11      cal2 ← verificar_directorio_launch()
12      cal3 ← verificar_archivos_launch_y_scripts()
13      cal4 ← verificar_permisos_ejecucion()
cal5 ← verificar_servicio_definido_en_ROS()
15      14      cal6 ← verificar_servicio_corriendo()
16.    cal7 ←verificar_mensaje_personalizado_servicio()
17       // Sumar calificaciones parciales
18       calificación_total ← cal1 + cal2 + cal3 + cal4 + cal5 + cal6 + cal7
19       MOSTRAR calificación_total
20       // Construir estructura de datos
21       CONSTRUIR estructura_JSON con resultados de cada prueba
22       ENVIAR resultados al servidor web mediante petición HTTP PUT
23       MOSTRAR respuesta del servidor
24       IMPRIMIR "FINALIZADO"
25 PROGRAMA PRINCIPAL:

26    CREAR objeto Verificador_Paquetes_ActividadGuiada1
27    MANTENER nodo activo con rospy.spin()
28 FUNCIÓN verificar_existencia_paquete():
29    Buscar paquete con "rospack find"
30    Verificar existencia de archivos:
31       - package.xml
32       - CMakeLists.txt
33    Revisar dependencias:
34       - message_generation
35       - rospy
36       - std_msgs
37    SI todo correcto → RETORNAR 7
38    SINO → RETORNAR 0

39 FUNCIÓN verificar_directorio_launch():
40    Buscar carpeta "launch" dentro del paquete
41    SI existe → RETORNAR 7
42    SINO → RETORNAR 0

43 FUNCIÓN verificar_archivos_launch_y_scripts():
44    Buscar archivos:
45       - server_manipulator_launch.launch
46       - go_location_server.py
47       - go_location_client.py
48    SI todos existen → RETORNAR 7
49    SINO → RETORNAR 0

50. FUNCIÓN verificar_permisos_ejecucion():
51.    Revisar si archivos ".launch" y ".py" tienen permisos de ejecución
52.    SI todos ejecutables → RETORNAR 7
53.    SINO → RETORNAR 0
54. FUNCIÓN verificar_servicio_definido_en_ROS():
55.    Revisar carpeta "srv" del paquete
56.    Buscar archivo "go_location_manipulator.srv"
57.    SI existe → RETORNAR 7
58.    SINO → RETORNAR 0
59. FUNCIÓN verificar_servicio_corriendo():
60.    Buscar "go_location_manipulator" en lista de servicios activos
61.    SI existe → RETORNAR 14
62.    SINO → RETORNAR 0
63. FUNCIÓN verificar_mensaje_personalizado_servicio():
64.    Inspeccionar definición del servicio con "rossrv show"
65.    Verificar que contenga:
66.       - entrada float64[4] kinematics_pose1,2,3
67.       - entrada float64 path_time
68.       - separador ---
69.       - salida bool ok
70.    SI cumple → RETORNAR 51
71.    SINO → RETORNAR 0
72. FIN clase
```

Algoritmo 3.2 *Paquete Verificador Sesión 1. Fuente: Elaboración propia*

El paquete verificador de la sesión 1 tiene como objetivo comprobar si el estudiante cumplió con los requisitos establecidos en el ejercicio propuesto de la sesión *Crazy-Gripper*, descrito en la Sección 3.4.3. el cual consiste en que el robot logre alcanzar correctamente las tres posiciones cartesianas definidas en el ejercicio, de acuerdo con los parámetros y condiciones establecidas. Para llevar a cabo esta verificación se lanza el paquete usando el siguiente comando:

$ roslaunch verificador_ejercicio_propuesto_sesion1 verificador_ejercicio_propuesto_sesion1.launch namespace:=/gamora token:=123456

Una vez ejecutado éste realizará las siguientes verificaciones:

1. Creación del paquete, (7 %)
2. Existencia del directorio launch, (7 %)
3. Existencia del archivo launch, existencia del script de Python, (7 %)
4. Permisos de ejecución del script de Python, (7 %)
5. Existencia de la acción en ROS (7 %)
6. Existencia del servicio /go_location (14 %).
7. Existencia del mensaje personalizado de la acción usando coordenadas XY expresadas en flotantes (51 %).

En el Algoritmo 3.2 se muestra el pseudocódigo de la estructura del paquete verificador, el cual verifica la existencia del paquete mediante el comando rospack find, asegurando que incluya los archivos esenciales *package.xml* y *CMakeLists.txt*, así como las dependencias requeridas (*message_generation*, *rospy* y *std_msgs*). Luego, valida que el paquete contenga la carpeta *launch* y los archivos correspondientes a los scripts del servidor y cliente (*server_manipulator_launch.launch*, *go_location_server.py* y *go_location_client.py*), además de comprobar que todos estos posean permisos de ejecución. Posteriormente, el verificador inspecciona la definición del servicio en ROS dentro del directorio *srv* y verifica que el servicio *go_location_manipulator* esté activo y correctamente configurado. Finalmente, analiza el mensaje personalizado del servicio, comprobando que incluya los tipos de datos, entradas y salidas esperadas según el formato definido, como se observa en la Figura 3.41.

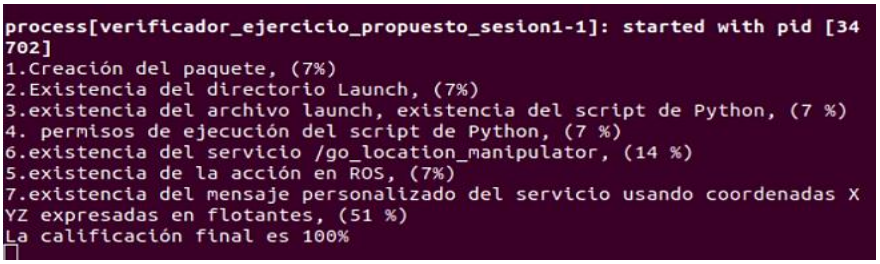


Figura 3.41 *Ejecución del paquete verificador sesión 1. Fuente: Elaboración propia*

3.4.6.2 Paquete verificador de la Sesión 2

El paquete verificador de la sesión 2 tiene como objetivo comprobar si el estudiante cumplió con los requisitos establecidos en el ejercicio propuesto de la sesión *Arm-Eye*, descrito en la Sección 3.4.4. el cual consiste en que el robot logre reconocer un objeto de color rojo y un objeto de color verde y que su centroide sea publicado en el tópico */objeto/Centroide*, de acuerdo con los parámetros y condiciones establecidas. Para llevar a cabo esta verificación se lanza el paquete usando el siguiente comando:

$ roslaunch verificador_ejercicio_propuesto_sesion2
verificador_ejercicio_propuesto_sesion2.launch namespace:=/gamora token:=112233

Se verificará que el robot segmenta los objetos del color especificado que se ubican en una posición definida del campo de visión de la cámara usando un paquete de evaluación desarrollado para tal fin, el cual medirá:

1. Creación del paquete, (7 %)
2. Existencia del directorio launch, (7 %)
3. Existencia del archivo launch, existencia del script de Python, (7 %)
4. Permisos de ejecución del script de Python, (7 %)
5. Existencia del tópico /objeto/Centroide. (7 %)
6. En el momento de ubicar un objeto rojo en la parte derecha del FOV de la cámara se detectará su posición a través del tópico (40 %).
7. En el momento de ubicar un objeto verde en la parte izquierda del FOV de la cámara se detectará su posición a través del tópico (25 %).

El funcionamiento general del programa puede representarse de forma algorítmica como se muestra en el Algoritmo 3.3. A diferencia de la sesión 1, donde el verificador se limitaba a comprobar la estructura del paquete, las dependencias y el funcionamiento básico del servicio, en la sesión 2 se amplía la verificación incorporando la detección de tópicos relacionados con el seguimiento visual de objetos y su correcta publicación dentro del

entorno ROS. En esta nueva versión, el nodo del paquete verificador realiza suscripciones y validaciones dinámicas de los mensajes publicados en los tópicos /objeto/Centroide, /rojo/Centroide y /verde/Centroide, comprobando que el robot responda adecuadamente a las posiciones de los objetos en el campo visual, como se observa en la Figura 3.42.

```
01 INICIAR clase Verificador_Ejercicio_Propuesto_SESION2
02 FUNCIÓN __init__():
03     INICIAR nodo ROS "verificador_ejercicio_propuesto_sesion2"
04     OBTENER parámetros:
05         - namespace
06         - token
07     DEFINIR nombre del paquete a evaluar
08     INICIALIZAR calificación_total = 0
09     // Verificaciones básicas heredadas de la sesión 1
10     cal1 ← verificar_existencia_paquete()
11     cal2 ← verificar_directorio_launch()
12     cal3 ← verificar_archivos_launch_y_scripts()
13     cal4 ← verificar_permisos_ejecucion()
14     cal5 ← verificar_servicio_definido_en_ROS()
15     cal6 ← verificar_servicio_corriendo()
16     cal7 ← verificar_mensaje_personalizado_servicio()
17     Enviar los datos al servidor mediante solicitud HTTP PUT (método requests.put)
18     Mostrar código de respuesta y contenido recibido del servidor
19     Imprimir "FINALIZADO"
20 EXCEPTO en caso de error:
21     Mostrar "Error en la inicialización"
22     Finalizar ejecución con código de error
23 FIN DEL MÉTODO __init__()__init__()
24 // Verificaciones adicionales propias de la sesión 2
25 cal8 ← existencia_topico_objeto_centroide()
26 cal9 ← solicitud_prueba_objeto_rojo_derecha()
27 cal10 ← solicitud_prueba_objeto_verde_izquierda()
28 // Calcular resultado total
29 calificación_total ← cal1 + cal2 + cal3 + cal4 + cal5 + cal6 + cal7 + cal8 + cal9 + cal10
30 // Construcción de resultados
31 CONSTRUIR estructura_JSON con resultados individuales
32 ENVIAR datos al servidor mediante protocolo HTTP PUT
33 MOSTRAR respuesta del servidor
34 IMPRIMIR "FINALIZADO"
35 FIN FUNCIÓN
36 FUNCIÓN existencia_topico_objeto_centroide():
37     COMPROBAR si el tópico /namespace/objeto/Centroide está publicado
38     SI está activo Y transmite información válida → RETORNAR 7
39     SINO → RETORNAR 0
40 FUNCIÓN solicitud_prueba_objeto_rojo_derecha():
41     SOLICITAR al usuario ubicar el objeto rojo en el lado derecho del campo visual
42     SUSCRIBIRSE al tópico /rojo/Centroide
43     LEER coordenadas publicadas
44     VALIDAR que la posición en X sea mayor a 160
45     SI cumple → RETORNAR 7
46     SINO → RETORNAR 0
47 FUNCIÓN solicitud_prueba_objeto_verde_izquierda():
48     SOLICITAR al usuario ubicar el objeto verde en el lado izquierdo del campo visual
49     SUSCRIBIRSE al tópico /verde/Centroide
50     LEER coordenadas publicadas
51     VALIDAR que la posición en X sea menor a 160
52     SI cumple → RETORNAR 7
53     SINO → RETORNAR 0
54 FUNCIÓN main():
55     INICIAR nodo ROS
56     CREAR instancia del objeto Verificador_Ejercicio_Propuesto_SESION2()
57     MANTENER nodo activo con rospy.spin()
58 FIN DEL MÉTODO main()
59 SI el script se ejecuta directamente:
60     LLAMAR a main()
```

Algoritmo 3.3 *Paquete verificador sesión 2. Fuente: Elaboración propia*

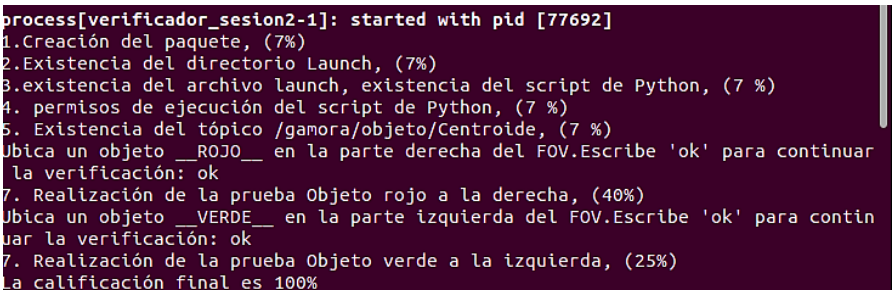


Figura 3.42 *Ejecución de paquete verificador sesión 2. Fuente: Elaboración propia*

3.4.6.3 Paquete verificador de la Sesión 3

El paquete verificador de la sesión 3 tiene como objetivo comprobar si el estudiante cumplió con los requisitos establecidos en el ejercicio propuesto de la sesión *Arm-Commander*, descrito en la Sección 3.4.5. el cual consiste en que el robot logre que su efector final se mueva a dos posiciones donde se ubican objetos de distinto color o forma, y luego trasladarlos a una posición en la estantería de almacenamiento, de acuerdo con los parámetros y condiciones establecidas. Para llevar a cabo esta verificación se lanza el paquete usando el siguiente comando:

$ roslaunch verificador_ejercicio_propuesto_sesion3 verificador_ejercicio_propuesto_sesion3.launch namespace:=/gamora token:=111222

Una vez ejecutado éste realizará las siguientes verificaciones:

1. Creación del paquete, (7 %)
2. Existencia del directorio launch, (7 %)
3. Existencia del archivo launch, existencia del script de Python, (7 %)
4. Permisos de ejecución del script de Python, (7 %)
5. Se verifica que los objetos se trasladan a coordenadas del espacio de trabajo previamente definidas. Esto se hará monitoreando la posición del efector final (72%).

