DESARROLLO DE UN LABORATORIO REMOTO PARA LA TELE-OPERACIÓN DE KITS LEGO MINDSTORM EV3

CHRISTIAN CAMILO MERCADO RÍOS

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2025

DESARROLLO DE UN LABORATORIO REMOTO PARA LA TELE-OPERACIÓN DE KITS LEGO MINDSTORM EV3

CHRISTIAN CAMILO MERCADO RÍOS

Trabajo de grado para optar por el título de Ingeniero Electrónico

Directores
Eval Bladimir Bacca Cortes, PhD.

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2025

Contenido

Índice de figuras

Índice de tablas

RESUMEN

Durante la pandemia de COVID-19, la educación mundial enfrentó importantes desafíos debido al cierre de instituciones educativas, especialmente en áreas prácticas como la robótica. La imposibilidad de acceder físicamente a equipos utilizados durante las prácticas impulsó el uso de entornos virtuales y herramientas de simulación. Sin embargo, estas soluciones no replican completamente la experiencia interactiva y práctica de un entorno físico.

Ante este contexto, el uso del sistema operativo para robots ROS (Robot Operating System) ha ganado protagonismo en la enseñanza de la robótica, gracias a sus capacidades para simplificar el desarrollo, facilitar la simulación y habilitar el control remoto de plataformas robóticas. Su integración en múltiples laboratorios remotos demuestra su valor educativo actual.

En este marco, el presente proyecto tuvo como objetivo principal el desarrollo de un laboratorio remoto para la teleoperación de kits LEGO MINDSTORM EV3, aprovechando que la Escuela de Ingeniería Electrónica de la Universidad del Valle dispone de cinco de estos kits. Este proyecto es desarrollado usando el sistema operativo para robots ROS y tecnologías web. A partir del análisis de antecedentes locales e internacionales, se identificaron requerimientos funcionales y no funcionales necesarios para su diseño.

La arquitectura del laboratorio se basa en un PC principal con Ubuntu 16.04, que ejecuta el servidor web (desarrollado en Node.js con el framework Express), la base de datos y el entorno ROS. El usuario accede al laboratorio a través de un navegador web, desde donde puede interactuar con el robot en tiempo real. Las acciones realizadas en la interfaz son procesadas por el servidor, que utiliza rosbridge para traducir y enviar los comandos correspondientes al EV3 mediante una conexión Wi-Fi.

El laboratorio permite la configuración de sensores y motores, reserva de sesiones por bloques horarios, pruebas de sensores, recolección y descarga de datos, control teleoperado mediante teclado, ejecución de ejercicios guiados y programación visual basada en bloques.

Para validar el funcionamiento, se realizó una fase de pruebas dividida en dos etapas: primero, pruebas internas realizadas por el autor; y segundo, pruebas con estudiantes mediante una encuesta tipo Likert. Los resultados indicaron una alta aceptación: en 16 de las 19 preguntas, el 100% de los encuestados se mostró totalmente de acuerdo con la funcionalidad y usabilidad del sistema.

Palabras claves – robótica educativa, laboratorio remoto, teleoperación, ROS, educación a distancia, LEGO MINDSTORM EV3.

ABSTRACT

During the COVID-19 pandemic, global education faced major challenges due to the closure of educational institutions, particularly in hands-on areas such as robotics. The inability to physically access equipment used during practical sessions led to the adoption of virtual environments and simulation tools. However, these alternatives do not fully replicate the interactive and hands-on experience of a physical setting.

In this context, the use of the Robot Operating System (ROS) has gained relevance in robotics education due to its capabilities to simplify development, support simulation, and enable remote control of robotic platforms. Its integration in numerous remote laboratories highlights its current educational value.

This project aimed to develop a remote laboratory for the teleoperation of LEGO MINDSTORM EV3 kits, leveraging the availability of five such kits at the School of Electronic Engineering at Universidad del Valle. The system was developed using ROS and web technologies. Based on the analysis of local and international experiences, functional and non-functional requirements were identified for its design.

The laboratory architecture is based on a main PC running Ubuntu 16.04, which hosts the web server (developed in Node.js with the Express framework), the database, and the ROS environment. Users access the laboratory via a web browser, allowing real-time interaction with the robot. User actions are processed by the server, which uses rosbridge to translate and send commands to the EV3 over a Wi-Fi connection.

The platform supports sensor and motor configuration, session reservations, sensor testing, data collection and download, teleoperated control via keyboard, guided exercises, and block-based visual programming.

To validate its functionality, a two-stage testing process was conducted: internal tests by the author and student testing through a Likert-scale survey. Results indicated high user satisfaction, with 100% agreement on 16 out of 19 items, confirming the system's usability and effectiveness.

Keywords – educational robotics, remote laboratory, teleoperation, ROS, remote learning, LEGO MINDSTORM EV3.

CAPÍTULO 1.

INTRODUCCIÓN

1.INTRODUCCIÓN

La robótica es un campo multidisciplinario que se ha introducido en todos los niveles de educación en los últimos años. Matemática, física, mecánica, electrónica, computación y control automático son algunos de los campos en los que se encuentra involucrada la robótica. La robótica educativa se define como una disciplina que facilita la apropiación cognitiva, permitiendo que los estudiantes se integren fácilmente con la tecnología y mejoren sus procesos de aprendizaje [1].

La interacción de los estudiantes con robots desempeña un papel fundamental en la facilitación del aprendizaje y la asimilación de conceptos teóricos, algoritmos y técnicas [2]. Los ejercicios que proponen en los laboratorios de robótica fomentan el paradigma del aprendizaje a través de la acción, o aprendizaje activo. Este enfoque práctico permite a los estudiantes aplicar directamente los conocimientos adquiridos en un entorno real, lo que fortalece la comprensión de los conceptos teóricos y mejora la retención del material. Además, la interacción directa con robots proporciona a los estudiantes una experiencia tangible y concreta, lo que puede resultar más motivador y estimulante que enfoques puramente teóricos (Figura 1.1).

Figura 1.1. Estudiantes experimentando con los robots educativos Bee-Bot y Blue-Bot [3].

Por lo anterior, durante y después de la pandemia del COVID-19 (ante las restricciones impuestas para contener la propagación del virus) los laboratorios remotos se convirtieron en pieza clave para la impartición de conceptos de robótica, pues no solo permiten reducir costes, sino también el ahorro de tiempo al disponer de ambientes para la práctica de conceptos en cualquier horario y sin necesidad disponer físicamente de una infraestructura. En este orden de ideas, se presenta una propuesta de trabajo, que abordará el desarrollo de un laboratorio remoto.

La propuesta de proyecto de grado inicia presentando el planteamiento del problema, junto con una pregunta problema. Después, se presentan algunos antecedentes, se justifica la relevancia del proyecto y se establecen los objetivos y la metodología a seguir. Cada uno de estos componentes gira en torno a la pregunta central planteada.

1.1 PLANTEAMIENTO DEL PROBLEMA

Durante el cierre temporal de las instituciones educativas debido a la pandemia de COVID-19, se presentaron desafíos significativos para los educadores de todo el mundo al momento de impartir sus clases [4]. La transición abrupta hacia la educación a distancia planteó varios desafíos, entre ellos, asegurar la conectividad para todos los estudiantes, la implementación de nuevas tecnologías, la necesidad de invertir en la estructuración de estrategias virtuales, la comunicación efectiva con los estudiantes y, especialmente, la adaptación de las actividades educativas al entorno virtual [5]. En particular, la enseñanza de robótica se vio afectada, ya que estas actividades suelen involucrar una variedad de herramientas de aprendizaje que no estaban disponibles para muchos estudiantes durante el período en que las escuelas estuvieron cerradas [6].

En el ámbito de la robótica educativa es común utilizar kits de robótica, de los cuales a menudo se disponen unidades limitadas. Esta restricción, agravada por la imposibilidad de interactuar físicamente con los kits durante la pandemia o porque se encuentran geográficamente distantes, lo cual condujo a la decisión de descartar su uso optando en su lugar por la simulación a través de software [7].