El funcionamiento general del paquete puede representarse de forma algorítmica como se muestra en el Algoritmo 3.4. En esta tercera sesión, el paquete verificador hereda las comprobaciones básicas establecidas en el Algoritmo 3.2 de la sesión 1 de la ruta de aprendizaje, las cuales incluyen la validación de la estructura del paquete, las dependencias necesarias y la correcta ejecución del servicio principal. Sin embargo, el alcance de la verificación se amplía para evaluar el desempeño del robot en una tarea de manipulación más compleja.

En esta versión, el nodo del paquete verificador realiza suscripciones a los tópicos /joint_states y /gripper/state, así como al tópico /verificar_posicion, desde los cuales obtiene información sobre las posiciones articulares, el estado del efector final y la confirmación de los movimientos ejecutados. A partir de estos datos, el sistema valida que el robot manipule correctamente los objetos detectados y los traslade a las posiciones asignadas en la estantería, verificando el cumplimiento de los requisitos establecidos en la sesión Arm-Commander, como se observa en la Figura 3.43.

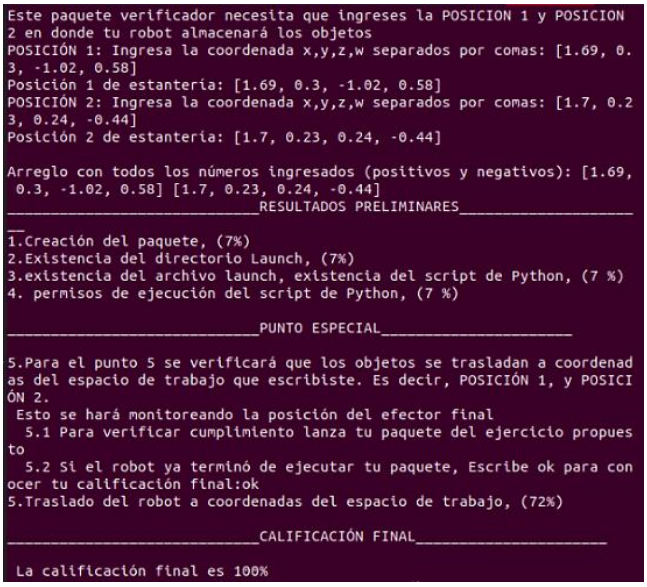


Figura 3.43 *Ejecución Paquete verificador sesión 3. Fuente: Elaboración propia*

```
01 INICIAR CLASE Verificador_Ejercicio_Sesion3
02 FUNCIÓN __init__():
03   INICIAR nodo ROS "verificador_ejercicio_propuesto_sesion3"
04   LEER parámetros:
05     - namespace
06     - token
07   DEFINIR url de servidor de calificación
08   INICIALIZAR variables de cumplimiento:
09     - cumplimiento_pos1, cumplimiento_pos2 ← None
10   INICIALIZAR listas:
11     - numeros_guardados1 ← []
12     - numeros_guardados2 ← []
13   MOSTRAR mensaje de bienvenida
14   // 1. INGRESO DE COORDENADAS
15   MIENTRAS True HACER
16     PEDIR al usuario: "Ingresa coordenadas x,y,z,w para POSICIÓN 1"
17     LIMPIAR caracteres [ ] o ( )
18     DIVIDIR respuesta por comas
19     SI cantidad ≠ 4 → MOSTRAR error y repetir
20     CONVERTIR a flotantes
21     GUARDAR en arreglo1
22     MOSTRAR posición 1 ingresada
23     SALIR del bucle
24   MIENTRAS True HACER
25     PEDIR al usuario: "Ingresa coordenadas x,y,z,w para POSICIÓN 2"
26     LIMPIAR caracteres [ ] o ( )
27     DIVIDIR respuesta por comas
28     SI cantidad ≠ 4 → MOSTRAR error y repetir
29     CONVERTIR a flotantes
30     GUARDAR en arreglo2
31     MOSTRAR posición 2 ingresada
32     SALIR del bucle
33   MOSTRAR los dos arreglos ingresados
34 // 2. SUSCRIPCIONES
35 SUSCRIBIRSE a "/joint_states" con callbacks:
36   - verificacion_pos1
37   - verificacion_pos2
38 DEFINIR nombre_paquete ← "tracking_dos_objetos"
39 ASIGNAR position_deseada1 ← arreglo1
40 ASIGNAR position_deseada2 ← arreglo2
41 Verificaciones básicas heredadas de la sesión 1
42 cal1 ← existencia_paquete()
43 cal2 ← existencia_directorio_launch()
44 cal3 ← existencia_archivo_launch_py()
45 cal4 ← permisos_ejecucion_py()
46 // 4. VERIFICACIÓN FUNCIONAL (TRASLADO)
47 MOSTRAR instrucciones sobre cómo lanzar el paquete
48 ESPERAR a que el usuario escriba "ok"
49 cal5 ← cumplimiento_traslado_objetos()
50 // 5. CALIFICACIÓN FINAL
51 PAQUETE ← cal1 + cal2 + cal3 + cal4 + cal5
52 MOSTRAR calificación final
53 ENVIAR resultados al servidor mediante solicitud HTTP PUT
54 FIN FUNCIÓN
55 FUNCIÓN verificacion_pos1(msg):
56   OBTENER posiciones actuales de articulaciones
57   ASIGNAR a temp_angle[joint1..4]
58   DEFINIR tolerancia ← 0.05
59   SI todas las diferencias |temp_angle[i] - numeros_guardados1[i]| ≤ tolerancia
60     cumplimiento_pos1 ← True
61   GUARDAR temp_angle como present_joint_angle
62 FIN FUNCIÓN
63 FUNCIÓN verificacion_pos2(msg):
64   OBTENER posiciones actuales de articulaciones
65   ASIGNAR a temp_angle[joint1..4]
66   DEFINIR tolerancia ← 0.05
67   SI todas las diferencias |temp_angle[i] - numeros_guardados2[i]| ≤ tolerancia
68     cumplimiento_pos2 ← True
69   GUARDAR temp_angle como present_joint_angle
70 FIN FUNCIÓN

71 FUNCIÓN cumplimiento_traslado_objetos():
72   SI cumplimiento_pos1 == True Y cumplimiento_pos2 == True
73     cal_traslado ← 72
74     MOSTRAR "Traslado correcto (72%)"
75   SINO
76     cal_traslado ← 0
77     MOSTRAR "Traslado no verificado (0%)"
78   RETORNAR cal_traslado
79 FIN FUNCIÓN

80 PROGRAMA PRINCIPAL:
81   CREAR objeto Verificador_Ejercicio_Sesion3
82   MANTENER nodo activo con rospy.spin()
83 FIN PROGRAMA
```

Algoritmo 3.4 *Paquete verificador sesión 3. Fuente: Elaboración propia*

3.4.7 Proyecto Pickup-Máster

En el marco del Ambiente de Aprendizaje para la Industria 4.0, surge la necesidad de contar con un entorno funcional que permita a los estudiantes interactuar con sistemas robóticos reales y comprender su integración en procesos automatizados. Con el fin de responder a esta necesidad, el proyecto Pickup-Máster se desarrolló con el objetivo de automatizar actividades de almacenamiento, transporte, carga y descarga de bienes. Para su implementación se utilizó una acción de ROS (Robot Operating System), la cual permite establecer una comunicación bidireccional entre el servidor y el cliente, además de ofrecer retroalimentación constante sobre el avance de la tarea. En este esquema, el cliente envía la solicitud de búsqueda y manipulación de un objeto definido por su forma, color o ambos, y el servidor responde ejecutando la acción encomendada. El paquete se ejecuta primero el servidor con el siguiente comando:

$ roslaunch action_carga_bienes_manipulador server_carga_bienes.launch namespace:=/gamora

Luego el cliente con el siguiente comando:

$ roslaunch action_carga_bienes_manipulador client_carga_bienes.launch namespace:=/gamora

Una vez ejecutada la acción, como se muestra en la Figura 3.44, el nodo del servidor se suscribe a los tópicos /states, /gripper/kinematics_pose y /joint_states, además de actuar como cliente de los servicios /set_actuator_state, /goal_joint_space_path, /goal_task_space_path y /goal_tool_control del controlador, con el fin de gestionar los movimientos del robot. Asimismo, el servidor se suscribe al tópico /usb_cam/image_raw para recibir y procesar la información proveniente de la cámara según las necesidades de la tarea. La acción implementada habilita los tópicos /carga_bienes/goal, /carga_bienes/result, /carga_bienes/cancel, /carga_bienes/status y /carga_bienes/feedback, a los cuales se suscribe el cliente para recibir actualizaciones y retroalimentación sobre el progreso y los resultados de la acción ejecutada.

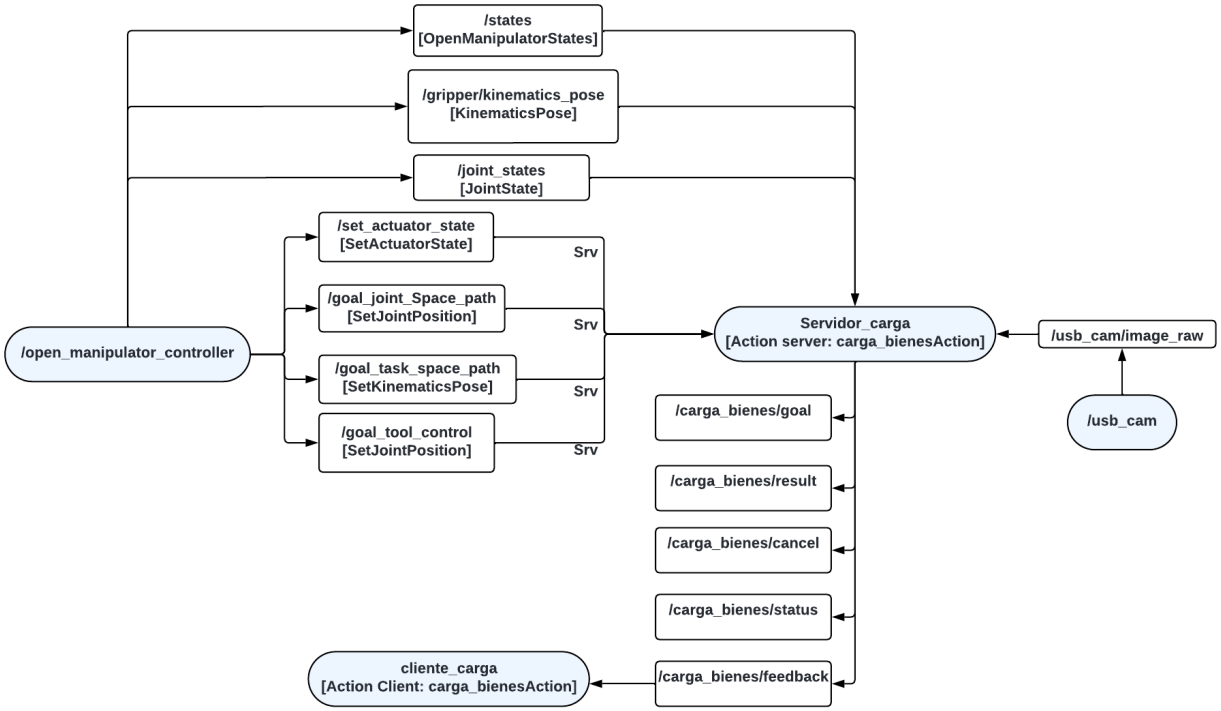


Figura 3.44 *Diagrama de nodos Proyecto Pickup-Master. Fuente: Elaboración propia*

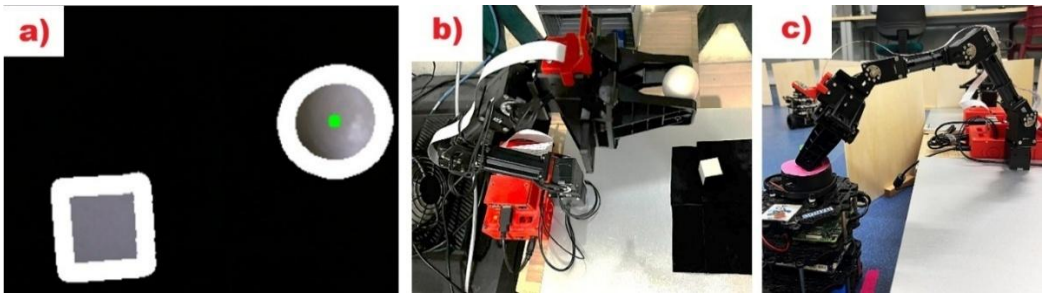


Figura 3.45 *a) Reconocimiento objeto forma redonda. b) Carga objeto c) Descarga objeto en base del turtlebot. Fuente: Elaboración propia*

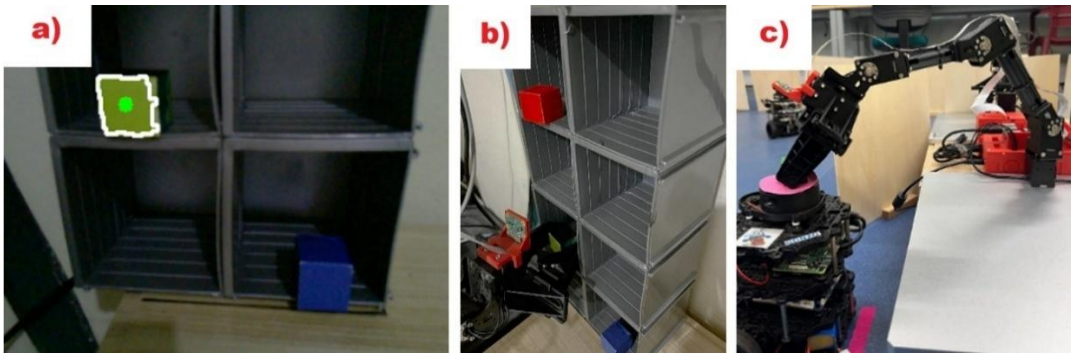


Figura 3.46 *a) Reconocimiento objeto verde. b) Carga objeto verde desde estantería. c) Descarga objeto verde en base del turtlebot. Fuente: Elaboración propia*

En el caso de una solicitud que involucre un objeto de forma redonda y otro de color verde, el robot manipulador inicia su proceso de búsqueda en función de los parámetros solicitados por el usuario. Primero, el sistema ejecuta el Algoritmo 3.6 de reconocimiento por forma. El robot se desplaza a su posición de reconocimiento (ver Figura 3.45a), analiza la imagen capturada por la cámara y detecta el objeto con la forma solicitada en este caso, redonda. Una vez identificado, calcula el centroide del contorno detectado y, según su ubicación en el eje X, selecciona la estrategia de agarre correspondiente: si $X < 160$ se dirige a la posición Op1, y si $X \geq 160$ se desplaza a la posición Op2. Luego, el robot ejecuta su rutina de agarre (ver Figura 3.45b) y deposita el objeto en la base de la estantería (ver Figura 3.45c). Posteriormente, el sistema ejecuta el algoritmo de reconocimiento por color, descrito en el Algoritmo 3.5.