A pesar de que la situación del COVID-19 está actualmente bajo control [8], las clases a distancia siguen siendo relevantes [9]. Esta modalidad de educación ofrece ventajas significativas, como el ahorro de tiempo y dinero, la flexibilidad en los horarios y la disponibilidad del material de estudio en cualquier momento [10], [11]. No obstante, los laboratorios virtuales a menudo no pueden proporcionar la misma experiencia que trabajar con un sistema real, ya que no transmiten la misma sensación táctil y práctica que se obtiene en un entorno físico [12]. Además, ciertas asignaturas, como la robótica, pueden ser inicialmente desafiantes de abordar de forma virtual debido a la complejidad de los conocimientos requeridos.

Considerando lo anterior, el empleo de ROS (Robotic Operating System, de sus siglas en inglés) en la enseñanza de la robótica y áreas afines ha ganado una importancia considerable, en gran parte debido a las múltiples herramientas que ofrece [13]. Estas herramientas contribuyen significativamente a reducir la complejidad inherente al mundo de la robótica. Esto se manifiesta claramente en su aplicación en entornos educativos, como lo demuestran Owen Gervais y Therese Patrosio de la Universidad de Tufts, que llevaron a cabo una implementación exitosa de ROS en colaboración con la plataforma LEGO SPIKE Prime con el propósito de impartir enseñanzas sobre conceptos vinculados a los controladores PID [14]. ROS también ha habilitado la capacidad de utilizar numerosas plataformas de manera remota, gracias a su conjunto de herramientas que facilitan este proceso. Un claro ejemplo de esto es la proliferación de implementaciones web que incorporan ROS para su uso en laboratorios a distancia [15] [16] [17].

Los kits LEGO MINDSTORM son una herramienta sumamente valiosa en la educación STEM (Ciencia, Tecnología, Ingeniería y Matemáticas), ya que proporcionan a los usuarios las herramientas esenciales para aprender robótica y les permiten diseñar y programar robots personalizados [18]. Estos continúan siendo ampliamente empleados en la actualidad, como se evidencia en proyectos contemporáneos en los que se integran junto

con la plataforma ROS [19] [20]. Este hecho pone de manifiesto su continua importancia en el campo de la ingeniería robótica contemporánea. Sin embargo, es importante destacar que pueden resultar son costosos.

En este contexto, la Escuela de Ingeniería Electrónica de la Universidad del Valle dispone de cinco kits LEGO MINDSTORM EV3, con el propósito de hacerlos más accesibles, aprovechando las ventajas del aprendizaje a distancia; y de hacer uso de ROS para simplificar el proceso de enseñanza de la robótica y aprovechar sus herramientas de conectividad en línea con el fin de brindar control remoto a los estudiantes, conduce al planteamiento de la siguiente pregunta: ¿Cómo desarrollar un laboratorio remoto para la tele-operación de kits LEGO MINDSTORM EV3 ?

1.2 JUSTIFICACION

Existen diversos tipos de herramientas para realizar experimentación remota de conceptos relacionados con la robótica móvil, entre estas destacan las herramientas de simulación que no dependen de una infraestructura física. También, es común la utilización de ROS como capa de abstracción de hardware en entornos robóticos, permitiendo centrarse en la implementación de soluciones a problemas específicos. Sin embargo, el kit Lego MINDSTORM EV3, debido a su extenso tiempo en el mercado, ha perdido soporte oficial y carece de características para una integración directa con ROS o tele-operación remota.

Este proyecto pretende como solución facilitar la práctica experimental de conceptos de robótica mediante la elaboración de un laboratorio remoto que involucre la integración de ROS en los kits LEGO MINDSTORM EV3, ofreciendo características de tele-operación, manejo de sensores y actuadores. La importancia de la marca LEGO MINDSTORM radica en la versatilidad que ofrece las piezas LEGO en combinación electrónica para obtener diferentes diseños de robots.

El propósito principal de este desarrollo es contribuir a la base de herramientas para la experimentación remota de conceptos de robótica, explorando la aplicación de ROS en los kits Lego MINDSTORM EV3. Se busca generar un nuevo recurso que ofrezca una alternativa a una infraestructura física limitada en su acceso, ofreciendo flexibilidad de horarios y reduciendo costos al utilizar equipos ya adquiridos por la Universidad del Valle y dándoles un nuevo propósito.

1.3 OBJETIVOS

OBJETIVO GENERAL

Desarrollar un laboratorio remoto para la tele-operación individual de kits LEGO MINDSTORM EV3.