El robot recorre las distintas posiciones de la estantería, analizando con su cámara si el color indicado en este caso, verde, se encuentra en alguna de ellas (ver Figura 3.46a). Si el sistema de visión detecta el color, calcula el centro del objeto, determina su ubicación y posiciona el efector final frente a él para realizar la rutina de agarre (ver Figura 3.46b). Finalmente, deposita el objeto en la base del robot móvil TurtleBot, junto con el objeto previamente manipulado (ver Figura 3.46c). Si el robot no encuentra el color solicitado tras revisar todas las posiciones, el sistema informa al usuario que el objeto no fue localizado y el manipulador retorna a su posición inicial.

```
01 INICIO
02     // 1. Recepción de la solicitud
03     RECIBIR solicitud del cliente (color_objetivo)
04     // 2. Inicialización
05     DEFINIR vistas_estanteria = [S1, S2]
06     DEFINIR posiciones_internas = [Pos1, Pos2, Pos3, Pos4]
07     DEFINIR resolucion_camara = (320, 240)
08     color_detectado ← FALSO
09     // 3. Búsqueda del objeto en S1
10     PARA cada posicion_estante EN posiciones_estanteria HACER
11         MOVER robot a posicion_estante
12         ACTIVAR algoritmo_reconocimiento_color(color_objetivo)
13         SI objeto_detectado == VERDADERO ENTONCES
14             OBTENER centroide (x, y)
15             // Clasificación de posición según coordenadas
16             SI x < 160 Y y < 120 ENTONCES
17                 posicion ← Pos1
18             SINO SI x ≥ 160 Y y < 120 ENTONCES
19                 posicion ← Pos2
20             SINO SI x < 160 Y y ≥ 120 ENTONCES
21                 posicion ← Pos3
22             SINO
23                 posicion ← Pos4
24             FIN SI
25             color_detectado ← VERDADERO
26             SALIR del bucle
27         FIN SI
29     // 4. Ejecución de la rutina de agarre o retorno
30     SI color_detectado == VERDADERO ENTONCES
31         MOVER robot a posicion
32         EJECUTAR rutina_agarre()
33         MOVER robot a posicion_descarga_TurtleBot
34         SOLTAR objeto
35     SINO
36         MOSTRAR "Objeto no encontrado"
37         MOVER robot a posicion_inicial
38     FIN SI
39 FIN
```

Algoritmo 3.5 *Algoritmo búsqueda por color en Pickup-master. Fuente: Elaboración propia.*

```
01 INICIO
02     A. ESPERAR solicitud del cliente (forma_objetivo)
03     B. MOVER robot a posición de reconocimiento
04         MOSTRAR "Robot en posición de reconocimiento"
05     C. CAPTURAR imagen desde la cámara
06         CONVERTIR imagen a escala de grises
07         APLICAR umbral o detección de bordes
08     D. DETECTAR contornos de los objetos
09         PARA cada contorno detectado HACER
10             CALCULAR área y perímetro
11             DETERMINAR forma geométrica (círculo, cuadrado, triángulo, etc.)
12             SI forma == forma  obietivo
ENTONCES
13                 CALCULAR centroide (x, y)
14                 DIBUJAR centroide en la imagen
15                 GUARDAR coordenadas (x, y)
16                 MOSTRAR "Objeto detectado con centroide en (x, y)"
17                 SALIR del ciclo
18             FIN SI
19         FIN PARA
20 FIN
21     E. SI no se detectó ningún objeto con forma_objetivo
22         MOSTRAR "Forma no encontrada"
23         TERMINAR proceso
24     FIN SI
25     F. COMPARAR coordenada x del centroide
26     SI x < 160 ENTONCES
27         posición = "Op1"
28     SINO
29         posición = "Op2"
30     FIN SI
31     MOSTRAR "Centroide en zona:", posición
32     G. ENVIAR comando al robot para MOVER EFECTOR FINAL a posición (Op1 u Op2)
33     EJECUTAR rutina de agarre del objeto
34     H. CONFIRMAR agarre exitoso
35     SI agarre exitoso == True
36         MOSTRAR "Objeto de forma", forma_objetivo, "agarrado con éxito"
37     SINO
38         MOSTRAR "Error en agarre"
39     FIN SI
40 FIN
```

Algoritmo 3.6 *Algoritmo búsqueda por forma en Pickup-master. Fuente: Elaboración propia.*

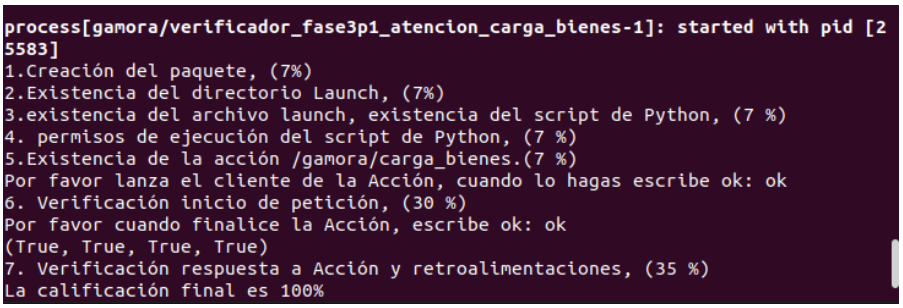


Figura 3.47 *Verificador fase 1. Fuente: Elaboración propia.*

Para este proyecto también se generaron paquetes verificadores como se observa en la Figura 3.20, para el primer paquete verificador se lanza con el siguiente comando en la terminal. Una vez ejecutado, se observa lo establecido en la Figura 3.47:

$ roslaunch verificador_fase3p1_atencion_carga_bienes verificador_fase3p1_atencion_carga_bienes.launch namespace:=/gamora token:=/1234

Para el paquete verificador 2 se ejecuta con el siguiente comando Una vez ejecutado, se observa lo establecido en la Figura 3.48:

$ roslaunch verificador_fase4p1_atencion_carga_bienes verificador_fase4p1_atencion_carga_bienes.launch namespace:=/gamora token:=1234

```
process[gamora/verificador_fase4p1_finalizacion_carga_bienes-1]: started with pi
d [25726]
1.Creación del paquete, (7%)
2.Existencia del directorio Launch, (7%)
3.existencia del archivo launch, existencia del script de Python, (7 %)
4. permisos de ejecución del script de Python, (7 %)
5.Existencia de la acción /gamora/carga_bienes.(7 %)
Por favor cuando finalice la Acción, escribe ok: ok
7. Verificación respuesta a Acción y retroalimentaciones, (65 %)
La calificación final es 100%
```

Figura 3.48 *Verificador de la fase 2. Fuente: Elaboración propia.*

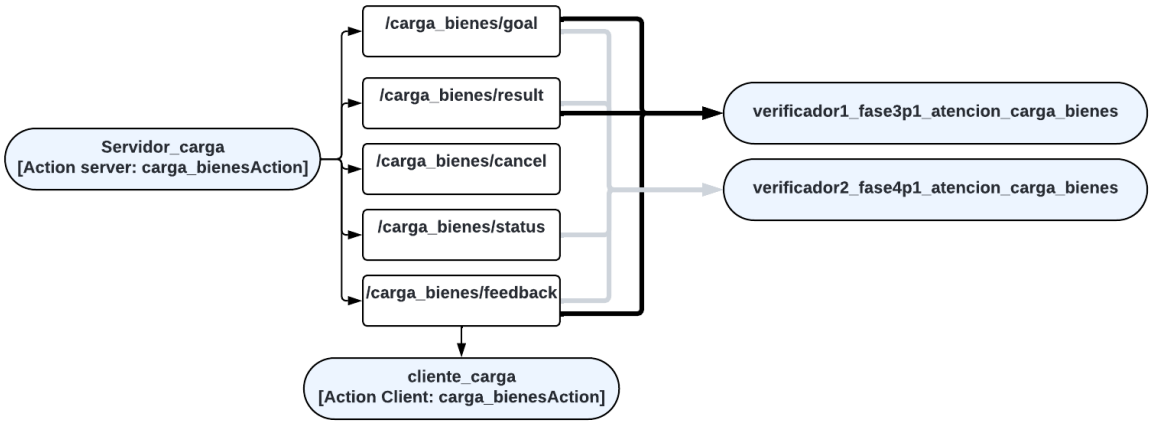


Figura 3.49 *Paquetes verificadores de Pickup-Master.*

El diagrama de nodos de la Figura 3.49 ilustra la interacción entre los paquetes verificadores y los tópicos asociados a la acción del robot. El paquete verificador 1 se encarga de suscribirse a los tópicos /carga_bienes/goal, /carga_bienes/feedback y /carga_bienes/result, con el propósito de validar el correcto funcionamiento de la acción. A través del tópico /carga_bienes/goal, el verificador comprueba que el objetivo (color y forma del objeto) haya sido declarado correctamente y que la acción pueda iniciarse sin errores. Mediante el tópico /carga_bienes/feedback, se analiza la retroalimentación enviada por el servidor de acción para verificar las distintas etapas del proceso, como cuando el robot se encuentra “Desocupado”, “Cargando” o “Procesando una petición”. Finalmente, el tópico /carga_bienes/result permite confirmar que la acción concluyó exitosamente, validando que el robot completó la tarea asignada.

Por su parte, el paquete verificador 2 se utiliza para comprobar la existencia de la acción completa en el entorno de ROS. Este verificador no solo revisa los tópicos anteriores, sino también los canales de comunicación que garantizan la estructura estándar de una acción, verificando que estén activos los tópicos /carga_bienes/status y /carga_bienes/cancel,

además de los ya mencionados goal, feedback y result. Con ello, el sistema puede asegurar que tanto el servidor de acción como el cliente fueron implementados correctamente y que los componentes de la acción están operando dentro del namespace correspondiente. En conjunto, ambos verificadores permiten evaluar de manera integral la implementación y ejecución de la acción /carga_bienes, garantizando su correcta comunicación y funcionamiento dentro del entorno del robot manipulador.

3.5 CONCLUSIONES

La implementación del aplicativo Edu-Arm permitió consolidar una plataforma integral para el aprendizaje y la experimentación en robótica manipuladora, combinando de manera efectiva el entorno simulado y el entorno real. La correcta configuración del sistema garantizó una comunicación estable entre el robot, el sistema operativo ROS y la interfaz de usuario, asegurando la interacción fluida entre los distintos componentes.

La interfaz de usuario se consolidó como el núcleo funcional del aplicativo, al integrar tres módulos principales: configuración, percepción y calibración. El módulo de configuración permitió gestionar los parámetros de conexión, establecer la comunicación con el robot y controlar sus movimientos en los diferentes modos de operación. El módulo de percepción incorporó herramientas de visión artificial que facilitaron el reconocimiento de objetos por color y por forma, mejorando la interacción del robot con su entorno y ampliando sus capacidades funcionales. Por último, el módulo de calibración permitió medir las posiciones y referencias espaciales del robot dentro del entorno de trabajo, garantizando la precisión necesaria para la correcta ejecución de las tareas de manipulación. En conjunto, estos módulos brindaron una experiencia de uso completa, didáctica y adaptable a distintos niveles de aprendizaje.

Por otro lado, la ruta de aprendizaje diseñada para instruir al usuario proporcionó un esquema formativo progresivo, dado que las tres sesiones *Crazy Gripper*, *Arm-Eye* y *Arm-Commander* evidenciaron la versatilidad del sistema y su potencial para integrar control, visión artificial y planificación de trayectorias en tiempo real.

El desarrollo de los paquetes verificadores representó un aporte clave al proceso educativo, al permitir la evaluación automática del desempeño del estudiante y garantizar la coherencia de sus implementaciones con la estructura de ROS. Finalmente, el proyecto Pickup-Máster integró todos los componentes diseñados, demostrando la capacidad del sistema para ejecutar tareas de manipulación inteligente mediante la coordinación entre visión, control y planificación.

En conjunto, Edu-Arm se consolida como una herramienta robusta y pedagógicamente valiosa que promueve la comprensión práctica de la robótica, fomenta la autonomía del estudiante y abre la posibilidad de expandir su aplicación hacia entornos educativos más amplios y proyectos de investigación en manipulación robótica avanzada.

4. PRUEBAS Y RESULTADOS

"La robótica educativa transforma la enseñanza tradicional en un proceso activo, donde el error es una oportunidad para aprender." — M. Resnick, investigador del MIT Media Lab.



Figura 4.1 *Escenario real IEP (Plataforma educativa innovadora). Fuente: Tomado de [33].*

4. PRUEBAS Y RESULTADOS 70

4.1. Introducción 70

4.2. PRUEBAS DE INTEGRACIÓN 70

4.3. PRUEBAS DE CAMPO 81

4.4. CONCLUSIONES 84

4.1 INTRODUCCIÓN

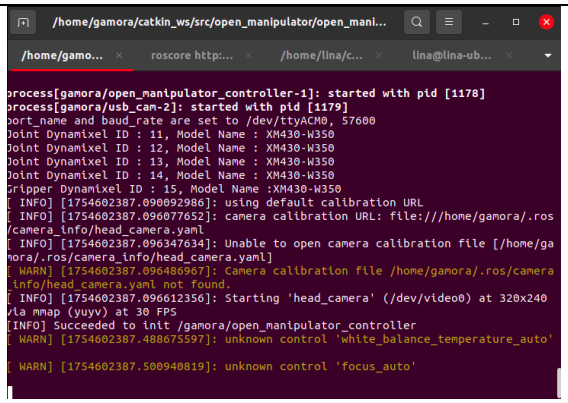
En este capítulo se presenta el procedimiento utilizado para evaluar el cumplimiento de los requerimientos funcionales y la aceptabilidad del aplicativo, con el objetivo de garantizar resultados confiables que cumplan con las expectativas del usuario final. Se muestran los resultados obtenidos a partir de las pruebas de integración y de campo. Las pruebas de integración permitieron verificar la interacción y el funcionamiento conjunto de cada componente del aplicativo con el robot manipulador en un entorno real. Por su parte, las pruebas de campo se llevaron a cabo con estudiantes de la Escuela de Ingeniería Eléctrica y Electrónica de la Universidad del Valle, sede Santiago de Cali.

4.2 PRUEBAS DE INTEGRACIÓN

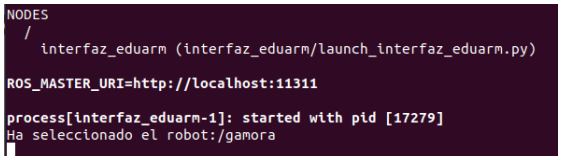
Para garantizar el correcto funcionamiento del aplicativo Edu-Arm y la adecuada comunicación entre sus componentes, se realizó una sesión de pruebas de integración enfocada en la verificación de cada uno de los módulos que conforman la interfaz de usuario. Estas pruebas tuvieron como objetivo comprobar la interacción entre la interfaz gráfica, el sistema operativo ROS y el robot, tanto en el entorno simulado como en el real, evaluando la estabilidad, la respuesta y la ejecución de los servicios asociados a cada módulo. En la Tabla 4.1 se presentan los resultados de las pruebas de integración del módulo de configuración de la interfaz Edu-Arm; en la Tabla 4.2, las correspondientes al módulo de percepción; y finalmente, en la Tabla 4.3, las pruebas realizadas al módulo de calibración.

Tabla 4.1 Pruebas de Integración del módulo configuración de la Interfaz EDU-ARM.

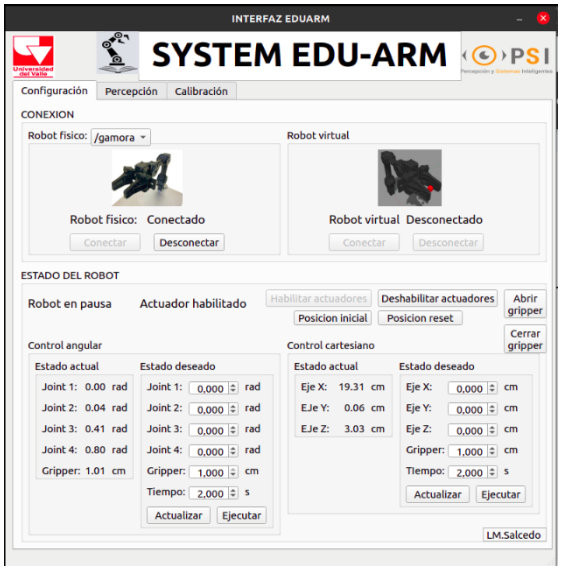
PRUEBAS DE INTEGRACIÓN- MÓDULO DE CONFIGURACIÓN	
• Establecer conexión con el robot real. • Habilitar y deshabilitar los actuadores del robot. • Abrir y Cerrar gripper del robot. • Llevar robot a posición inicial y posición de reset. • Enviar al robot a una posición específica en el plano articular. • Enviar el robot a una posición específica en el plano cartesiano.	
Tipo de Prueba	Integración
Hardware Requerido	1. Computador con características: • Sistema operativo Ubuntu 20.04.5 LTS (Focal Fossa) • ROS Noetic desktop-full (versión 1.15) 2. Open Manipulator-X con las siguientes características: • Raspberry pi 4B. • OpenCR 1.0. • Módulo de cámara raspberry pi V1.0
Software Requerido	Interfaz Edu-Arm
Objetivo	Validar el correcto funcionamiento del módulo configuración de la Interfaz EDU-ARM.
Establecer conexión con el robot real	
Objetivo	Validar la correcta comunicación entre la interfaz en el computador del usuario y el robot open Manipulator-x real.
Datos de la prueba	En esta prueba se necesita que el usuario realice las configuraciones para conectarse a la red del laboratorio de la Escuela de Ingeniería eléctrica y electrónica, abra la interfaz Edu-Arm y se conecte con el robot real.
Procedimiento	1. Conectar el computador del usuario a la red local del laboratorio utilizando las configuraciones de IPv4 establecidas para el entorno de trabajo. 2. Alimentar las tarjetas OpenCR 1.0 y la Raspberry Pi siguiendo las instrucciones descritas en la guía “Interfaz Edu-Arm: guía de uso”, ubicada en el repositorio de GitHub del proyecto. 3. Iniciar el entorno de ROS. 4. Lanzar el controlador. 5. Ejecutar la Interfaz. 6. Seleccionar el robot a usar. 7. Presionar el botón “Conectar”.
Resultado esperado	1. Se habilitan todos los botones de la Interfaz EDU-ARM para operar el robot. 2. Se ejecuta el nodo del controlador del robot y el nodo de la interfaz.
Resultado Obtenido	1. La terminal del robot en donde se lanzó el controlador muestra éxito en ejecución:



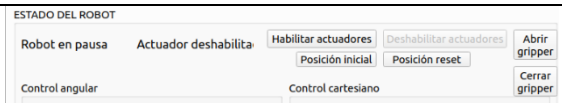
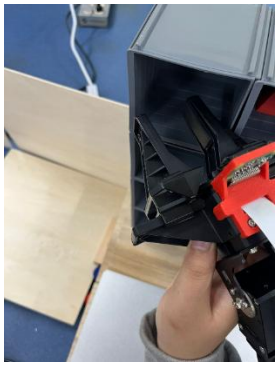
2. Al lanzar la interfaz Edu-Arm se ejecuta el nodo de la interfaz, y al seleccionar el namespace “gamora” y presionar “conectar” la terminal de la interfaz informa la selección:

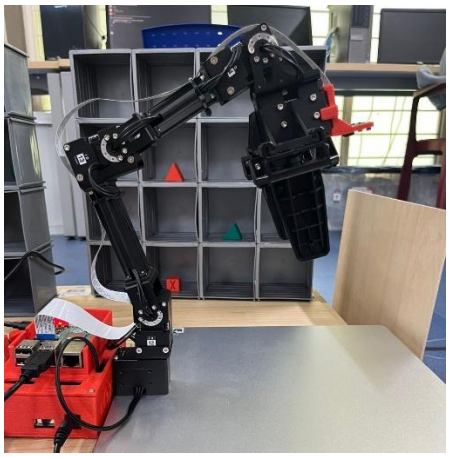
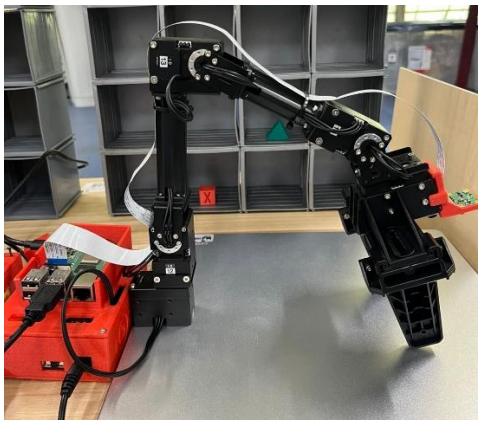
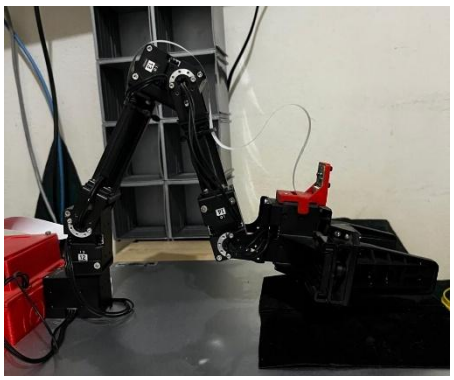


3. La interfaz Edu-Arm se muestra con los botones habilitados una vez se logra la exitosa conexión con el robot:



Habilitar y deshabilitar los actuadores del robot	
Objetivo	Verificar el correcto estado de habilitación o deshabilitación de los actuadores del robot.
Datos de la prueba	Se prueba que el botón “Habilitar actuadores” funcione correctamente.
Procedimiento	1. En la interfaz EDU-ARM presionar el botón “Habilitar actuadores”. 2. Revisar que el robot real puede moverse manualmente sin presentar fuerza o resistencia por parte de los actuadores.
Resultado esperado	Los actuadores del robot se deshabilitan.
Resultado obtenido	1. Al presionar el botón “Deshabilitar actuadores” el botón se deshabilita y solo queda activo el de “Habilitar actuadores”.

	 2. El robot no presenta resistencia al moverlo manualmente. 
Abrir o cerrar el gripper del robot	
Objetivo	Validar que con la Interfaz EDU-ARM el robot si abre y cierra el gripper según se indique.
Datos de la prueba	En esta prueba se necesita que el usuario abre y cierre el gripper del robot usando los botones de la interfaz “Abrir gripper” y “Cerrar gripper”.
Procedimiento	1. El usuario presiona “Abrir gripper”. 2. Revisar que se abra el gripper del robot. 3. El usuario presiona “Cerrar Gripper”. 4. Revisar que se cierre el gripper del robot”.
Resultado esperado	El robot abre y cierra su gripper.
Resultado obtenido	1. Una vez el usuario presiona “Abrir gripper” se abre el gripper del robot. 2. Una vez el usuario presiona “Cerrar gripper” se cierra el gripper del robot. 
Llevar robot a posición inicial y posición de reinicio.	
Objetivo	Validar que el robot Open Manipulator x se dirige a su posición inicial y a su posición de reinicio según se indique en la interfaz.
Datos de la prueba	En esta prueba se necesita que el usuario presione el botón “Posición inicial” y “Posición reinicio” para mover el robot.
Procedimiento	1. Presionar el botón “Posición inicial”. 2. Verificar que el robot se desplaza a su posición inicial. 3. Presionar botón “Posición reinicio”. 4. Verificar que el robot se desplaza a su posición de reinicio.
Resultado esperado	El robot se desplaza a su posición inicial y posición de reinicio.

Resultado obtenido	1. Al presionar el botón posición inicial, el robot se desplaza a su posición inicial:  2. Al presionar el botón “Posición de reinicio” el robot se desplaza a su posición de reinicio: 
Se envía el robot a una posición en el espacio cartesiano, y la interfaz visualiza el estado del robot.	
Objetivo	Verificar que, al asignarle una ubicación en coordenadas cartesianas en la interfaz, el efector final del robot se desplace correctamente dentro de su espacio cartesiano.
Datos de la prueba	En esta prueba se envía al robot las coordenadas cartesianas [x,y,z]=[18.00,0.00,7.00].
Procedimiento	1. Se escribe la coordenada X en la casilla de la sección de movimiento cartesiano. 2. Se escribe la coordenada Y en la casilla de la sección de movimiento cartesiano. 3.Se escribe la coordenada Z en la casilla de la sección de movimiento cartesiano.