OBJETIVOS ESPECÍFICOS

1. Documentar los antecedentes sobre laboratorios remotos, con el fin de extraer los requerimientos funcionales y no funcionales de la herramienta de gestión del laboratorio remoto.
2. Diseñar la herramienta de gestión del laboratorio remoto para la tele-operación de kits Lego MINDSTORM EV3.
3. Implementar la herramienta de gestión del laboratorio remoto para la tele-operación de kits Lego MINDSTORM EV3.
4. Validar la funcionalidad del laboratorio remoto considerando los requerimientos funcionales planteados inicialmente.

1.4 SOLUCIÓN PROPUESTA

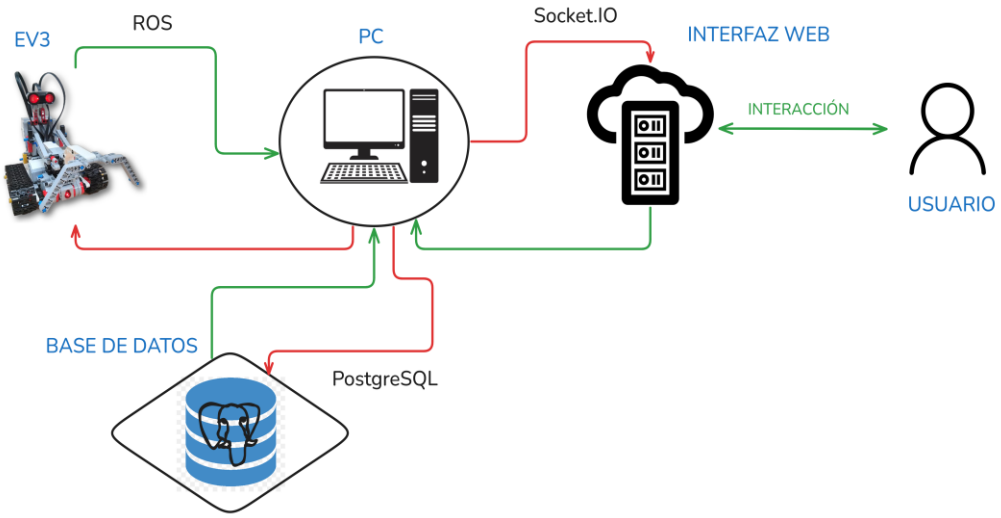


Figura 1.2. Estructura del laboratorio.

Para la solución se planteó un sistema compuesto por el robot Lego MINDSTORM EV3, un PC principal con sistema operativo Ubuntu 16.04 y una solución de software basada en la web que funciona como laboratorio remoto, encargado de gestionar y manipular toda la información.

La comunicación entre el PC y el robot se realizará mediante el sistema operativo para robots ROS, utilizando una conexión inalámbrica (Wi-Fi). A su vez, ROS se comunica con el servidor web vía WebSockets, que se encarga de traducir las acciones del usuario y enviarlas a ROS, para que luego sean ejecutadas en el robot (Figura 1.2).

El laboratorio remoto estará en capacidad de brindar al usuario opciones como la configuración de sensores, el control teleoperado del robot y la realización de ejercicios.

1.5 ESTRUCTURA DEL DOCUMENTO

El documento de este trabajo de grado inicia con un capítulo introductorio en el que se presenta la problemática abordada, se contextualiza la situación y se expone la solución propuesta.

El segundo capítulo, **Antecedentes y marco teórico**, ofrece una revisión de los términos utilizados, así como de trabajos previos que sirvieron de base para el desarrollo del laboratorio.

El capítulo correspondiente al **desarrollo del laboratorio remoto para la teleoperación de kits LEGO MINDSTORMS EV3** se centra en los requerimientos del sistema, el diagrama de clases y el funcionamiento general de la aplicación.

Finalmente, en el capítulo de **pruebas de funcionamiento**, se describen una serie de pruebas diseñadas para evaluar el cumplimiento de los requerimientos establecidos, seguido de un análisis de los resultados obtenidos.

CAPÍTULO 2.