Resultado esperado	El robot se moviliza a la posición [x,y,z]=[18.00,0.00,7.00].
Resultado obtenido	Una vez escritos los valores en la Interfaz EDU-ARM, el robot se desplaza a la posición [x,y,z]=[18.00,0.00,7.00]. 

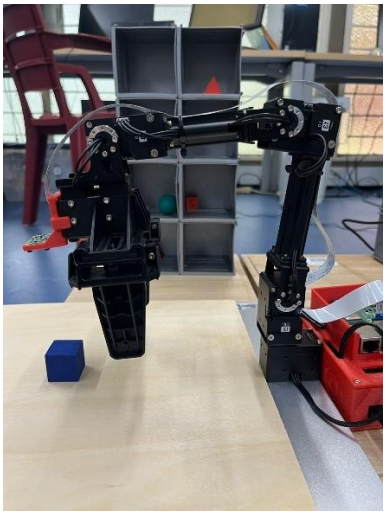
Se envía el robot a una posición en el espacio articular y la interfaz supervisa el estado del robot	
Objetivo	Verificar que, al asignarle una posición en la interfaz, el robot se desplaza para cumplir con esa posición en el espacio articular.
Datos de la prueba	En esta prueba se envía al robot a una posición en el espacio articular [j1,j2,j3,j4]=[0.00,0.00,0.00,1.456].
Procedimiento	1. Se escribe el dato de la articulación 1 (Joint1). 2. Se escribe el dato de la articulación 2 (Joint2). 3. Se escribe el dato de la articulación 3 (Joint3). 4. Se escribe el dato de la articulación 4 (Joint4). 5. Presionar el botón ejecutar. 6. Verificar que el robot sí se desplaza en el plano articular a la posición deseada.
Resultado esperado	El robot se moviliza a la posición [j1,j2,j3,j4]=[0.00,0.00,0.00,1.456].
Resultado obtenido	Una vez ingresados los datos en la sección movimiento articular de la interfaz. El robot se desplaza a la posición [j1,j2,j3,j4]=[0.00,0.00,0.00,1.456]. 

Tabla 4.2 Pruebas de Integración del módulo Percepción de la Interfaz Edu-ARM

PRUEBAS DE INTEGRACIÓN- MÓDULO DE PERCEPCIÓN	
• Usar reconocimiento de objetos por color. • Usar reconocimiento de objetos por forma.	
Tipo de Prueba	Integración
Hardware Requerido	1. Computador con características: • Sistema operativo Ubuntu 20.04.5 LTS (Focal Fossa) • ROS Noetic desktop-full (versión 1.15) 2. Open Manipulator-X con las siguientes características: • Raspberry pi 4B. • OpenCR 1.0. • Módulo de cámara raspberry pi V1.0
Software Requerido	Interfaz EDU-ARM
Objetivo	Validar el correcto funcionamiento del módulo Percepción de la Interfaz EDU-ARM.

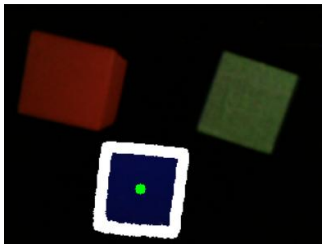
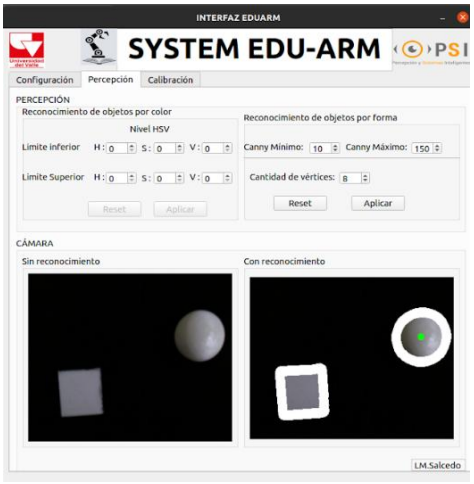
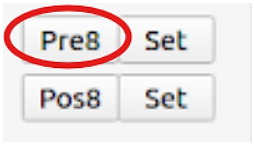
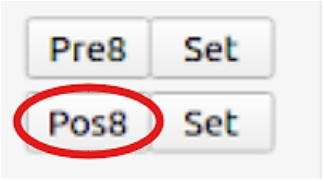
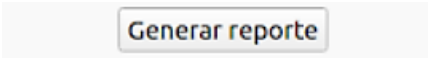
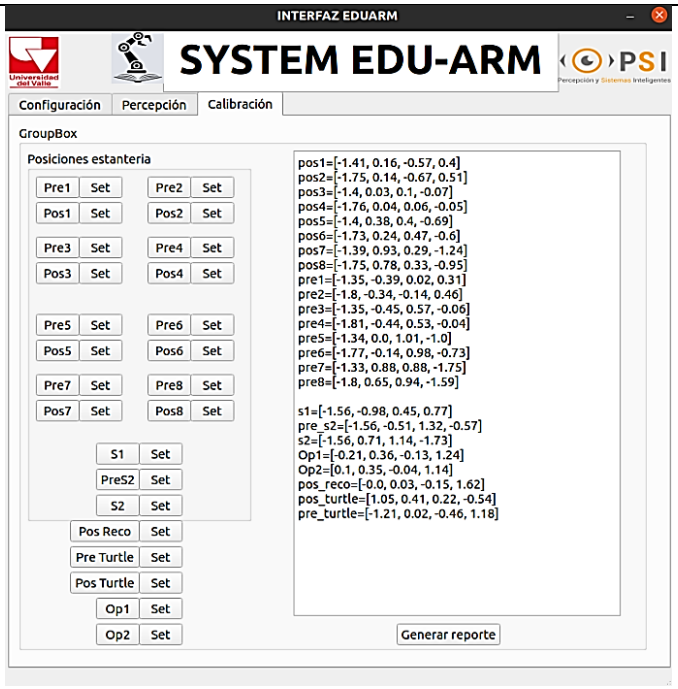
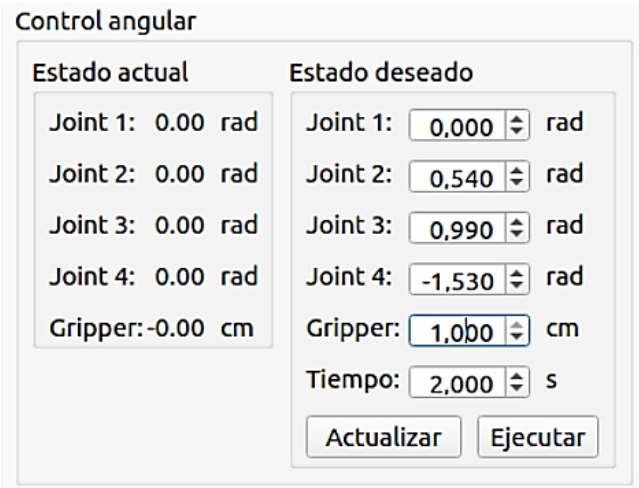
Usar reconocimiento de objetos por color	
Objetivo	Verificar el funcionamiento del algoritmo de reconocimiento de objetos por color en la Interfaz EDU-ARM.
Datos de la prueba	En esta prueba se prueba el algoritmo con un HSV mínimo [100,50,50] y un HSV máximo [140,255,255]
Procedimiento	1. Ingresar en las casillas HSV mínimo de [100,50,50] 2.Ingresar en las casillas HSV máximo [140,255,255]. 3. Verificar que la interfaz muestre la imagen tanto sin reconocimiento de objetos como con el reconocimiento de objetos de color azul aplicado.
Resultado esperado	El sistema reconoce objetos de color azul.
Resultado obtenido	Al ingresar los parámetros de HSV mínimo y HSV máximo el sistema muestra en la ventana de ejecución. 
Usar reconocimiento de objetos por forma	
Objetivo	Verificar el funcionamiento del algoritmo de reconocimiento de objetos por forma en la Interfaz EDU-ARM.
Datos de la prueba	En esta prueba se prueba el algoritmo con un Canny mínimo= 30, un Canny Máximo=150, y cantidad de lados=8
Procedimiento	1. Ingresar en las casillas de la interfaz un Canny mínimo =30 2. Ingresar en las casillas de la interfaz un Canny máximo =150. 3. Ingresar la cantidad de lados=8 en la casilla. 4. Presionar el botón “Ejecutar”. 5. Verificar que la interfaz muestra la imagen tanto sin reconocimiento de objetos como con el reconocimiento de objetos de forma redonda aplicados.
Resultado esperado	El sistema reconoce objetos de forma redonda.
Resultado obtenido	Al ingresar los parámetros de Canny mínimo, Canny máximo y la cantidad de lados, la interfaz muestra la imagen con el objeto de forma redonda contorneado y dibujado su centroide. 

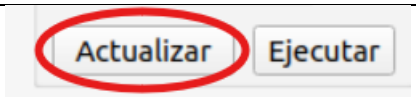
Tabla 4.3 Pruebas de Integración del módulo de Calibración de la Interfaz Edu-Arm.

<table>
<tr><th colspan="2">PRUEBAS DE INTEGRACIÓN- MÓDULO DE CALIBRACIÓN</th></tr>
<tr><td colspan="2">• Llevar el robot a una posición pre8.
• Llevar el robot a una posición pos8.
• Generar reporte de posición.
• Actualizar posiciones con botón set y generar reporte con la actualización.</td></tr>
<tr><td>Tipo de Prueba</td><td>Integración</td></tr>
<tr><td>Hardware Requerido</td><td>1. Computador con características:
Sistema operativo Ubuntu 20.04.5 LTS (Focal Fossa)
ROS Noetic desktop-full (versión 1.15)
2. Open Manipulator-X con las siguientes características:
Raspberry pi 4B.
OpenCR 1.0.
Módulo de cámara raspberry pi V1.0</td></tr>
<tr><td>Software Requerido</td><td>Interfaz Edu-Arm.</td></tr>
<tr><td>Objetivo</td><td>Validar el correcto funcionamiento del módulo Calibración de la Interfaz EDU-ARM.</td></tr>
<tr><th colspan="2">Llevar el robot a una posición pre8.</th></tr>
<tr><td>Objetivo</td><td>Verificar el funcionamiento del botón “Pre8” para llevar al robot a la posición pre1 de la estantería.</td></tr>
<tr><td>Datos de la prueba</td><td>En esta prueba se lleva l robot a la posición Pre8.</td></tr>
<tr><td>Procedimiento</td><td>1. Conectar el robot real.
2. Presionar el botón de “Pre8”.
3. El robot se dirige a la posición Pre8.</td></tr>
<tr><td>Resultado esperado</td><td>El robot se dirige a la posición Pre8.</td></tr>
<tr><td>Resultado obtenido</td><td>1. Presionar el botón “Pre8”.


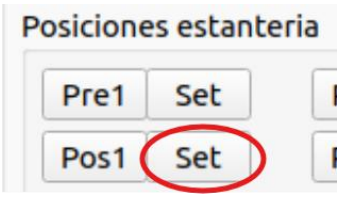
2. El robot se dirige a la posición “Pre8”.
</td></tr>
</table>

Llevar el robot a una posición pos8.	
Objetivo	Verificar el funcionamiento del botón “Pos8” para llevar al robot a la posición pos1 de la estantería.
Datos de la prueba	Se lleva el botón a la posición Pos8
Procedimiento	1. Conectar el robot real. 2. Presionar el botón de “Pos8”. 3. El robot se dirige a la posición Pos8.
Resultado esperado	El robot se dirige a la posición Pos8.
Resultado obtenido	1. Presionar el botón “Pos8”.  2. El robot se dirige a la posición “Pos8”. 
Generar reporte de posición.	
Objetivo	Verificar el funcionamiento del botón “Generar reporte” para obtener el listado de todas las posiciones guardadas
Datos de la prueba	Una vez conectado el robot, se genera el reporte del listado de las posiciones guardadas.
Procedimiento	1. Conectar el robot real. 2. Presionar el botón “Generar reporte”.
Resultado esperado	La interfaz muestra el listado de las posiciones guardadas.
Resultado obtenido	1. Presionar el botón “Generar reporte”.  2. La interfaz muestra el listado de posiciones guardadas, como se muestra a continuación:

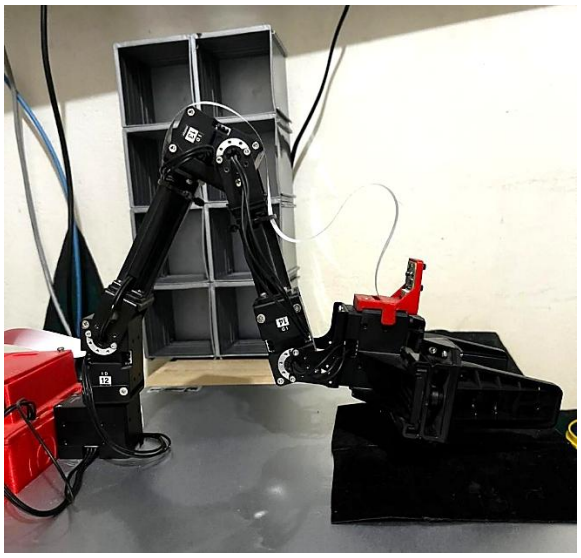
	
	Actualizar posiciones con botón set y generar reporte con la actualización.