ANTECEDENTES Y MARCO TEÓRICO

2. ANTECEDENTES Y MARCO TEÓRICO

2.1 INTRODUCCIÓN

En este capítulo se presenta trabajos previos relacionados con el presente proyecto, los cuales fueron de utilidad para identificar características relevantes que contribuyeron al diseño y construcción del laboratorio.

Además, se incluyen un resumen de la información consultada y utilizada durante el desarrollo del laboratorio, incluyendo temáticas relacionadas con el hardware (LEGO MINDSTORM EV3) y software empleado (ROS), así como la metodología de desarrollo aplicada.

2.2 ANTECEDENTES

En esta sección, se presentan trabajos que se enfocan en proyectos educativos que hacen uso de kits en mayor medida de LEGO MINDSTORM o ROS. Sin embargo, estos proyectos están principalmente centrados en propuestas de laboratorios remotos. La finalidad de este apartado es recopilar información relevante para el desarrollo de la solución propuesta, además de analizar diversas implementaciones existentes y las herramientas tecnológicas aplicadas en cada una.

En el contexto de la investigación actual, se destacaron 9 documentos en el análisis de antecedentes relacionados. La búsqueda se enfocó en trabajos realizados a partir del año 2017, considerando la creciente importancia de los laboratorios remotos debido a la pandemia del COVID-19 y a la rápida evolución de nuevas tecnologías.

Criterios de comparación de antecedentes

Robot: Describe qué tipo de robot se utiliza o con cuál robot se lleva a cabo las pruebas del laboratorio.

ROS: Menciona si el laboratorio hace uso de ROS como medio de comunicación con el robot.

Lenguaje de programación: Menciona los lenguajes de programación predominantes en el desarrollo del laboratorio.

Visualización del robot: Destaca el método a través del cual se presenta el robot al usuario en la interfaz del laboratorio.

Permite programación: Indica si el laboratorio ofrece la posibilidad de programar directamente el robot mediante algún tipo de lenguaje de programación.

Ejercicios propuestos: Informa si la plataforma cuenta con ejercicios de ejemplo disponibles para realizar o una sección de guía y/o ayuda.

Base de usuarios: Señala que el laboratorio cuenta con un sistema de autenticación en la plataforma mediante una base de usuarios.

Almacenamiento de datos: Informa sobre la capacidad de la plataforma para guardar datos experimentales, ya sea de manera local o en una base de datos.

Sensores: Indica cuáles son los sensores que son funcionales en la plataforma.

Actuadores: Indica cuáles son los actuadores que son funcionales en la plataforma.

Tabla 2.1. Tabla comparativa de implementación de laboratorios remotos.

Ref.	Robot	ROS	Lenguajes de programación	Visualización del robot	Permite programación	Ejercicios propuestos	Base de usuarios	Almacenamiento de datos	Sensores	Actuadores
[14]	Simulación	SI	Python	Modelo 3D	SI	SI	NO	NO	Laser, ultrasónico	Motores
[17]	UR3	SI	JavaScript	Modelo 3D	NO	NO	SI	SI	IMU, enconders	Servomotores
[19]	Lego Minstorm EV3, General	SI	MATLAB Language	Cámara	NO	NO	NO	NO	Ultrasónico, Infrarrojo	Motores
[21]	UVBots2	NO	Python, JavaScript	*N/A*	SI	SI	SI	NO	Proximidad, línea, IMU	Motores, zumbador
[22]	4R EIA	NO	Python, JavaScript	Modelo 3D	NO	SI	SI	SI	IMU, enconders	Servomotores
[23]	Lego Mindstorm EV3	SI	Python, JavaScript	*N/A*	SI	NO	NO	NO	Ultrasónico, color, luz, tacto, IMU	Motores
[24]	Lego Minstorm EV3	NO	Python, JavaScript	Cámara	SI	NO	NO	NO	Ultrasónico, color, luz, IMU	Motores
[25]	Khepera IV	NO	Java	Cámara, Modelo 2D	SI	NO	SI	NO	IMU, infrarrojos, ultrasónicos	Motores
[26]	Dobot Magician	NO	JavaScript, Python	Cámara	SI	NO	NO	SI	IMU, enconders	Servomotores
[27]	Roboton, General	SI	JavaScript	Cámara	SI	SI	SI	SI	Cámara, encoders,	Motores
[28]	Simulación	SI	Python, JavaScript	Modelo 3D	SI	NO	NO	SI	Múltiples	Múltiples

N/A: No aplica, no especifica, no se utilizó.

Ref.	Robot	ROS	Lenguajes de programación	Visualización del robot	Permite programación	Ejercicios propuestos	Base de usuarios	Almacenamiento de datos	Sensores	Actuadores
[29]	JetBot	SI	JavaScript	Cámara	SI	NO	SI	SI	Laser, encoders	Motores
[30]	Pioneer P3-DX	NO	PHP	Cámara	NO	SI	SI	SI	Encoders	Motores
[31]	Simulación/ General	SI	JavaScript	Cámara, Modelo 3D	SI	SI	SI	SI	Múltiples	Múltiples

N/A: No aplica, no especifica, no se utilizó.

2.2.1 ANTECEDENTES LOCALES

En la base de datos de la Universidad del Valle, se encontró el siguiente trabajo que aborda el proyecto de un laboratorio remoto.

En [21], los autores crearon una plataforma para promover la interacción con dispositivos físicos mediante la programación de diversos robots, específicamente los UVBots2. La plataforma permite la interacción a varios niveles de conocimiento de programación (ANSI-C, Python) a través de una interfaz web desarrollada en Python con el framework Django. Los robots se comunican con el usuario a través de RS-232, y la herramienta de gestión proporcionada incluye ejercicios de prueba, soporte para múltiples sesiones, programación por bloques y acceso a diversos sensores y actuadores.

En [22], los autores han desarrollado un laboratorio virtual para el control del manipulador 4R EIA. La interfaz de usuario se crea mediante un sistema de gestión de aprendizaje, facilitando la creación y administración de entornos de aprendizaje en línea de manera automatizada. El manipulador se comunica con un servidor a través de la tarjeta de desarrollo Arduino, utilizando tecnologías como JavaScript para la interfaz, Django para el backend, MATLAB para la simulación mediante Simulink y WebSocket para transmitir la simulación a la plataforma. La interfaz permite realizar solicitudes HTTP POST para manipular la posición del manipulador, llevarlo a la posición inicial, cambiar el ángulo de vista de la simulación, generar trayectorias, manejo de usuarios e instructivo de uso.

2.2.2 ANTECEDENTES INTERNACIONALES

En [14], los autores detallan la última actualización de la herramienta Robotics Academy, que es utilizada en el curso abierto Intelligent Robotics. Esta actualización presenta nuevos complementos que se han adaptado a las tecnologías actuales. La versión más reciente de esta herramienta incluye una interfaz web que permite a los usuarios interactuar con una serie de ejercicios mediante la escritura de código para controlar simulaciones basadas en Gazebo. La instalación se realiza a través de una imagen de Docker que contiene toda la configuración necesaria para la interacción web, incluyendo el servidor web y la simulación de Gazebo.

El núcleo de esta plataforma web está basado en un servidor web Django. Desde este servidor, se presentan los diversos ejercicios disponibles para que los usuarios elijan cuál desean resolver. Una vez que un usuario selecciona un ejercicio, el servidor web mostrará la plantilla correspondiente a ese ejercicio, y la simulación se llevará a cabo en un contenedor Docker. Estos ejercicios se centran en algoritmos de robótica, y las soluciones involucran el uso de controladores PID, navegación local y global, así como algoritmos de cobertura aleatoria y óptima. Los ejercicios aprovechan los sensores y actuadores incorporados en los robots, los cuales son replicados de manera simulada durante la ejecución de las actividades. La herramienta de gestión del laboratorio permite tanto cargar como guardar el código realizado durante las sesiones.

En [17], los autores presentan un laboratorio virtual de robótica en el que se desarrolla una plataforma web segura de acceso abierto para el control remoto de manipuladores robóticos, validada mediante el control remoto de un manipulador UR3 real. Esta plataforma permite a estudiantes e investigadores utilizar una herramienta educativa que difiere de los simuladores

robóticos tradicionales al ofrecer una experiencia virtual que conecta manipuladores reales en todo el mundo a través de Internet.