Objetivo	Verificar que, al llevar el robot a una posición específica y presionar los botones “Set” y “Generar reporte”, se genere el listado de posiciones articulares con la información de la posición actualizada.
Datos de la prueba	Se lleva el robot a la posición j1,j2,j3,j4=[0.00, 0.540,0.990, -1.530], se presiona el botón set de la posición 1 y se genera el reporte para ver la posición actualizada.
Procedimiento	1. Conectar el robot real. 2. Ingresar los datos de posición cartesiana para enviar el robot en la posición deseada [0.00, 0.540,0.990, -1.530]. 3. Presionar “Ejecutar”. 4. Presionar el botón “Set”. 5. Presionar “Generar reporte”.
Resultado esperado	Al generar el reporte se actualiza la posición cambiada.
Resultado obtenido	1. Se ingresan los parámetros j1,j2,j3,j4 en las casillas del estado deseado en la interfaz, como se muestra a continuación:  2. Se presiona el botón “Ejecutar”.



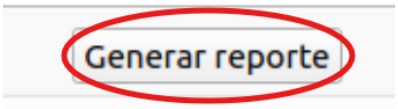
3. Se presiona el botón “Set”.



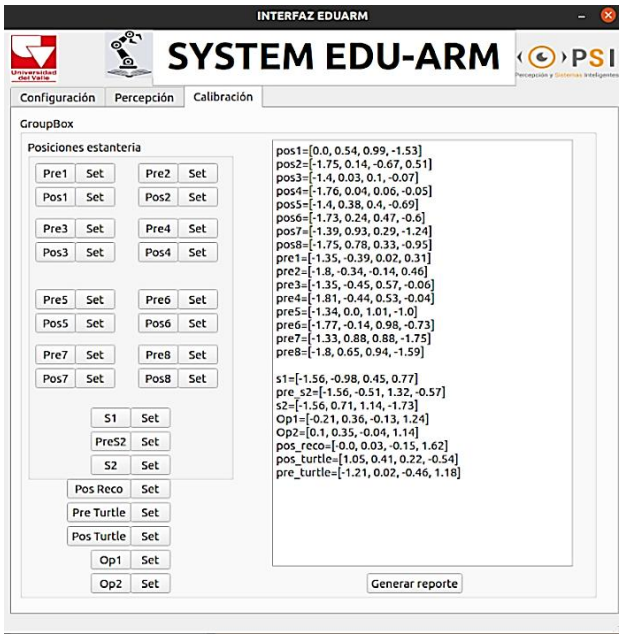
4. El robot se dirige a la posición deseada.



5. Se presiona Generar reporte.



6. La interfaz muestra el nuevo listado con la posición actualizada.



4.3 PRUEBAS DE CAMPO

Las pruebas de campo se llevaron a cabo con ocho estudiantes de la Escuela de Ingeniería Eléctrica y Electrónica de la Universidad del Valle, quienes contaban con conocimientos en robótica móvil, programación en Python y el uso de ROS. En la Figura 4.2 se muestra la evidencia de algunos estudiantes probando el aplicativo usando la computadora XPS del laboratorio de robótica Móvil y sus computadoras personales.

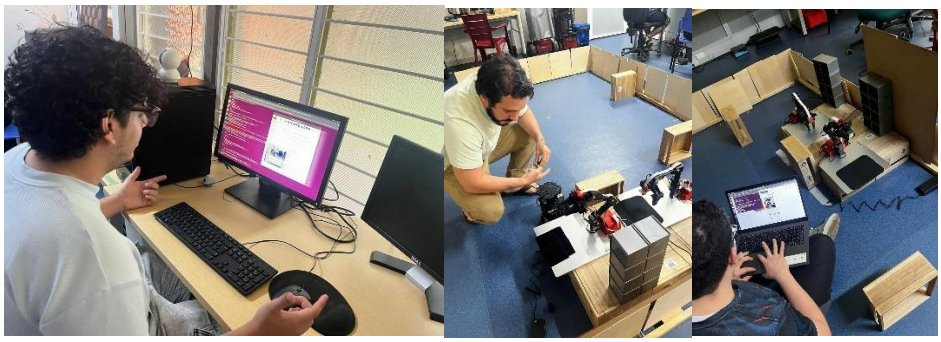


Figura 4.2 *Pruebas de campo del aplicativo EDU-ARM en la Universidad del Valle*

La prueba de campo consistió en que los estudiantes leyeran la documentación orientativa de la ruta de aprendizaje ubicada en el repositorio del proyecto y la llevaran a cabo utilizando los robots manipuladores reales.

- **EVALUACIÓN DEL AMBIENTE DE APRENDIZAJE**

La evaluación del entorno de aprendizaje se realizó mediante una encuesta dirigida a los estudiantes que participaron en las pruebas de cada sesión de la ruta de aprendizaje. Esta encuesta estuvo conformada por 12 preguntas con escala tipo Likert, cuyas opciones de respuesta iban desde “totalmente de acuerdo” hasta “totalmente en desacuerdo”. En la Tabla 4.4 se muestran las preguntas aplicadas a los estudiantes tras la ejecución de las Sesiones 1, 2 y 3 de la ruta de aprendizaje.

Tabla 4.4 Preguntas de la encuesta

#	Preguntas
1	El robot Open manipuator-x fue fácil de utilizar durante las pruebas de campo.
2	La Interfaz Edu-Arm y la documentación guiada me permitió usar el robot Open Manipulator x dentro de ROS.
3	Siento que mejoré mis habilidades en programación con Python aplicado a robótica.
4	El entorno de trabajo fue adecuado para realizar actividades de almacenamiento, transporte, carga y descarga con el robot manipulador.
5	La información suministrada en las guías es suficiente para comprender el funcionamiento del robot OpenManipulator X.
6	Los documentos guía fueron claros y comprensibles para desarrollar las actividades.
7	Pude desarrollar los ejercicios propuestos y los paquetes verificadores me sirvieron para comprobar mi trabajo.
8	Los paquetes verificadores me ayudaron a identificar errores en mis códigos.
9	Los mensajes de los paquetes verificadores fueron claros y fáciles de interpretar.
10	La interfaz de usuario Edu-Arm es intuitiva y fácil de usar.
11	Considero que la Interfaz Edu-Arm, la documentación guiada y el uso del robot manipulador fueron suficientes para realizar las actividades de transporte de objetos.
12	Considero que la Interfaz Edu-Arm, la documentación guiada y el uso del robot manipulador fueron suficientes para realizar las actividades de reconocimiento de objetos por color y forma.

La encuesta fue aplicada a los estudiantes una vez finalizada la actividad de pruebas de campo. Los resultados obtenidos se presentan en los diagramas circulares mostrados en la Figura 4.3, donde cada gráfico corresponde a una de las preguntas planteadas, abarcando desde la pregunta 1 hasta la 13. Asimismo, se incluyen las opciones de respuesta representadas mediante los siguientes colores:

a) Totalmente de acuerdo (azul).

b) De acuerdo (rojo).

c) Neutral (amarillo).

d) En desacuerdo (verde).

e) Totalmente en desacuerdo (púrpura).

En la figura también se aprecia la distribución cromática de las respuestas, destacándose principalmente los colores rojo, amarillo y azul, que representan las opciones a,b, y c como mayor frecuencia de selección.

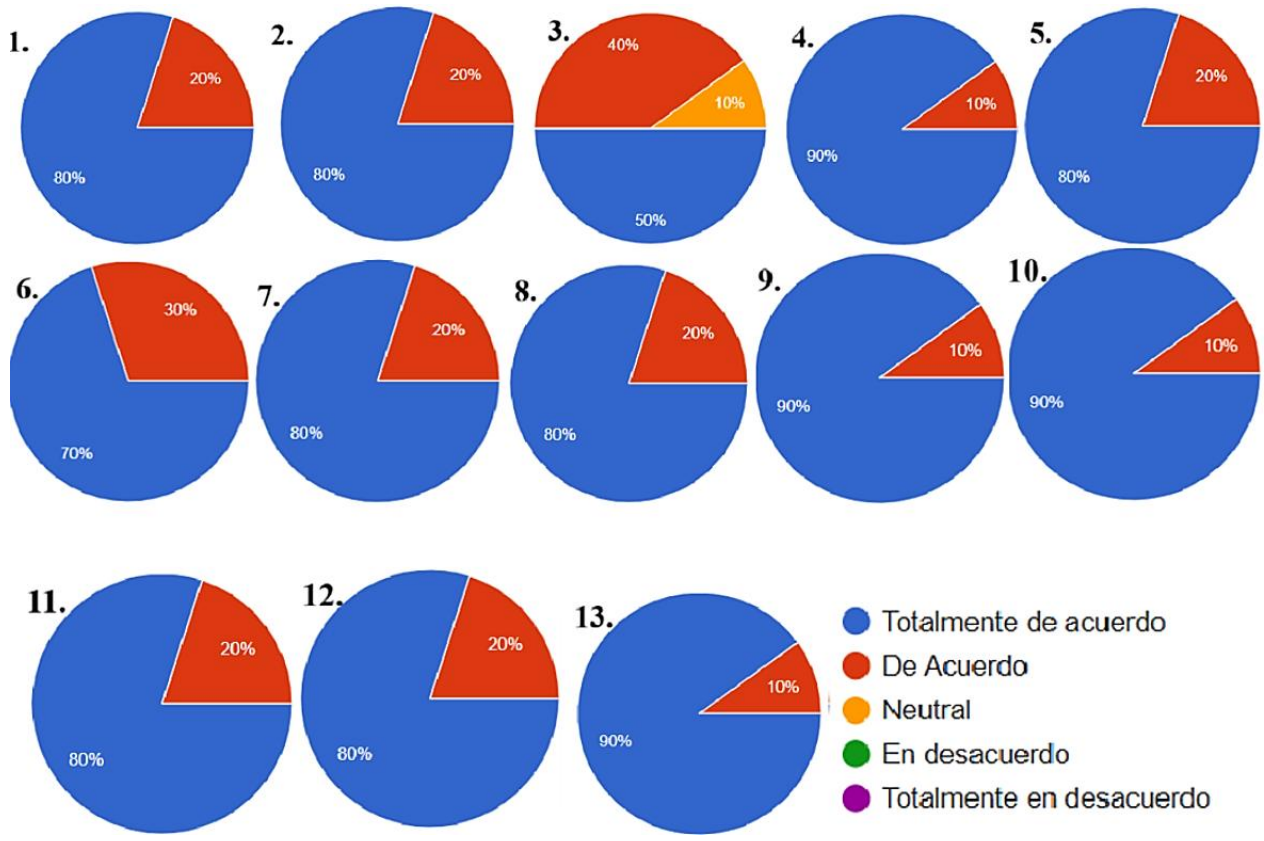


Figura 4.3 *Resumen de resultados de la encuesta de evaluación del ambiente de aprendizaje.*

Con base en las respuestas proporcionadas por los estudiantes en la encuesta, la Tabla 4.4 presenta un resumen de los resultados obtenidos en la evaluación del ambiente de aprendizaje. En la columna “Conclusión” se incluye un análisis derivado de dichas respuestas.

Tabla 4.5 Resumen encuesta a estudiantes en pruebas de campo.

Pregunta	a	b	c	d	e	Conclusión
1	75%	25%	0%	0%	0%	La mayoría de los estudiantes (75% totalmente de acuerdo y 25% de acuerdo) consideró que el robot OpenManipulator-X fue fácil de utilizar durante las pruebas de campo, lo que refleja una buena usabilidad en la práctica.
2	75%	25%	0%	0%	0%	Un 100% de los participantes coincidió (75% totalmente de acuerdo, 25% de acuerdo) en que la Interfaz Edu-Arm y la documentación guiada facilitaron el uso del robot dentro de ROS, lo que evidencia que el material fue pertinente y útil.
3	50%	37.5%	12.5%	0%	0%	El 87.5% de los estudiantes (50% totalmente de acuerdo, 37.5% de acuerdo) reconoció una mejora en sus habilidades de programación en Python aplicadas a la robótica, aunque un 12.5% permaneció neutral, lo que sugiere que algunos perciben un impacto menor en este aspecto.
4	87,5%	12.5%	0%	0%	0%	Un 100% de los encuestados (87.5% totalmente de acuerdo, 12.5% de acuerdo) afirmó que el entorno de trabajo fue adecuado para las actividades de almacenamiento, transporte, carga y descarga, lo que garantiza que las condiciones fueron propicias para el aprendizaje.
5	87.5%	12.5%	0%	0%	0%	La totalidad de los participantes (87.5% totalmente de acuerdo, 12.5% de acuerdo) indicó que la información de las guías fue suficiente para comprender el funcionamiento del robot, demostrando claridad y completitud en el material educativo.
6	75%	25%	0%	0%	0%	El 100% (75% totalmente de acuerdo, 25% de acuerdo) consideró que los documentos guía fueron claros y comprensibles para desarrollar las actividades, lo cual refuerza la efectividad del recurso didáctico.
7	87.5%	12.5%	0%	0%	0%	Un 100% (87.5% totalmente de acuerdo, 12.5% de acuerdo) aseguró que los ejercicios y paquetes verificadores permitieron comprobar adecuadamente su trabajo, destacando su utilidad como herramienta de retroalimentación.
8	87.5%	12.5%	0%	0%	0%	El 100% (87.5% totalmente de acuerdo, 12.5% de acuerdo) manifestó que los paquetes verificadores les ayudaron a identificar errores en sus códigos, confirmando su valor como apoyo en la detección de fallas.
9	87.5%	12.5%	0%	0%	0%	El 100% de los estudiantes (87.5% totalmente de acuerdo, 12.5% de acuerdo) opinó que los mensajes de los paquetes verificadores fueron claros y fáciles de interpretar, lo que evidencia que la retroalimentación fue comprensible.
10	87.5%	12.5%	0%	0%	0%	El 100% (87.5% totalmente de acuerdo, 12.5% de acuerdo) consideró que la interfaz Edu-Arm es intuitiva y fácil de usar, lo que refleja una buena experiencia de usuario en la interacción con el sistema.
11	87.5%	12.5%	0%	0%	0%	La totalidad de los encuestados (87.5% totalmente de acuerdo, 12.5% de acuerdo) coincidió en que la combinación de la Interfaz Edu-Arm, la documentación y el robot fue suficiente para realizar actividades de transporte de objetos, demostrando la efectividad de la integración.
12	75%	25%	0%	0%	0%	El 100% (75% totalmente de acuerdo, 25% de acuerdo) opinó que la Interfaz Edu-Arm, la documentación y el robot fueron suficientes para realizar actividades de reconocimiento de objetos por color y forma, lo que valida la utilidad del conjunto de herramientas para este fin.
13	87.5%	12.5%	0%	0%	0%	Finalmente, el 100% de los estudiantes (87.5% totalmente de acuerdo, 12.5% de acuerdo) afirmó que la Interfaz Edu-Arm, la documentación guiada y el uso del robot fueron suficientes para realizar actividades de carga y descarga de objetos a partir de su color y forma, confirmando la pertinencia del ambiente de aprendizaje para tareas más complejas.

4.4 CONCLUSIONES

En la etapa de pruebas de integración se verificó el cumplimiento de los principales requerimientos funcionales del sistema. Los estudiantes utilizaron la interfaz gráfica de usuario (GUI) para supervisar y teleoperar el robot Open Manipulator X de Robotis, controlando la posición y orientación del efector final. A lo largo de la ruta de aprendizaje, pudieron aplicar las herramientas de filtrado de objetos basadas en patrones de color y formas geométricas mediante el uso de la cámara del robot. El sistema demostró su capacidad para permitir el agarre manual de objetos con el gripper, grabar posiciones en el plano, tomar elementos desde la parte superior del robot móvil y almacenarlos en ubicaciones específicas dentro de una estantería. Asimismo, los estudiantes comprobaron la correcta interacción entre los paquetes verificadores y la interfaz de comunicación web, estableciendo el intercambio de datos con la aplicación iBotAcademy. Además, se constató que tanto el hardware como el software del robot satisfacen los requerimientos no funcionales, ya que el desarrollo se llevó a cabo en Python, con una interfaz creada en QtDesigner, bajo el sistema operativo Ubuntu 20.04.5 y el marco de desarrollo ROS Noetic. También se verificó el funcionamiento del entorno virtual en el simulador Gazebo, utilizando como modelo de robot el Open Manipulator X de Robotis. En conjunto, los resultados obtenidos evidencian la correcta integración y operatividad del sistema, así como el cumplimiento de los objetivos planteados en el desarrollo del aplicativo.

Los resultados de la encuesta aplicada a los estudiantes demuestran una percepción altamente positiva del ambiente de aprendizaje implementado con el robot OpenManipulator-X y la Interfaz Edu-Arm. En todos los aspectos evaluados, la totalidad de los participantes manifestó estar de acuerdo o totalmente de acuerdo, lo que evidencia una adecuada usabilidad, claridad en la documentación y efectividad de las herramientas utilizadas.

Los estudiantes destacaron la facilidad de uso del robot, la pertinencia de la interfaz y la utilidad de los paquetes verificadores para identificar errores y reforzar el aprendizaje. Asimismo, reconocieron una mejora en sus habilidades de programación en Python y valoraron positivamente el entorno de trabajo, lo que indica que las condiciones técnicas y pedagógicas fueron apropiadas para el desarrollo de las actividades.

En conjunto, los resultados confirman que la propuesta educativa logró cumplir sus objetivos al integrar de forma efectiva los componentes tecnológicos y didácticos, favoreciendo un aprendizaje práctico, autónomo y significativo. El ambiente de aprendizaje resultó ser funcional, motivador y pertinente para fortalecer competencias en robótica educativa, constituyéndose en una herramienta valiosa para futuras experiencias formativas en este campo.

5. CONCLUSIONES Y TRABAJOS FUTUROS



Figura 5.1 *Robots disponibles en Laboratorio de robótica móvil de la Universidad del Valle. Fuente: Elaboración propia.*

5. CONCLUSIONES Y TRABAJOS FUTUROS 85
5.1. Conclusiones generales 85
5.2. Trabajos Futuros 86

5.1 CONCLUSIONES GENERALES

En la actualidad, la industria vive un proceso de transformación acelerado, uno de los principales desafíos que enfrentan las instituciones educativas consiste en ofrecer a los estudiantes entornos prácticos que les permitan interactuar con estas tecnologías de forma segura, accesible y formativa. Frente a esta problemática, el presente trabajo de grado se planteó como una solución orientada a fortalecer la formación de los estudiantes de la Universidad del Valle mediante el desarrollo de una herramienta software para la supervisión y teleoperación de un robot manipulador en un entorno de aprendizaje. La propuesta busca integrar la robótica y la simulación dentro del proceso educativo, permitiendo que los estudiantes apliquen conceptos teóricos en tareas de almacenamiento, carga, descarga y transporte de bienes en un ambiente controlado y pedagógico.

El primer paso consistió en sintetizar información bibliográfica sobre los métodos y procedimientos existentes para el desarrollo de software con ROS (Robot Operating System) aplicado al control y supervisión de robots manipuladores simulados. Esta revisión exhaustiva permitió identificar los enfoques más utilizados, las tecnologías disponibles y los requerimientos funcionales necesarios para diseñar un sistema educativo sólido. Los resultados de este análisis, presentados en el capítulo 2, sirvieron como base conceptual para definir los componentes tecnológicos, el framework, los lenguajes de programación, las plataformas y los sensores más adecuados.

Con esto claro, se procedió a diseñar una herramienta software orientada a la supervisión y teleoperación de un robot manipulador simulado, considerando su aplicación en procesos de almacenamiento y transporte. El diseño, descrito en el capítulo 3, abarcó desde la definición de los requerimientos funcionales y no funcionales, hasta la elaboración de diagramas conceptuales y de nodos. Se estableció que el robot debía contar con dos funcionalidades principales: movimiento y reconocimiento por color y forma. Además, se

definió como requerimiento adicional el desarrollo de una ruta de aprendizaje que guiara al estudiante en el uso progresivo del robot manipulador.

Posteriormente, se llevó a cabo la implementación del sistema, materializando el diseño conceptual en una aplicación práctica basada en ROS. La interfaz se estructuró en tres módulos principales: configuración, percepción y calibración. El módulo de configuración permite establecer la conexión con el robot y definir sus movimientos; el de percepción ofrece funciones de detección por color y forma; y el de calibración registra posiciones de referencia para las operaciones de carga y descarga, permitiendo que el usuario ajuste el entorno a los movimientos esperados del robot. Además, se implementó la ruta de aprendizaje, que comenzó con la estructuración del entorno real de trabajo del robot, seguida de la definición de sesiones secuenciales y paquetes de verificación diseñados para evaluar el progreso del estudiante al operar el robot en ejercicios prácticos dentro del entorno de ROS.

Finalmente, se realizaron pruebas de integración y de campo con el propósito de validar el funcionamiento del sistema en un entorno educativo. Las pruebas, documentadas en el capítulo 4, demostraron la eficacia de la herramienta bajo condiciones reales. El robot manipulador ejecutó con éxito las tareas de supervisión y teleoperación, evidenciando la correcta integración entre la interfaz Edu-Arm y los recursos de hardware. Asimismo, las pruebas de campo realizadas con estudiantes activos de la Universidad del Valle se validaron mediante una encuesta en donde los resultados muestran una alta satisfacción de los estudiantes con el uso del robot OpenManipulator-X y la interfaz Edu-Arm. El 100% coincidió en que la interfaz, la documentación y el robot facilitaron la ejecución de las actividades, mientras que el 75% afirmó que el robot fue fácil de usar. Asimismo, el 87.5% reconoció mejoras en sus habilidades de programación en Python, y la totalidad destacó la claridad de las guías, la utilidad de los paquetes verificadores y la adecuación del entorno de trabajo para las tareas de manipulación, transporte y reconocimiento de objetos. En conjunto, los resultados confirman que el aplicativo proporciona una experiencia de aprendizaje interactiva, didáctica y alineada con las demandas de la industria moderna.

En conclusión, este trabajo de grado ofrece a la Universidad del Valle una solución concreta para fortalecer la enseñanza de la robótica industrial. La herramienta desarrollada no solo facilita el aprendizaje práctico de los estudiantes de Ingeniería, sino que también establece una base sólida para futuros proyectos que integren nuevas tecnologías, ampliando el alcance y las posibilidades de aplicación en entornos académicos vinculados a la Industria 4.0.

5.2 TRABAJOS FUTUROS

Con base en los resultados obtenidos y las limitaciones encontradas durante el desarrollo de este trabajo, se reconocen diversas oportunidades para mejorar y ampliar el sistema, las cuales podrían servir como punto de partida para futuras investigaciones o aplicaciones prácticas.

Como proyección de este trabajo de grado, se propone para trabajos futuros continuar con la consolidación del aplicativo de teleoperación y supervisión, integrando nuevas funcionalidades que fortalezcan el proceso formativo dentro del ambiente de aprendizaje. Una posible línea de desarrollo consiste en ampliar el conjunto de actividades y escenarios de práctica, permitiendo que los estudiantes interactúen con diferentes entornos de manipulación y ejecuten tareas con distintos grados de complejidad.

También se identifica como una oportunidad de mejora la incorporación de nuevos sensores, como los de fuerza, presión o presencia, que amplíen la capacidad perceptiva del sistema y permitan ejecutar tareas de manipulación con mayor precisión y seguridad. De manera complementaria, se propone integrar técnicas de visión por computador apoyadas en inteligencia artificial, empleando algoritmos de *machine learning* o *deep learning* para el reconocimiento, clasificación y seguimiento de objetos en tiempo real. La combinación de estos sistemas sensoriales con modelos de IA (Inteligencia Artificial) permitiría dotar al robot de una mayor autonomía en la toma de decisiones, optimizar la planificación de movimientos y adaptarse dinámicamente a las condiciones del entorno. Además, la implementación de inteligencia artificial proyectaría el desarrollo hacia los estándares de la Industria 5.0, favoreciendo que los estudiantes se familiaricen con tecnologías emergentes y adquieran competencias alineadas con las tendencias actuales de la automatización inteligente [34].

Finalmente, se recomienda explorar la implementación del aplicativo en plataformas remotas o en entornos virtualizados accesibles vía web, lo que permitiría a los estudiantes interactuar con el robot desde cualquier lugar, fortaleciendo el componente de teleoperación, la autonomía en el aprendizaje y la aplicación práctica de los conocimientos adquiridos en robótica educativa.

6. REFERENCIAS

[1] Interempresas.net, “La formación en robótica en la nueva era de la Industria 4.0”, Interempresas.net. Accedido: 5 de febrero de 2026. [En línea]. Disponible en: https://www.interempresas.net/Alimentaria/368209-La-formacion-en-robotica-en-la-nueva-era-de-la-Industria-40.html

[2] H. Kim, “open_manipulator”, ROS Wiki. Accedido: 5 de febrero de 2026. [En línea]. Disponible en: https://wiki.ros.org/open_manipulator

[3] S. Wang, L. Jiang, J. Meng, Y. Xie, y H. Ding, “Training for smart manufacturing using a mobile robot-based production line”, *Frontiers of Mechanical Engineering*, vol. 16, nº 2, pp. 249–270, jun. 2021, doi: 10.1007/s11465-020-0625-z.

[4] Z. Lafhaj, W. AlBalkhy, y T. Linner, “‘Imagine and make’: teaching construction robotics for higher education students”, *Construction Robotics*, vol. 7, nº 1, pp. 65–75, may 2023, doi: 10.1007/s41693-023-00092-9.

[5] E. Tosello, N. Castaman, y E. Menegatti, “Using robotics to train students for Industry 4.0”, en *IFAC-PapersOnLine*, Elsevier B.V., 2019, pp. 177–183. doi: 10.1016/j.ifacol.2019.08.185.

[6] A. Palacios Gonzalo, “El impacto de la automatización en la sustitución de mano de obra humana por robots en la industria: perspectivas sociales, económicas y laborales”, Universidad Rey Juan Carlos, Madrid, 2024. Accedido: 5 de febrero de 2026. [En línea]. Disponible en: https://burjcdigital.urjc.es/server/api/core/bitstreams/80cd7f76-b95f-42fe-a619-452f6968d7b8/content

[7] L. Louw y Q. Deacon, “Teaching Industrie 4.0 technologies in a learning factory through problem-based learning: Case study of a semi-automated robotic cell design”, en *Procedia Manufacturing*, Elsevier B.V., 2020, pp. 265–270. doi: 10.1016/j.promfg.2020.04.105.

[8] R. Woitsch, “Industrial Digital Environments in Action: The OMiLAB Innovation Corner”, en *Lecture Notes in Business Information Processing*, Springer Science

and Business Media Deutschland GmbH, 2020, pp. 8–22. doi: 10.1007/978-3-030-63479-7_2.

[9] M. Lanz, R. Pieters, y R. Ghabcheloo, “Learning environment for robotics education and industry-academia collaboration”, en *Procedia Manufacturing*, Elsevier B.V., 2019, pp. 79–84. doi: 10.1016/j.promfg.2019.03.013.

[10] M. Simonic *et al.*, “Modular ROS-based software architecture for reconfigurable, Industry 4.0 compatible robotic workcells”, en *2021 20th International Conference on Advanced Robotics, ICAR 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 44–51. doi: 10.1109/ICAR53236.2021.9659378.

[11] E. A. Avila, D. P. Chapa, I. D. Arenas, y C. V. Hurtado, “A Digital Twin implementation for Mobile and collaborative robot scenarios for teaching robotics based on Robot Operating System”, en *IEEE Global Engineering Education Conference, EDUCON*, IEEE Computer Society, 2022, pp. 559–564. doi: 10.1109/EDUCON52537.2022.9766583.

[12] C. Engelhardt-Nowitzki, M. Aburaia, R. Otrebski, J. Rauer, y H. Orsolits, “Research-based teaching in digital manufacturing and robotics - The digital factory at the uas technikum wien as a case example”, en *Procedia Manufacturing*, Elsevier B.V., 2020, pp. 164–170. doi: 10.1016/j.promfg.2020.04.089.

[13] J. Krushnan y F. Schrödel, “Development of a Modern, Low Cost, Lab Scale Industry 4.0 Plant for Education”, en *IFAC-PapersOnLine*, Elsevier B.V., jul. 2022, pp. 156–161. doi: 10.1016/j.ifacol.2022.09.273.

[14] S. Muñoz Restrepo, “Repotenciación del robot SCORA_ER14 de la celda de manufactura flexible del laboratorio de estudio y medición de trabajo de la Universidad Autónoma de Occidente”, Universidad Autónoma de Occidente, 2023.

[15] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3ª ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

[16] Open Source Robotics Foundation, “ROS Wiki”, ROS Wiki. Accedido: 6 de febrero de 2026. [En línea]. Disponible en: https://wiki.ros.org/

[17] GazeboSim, “Gazebo”. Accedido: 6 de febrero de 2026. [En línea]. Disponible en: https://gazebosim.org/home

[18] A. Huamán, “Canny Edge Detection”, OpenCV Documentation. Accedido: 12 de octubre de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/da/d5c/tutorial_canny_detector.html

[19] A. Huamán, “Image Moments”, OpenCV Documentation. Accedido: 12 de octubre de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/d0/d49/tutorial_moments.html

[20] J. Kamlofsky y M. L. Bergamini, “Aproximación de Formas de Objetos Digitales por Polígonos”.

[21] A. Huamán, “Hough Line Transform”, OpenCV Documentation. Accedido: 12 de octubre de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/d9/db0/tutorial_hough_lines.html

[22] G. Solano, “Contando objetos (Aplicando detección de bordes con Canny) en Python-OpenCV”, OMES-VA. Accedido: 6 de febrero de 2026. [En línea]. Disponible en: https://omes-va.com/contando-objetos-aplicando-deteccion-de-bordes-con-canny-en-python-opencv/?utm_source=chatgpt.com

[23] L. García, “Operaciones de umbral mediante inRange”. Accedido: 12 de octubre de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/da/d97/tutorial_threshold_inRange.html

[24] S. Telge, “OpenCV – Invert Mask”, GeeksforGeeks. Accedido: 6 de febrero de 2026. [En línea]. Disponible en: https://www.geeksforgeeks.org/opencv-invert-mask/?utm_source=chatgpt.com

[25] “Histogram - 4 : Histogram Backprojection”, OpenCV Documentation. Accedido: 12 de octubre de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/dc/df6/tutorial_py_histogram_backprojection.html

[26] “K-Means Clustering in OpenCV”, OpenCV Documentation. Accedido: 12 de octubre de 2025. [En línea]. Disponible en: https://docs.opencv.org/4.x/d1/d5c/tutorial_py_kmeans_opencv.html

[27] P. Kruchten, *The Rational Unified Process: An Introduction*, 3ª ed. Boston, MA: Addison-Wesley, 2004.

[28] Inc. Robotis, “OpenMANIPULATOR-X Overview”, ROBOTIS e-Manual. Accedido: 12 de septiembre de 2025. [En línea]. Disponible en: https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/

[29] Raspberry Pi Foundation, “Raspberry Pi OS – Raspberry Pi Documentation”. Accedido: 10 de octubre de 2025. [En línea]. Disponible en: https://www.raspberrypi.com/documentation/computers/os.html?utm_source=chatgpt.com

[30] Octopus, “How can I enable the camera without using raspi-config?”, Raspberry Pi Stack Exchange. Accedido: 10 de octubre de 2025. [En línea]. Disponible en: https://raspberrypi.stackexchange.com/questions/14229/how-can-i-enable-the-camera-without-using-raspi-config

[31] I. Autodesk, “Fusion 360 – 3D CAD, CAM, CAE & PCB Design Software”. Accedido: 10 de octubre de 2025. [En línea]. Disponible en: https://www.autodesk.com/products/fusion-360

[32] P. Rojas Guerrero, “Aplicación web para el seguimiento progresivo y evaluación de los resultados de aprendizaje y competencias adquiridas por estudiantes en un ambiente de aprendizaje sobre habilidades y competencias de la Industria 4.0”, Universidad del Valle- En curso.

[33] S. Wang, J. Meng, Y. Xie, L. Jiang, H. Ding, y X. Shao, “Reference training system for intelligent manufacturing talent education: platform construction and curriculum development”, *J. Intell. Manuf.*, vol. 34, nº 3, pp. 1125–1164, mar. 2023, doi: 10.1007/s10845-021-01838-4.

[34] J. Vyhmeister and G. Castane, “When Industry Meets Trustworthy AI: A Systematic Review of AI for Industry 5.0”, *arXiv preprint arXiv:2403.03061*, 2024, Accedido: 24 de octubre de 2025. [En línea]. Disponible en: https://arxiv.org/pdf/2403.03061

7. ANEXOS

En este apartado se deja la constancia de los documentos que respaldan este trabajo de grado, se alojan en el repositorio del proyecto:

Open_Manipulator/Documentos guias at main · linasalcedo01/Open_Manipulator.

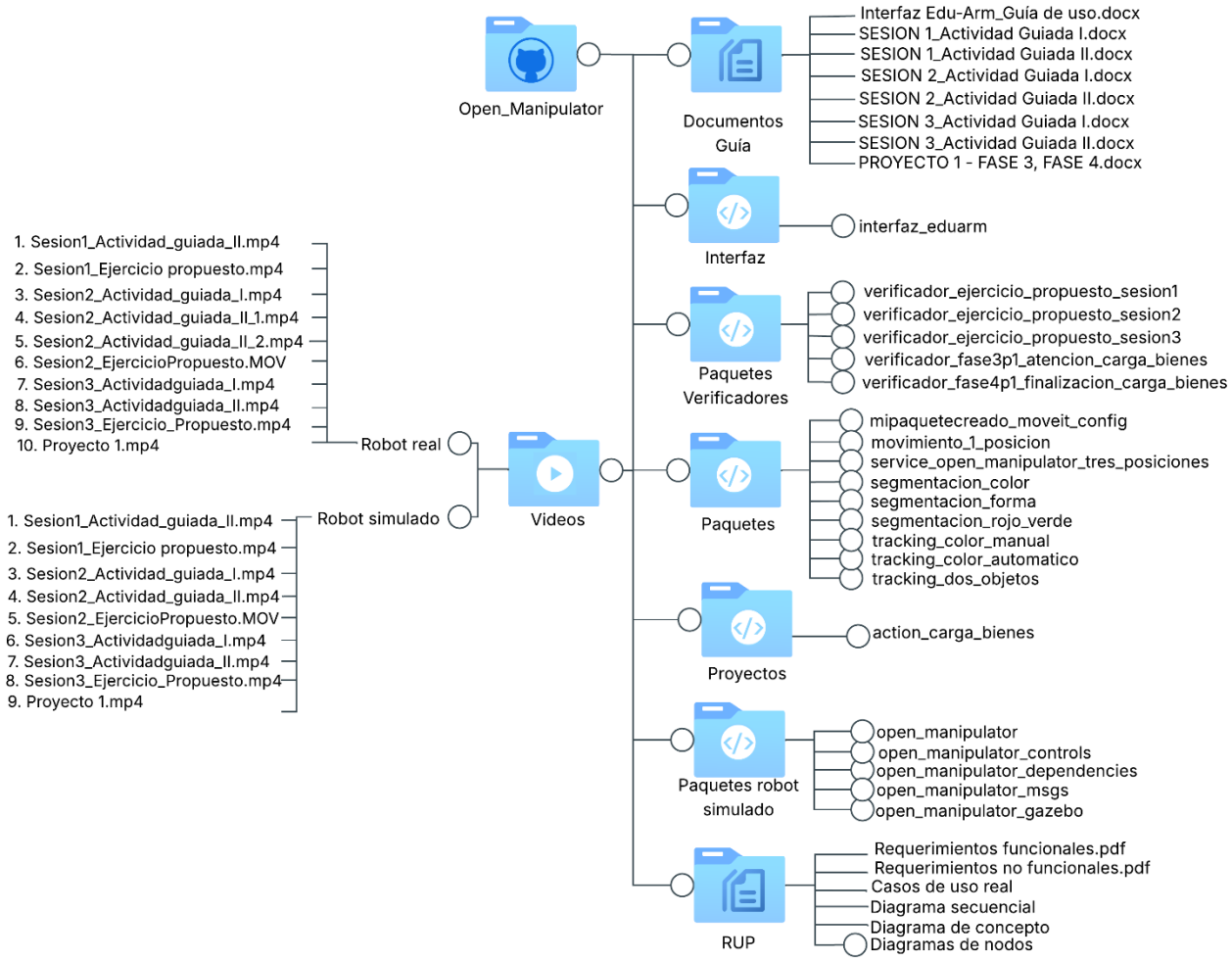


Figura 7.1 *Estructura del Repositorio GitHub*

Como se observa en la Figura 7.1, el repositorio está organizado en ocho directorios principales, los cuales se describen a continuación:

- **Documentos guía:** En este directorio se almacenan los archivos en formato *.docx* correspondientes a los documentos guía de la ruta de aprendizaje, así como la guía de uso de la interfaz Edu-Arm. En conjunto, constituyen el material necesario para orientar al estudiante durante el proceso de aprendizaje.
- **Interfaz:** Contiene el paquete de ROS utilizado para la ejecución de la interfaz Edu-Arm.
- **Paquetes verificadores:** En este directorio se incluyen los paquetes de verificación correspondientes a cada sesión de la ruta de aprendizaje, además de los dos paquetes verificadores asociados al proyecto final.
- **Paquetes:** Aquí se encuentran todos los paquetes de ROS desarrollados para cada actividad guiada y ejercicio propuesto de la ruta de aprendizaje. Los paquetes son los siguientes:

    - *mipaquetecredo_moveit_config:* Desarrollo de la actividad guiada I de la sesión 1.
    - *movimiento_1_posicion:* Desarrollo de la actividad guiada II de la sesión 1.
    - *service_open_manipulator_tres_posiciones:* Desarrollo del ejercicio propuesto de la sesión 1.
    - *segmentación_color:* Desarrollo de la actividad guiada I de la sesión 2.
    - *segmentación_forma:* Desarrollo de la actividad guiada II de la sesión 2.
    - *segmentación_rojo_verde:* Desarrollo del ejercicio propuesto de la sesión 2.
    - *tracking_color_manual:* Desarrollo de la actividad guiada I de la sesión 3.
    - *tracking_color_automatico:* Desarrollo de la actividad guiada II de la sesión 3.
    - *tracking_dos_objetos:* Desarrollo del ejercicio propuesto de la sesión 3.
- **Proyecto:** Este directorio contiene el paquete de ROS correspondiente al desarrollo del proyecto final Pickup-Master.
- **Videos:** En este directorio se almacenan los videos correspondientes a las pruebas realizadas en cada actividad guiada y ejercicio propuesto, tanto en el robot simulado como en el robot real. Ambos conjuntos incluyen los videos de ejecución de cada ejercicio a lo largo de la ruta de aprendizaje, hasta llegar al proyecto final, y están disponibles para la consulta del estudiante.
- **Paquetes robot simulado:** Contiene los paquetes necesarios para la utilización del robot simulado. Dentro de este directorio se incluye la carpeta *open_manipulator*, que agrupa todas las características de diseño asociadas al controlador del robot y al modelo en gazebo.
- **RUP:** En este directorio se almacena la documentación generada según la metodología RUP, utilizada durante el desarrollo del proyecto. Incluye los documentos de requerimientos funcionales y no funcionales, los diagramas secuenciales, de nodos y de concepto, así como los casos de uso reales.