La plataforma hace uso de una aplicación web fácil de usar con visualización 3D del modelo robótico, botones interactivos y comandos para la manipulación del robot virtualizado, ejecución en el robot real y una página de inicio de autenticación de usuario. Después de visualizar la posición inicial del robot, el usuario puede realizar cambios deseados en esa posición para lograr un objetivo específico en el robot virtualizado a través de un marcador interactivo que permite mover los ejes x, y y z, o seleccionar el valor de cada articulación. La interfaz también incluye botones de acción como "reiniciar" para volver a la posición inicial y "planificar y ejecutar" para llevar a cabo el movimiento a la posición deseada enviando las trayectorias de planificación desde la computadora del servidor (basado en ROS) al robot real.

En [19] se presenta una plataforma educativa basada en Matlab y ROS que permite interactuar en tiempo real con robots de cualquier tipo. Se propone una generalización de una plataforma anterior basada en Matlab/Simulink/Lego EV3 que estaba limitada a este kit. El artículo presenta la migración de la parte de Simulink a la parte de ROS para generalizar la plataforma y muestra pruebas experimentales sobre cómo utilizarla. Para comunicarse con los robots se hace uso de la librería que ROS ha adaptado para MATLAB. El PC donde corre MATLAB se convierte entonces en el ROS MASTER (servidor) y el robot tiene que tener configurado la IP de este para realizar la comunicación. Desde aquí la comunicación se realiza y utilizando las herramientas de ROS y el procesamiento de los datos se realiza con MATLAB.

En [23], se propone el desarrollo de una plataforma web que permite programar acciones en el LEGO MINDSTORM EV3 mediante el uso de bloques, para el uso de este en un laboratorio de robótica. Este planteamiento se basa en el framework de ROS para establecer una comunicación efectiva con el kit. En el desarrollo de este proyecto, se utilizó el paquete de ROS denominado robot_blockly, que proporciona una interfaz web y herramientas de programación basadas en bloques. Además, se aprovechó la imagen del sistema operativo ev3dev con soporte ROS para el LEGO EV3. La herramienta de gestión de este laboratorio permite el manejo de sensores, actuadores del EV3 a la vez que permite el almacenamiento y carga de código.

En [24], los autores detallan la última actualización de la herramienta Robotics Academy, que es utilizada en el curso abierto Intelligent Robotics. Esta actualización presenta nuevos complementos que se han adaptado a las tecnologías actuales. La versión más reciente de esta herramienta incluye una interfaz web que permite a los usuarios interactuar con una serie de ejercicios mediante la escritura de código para controlar simulaciones basadas en Gazebo. La instalación se realiza a través de una imagen de Docker que contiene toda la configuración necesaria para la interacción web, incluyendo el servidor web y la simulación de Gazebo.

La clave de este proyecto radica en el aprovechamiento de las capacidades de las herramientas web de Google para ejecutar scripts personalizados. A través de estos scripts, se realizan solicitudes HTTP y HTTPS al servidor que se ejecuta en el kit de LEGO. Esto posibilita que varios usuarios trabajen de manera colaborativa, ya sea en el mismo entorno donde se encuentra el kit o en ubicaciones remotas. En el segundo caso, se utiliza la herramienta Google Meet para visualizar el kit a través de una cámara. Esta solución ofrece a los usuarios una amplia gama de

opciones para controlar el kit, desde mover los motores hasta leer sensores y programar en Python.

En [25], los autores presentan el desarrollo de una plataforma fácil de usar diseñada para impartir conceptos de control de robots móviles (Control de posición, seguimiento de trayectorias, seguimiento de rutas y evitación de obstáculos). En este laboratorio hace uso del robot Khepera IV al que le han proporcionado su propio entorno de pruebas equipado con cámaras para la visualización y cálculo de su ubicación mediante el software SwisTrack. La comunicación con el robot se establece a través de Wi-Fi, y se conecta a un servidor en una PC con Linux. Esta PC se encarga de recibir la información del estudiante y transmitírsela al robot, así como de procesar la posición utilizando los datos de las cámaras. Los estudiantes hacen control del robot a través de una interfaz realizada con Easy Java Simulations (EJS) donde se realiza todo el monitoreo del robot y se envía la información con las trayectorias a seguir.

En [26], se presenta un laboratorio remoto desarrollado en la Universidad Complutense de Madrid (UCM) que destaca por ofrecer acceso a distancia al robot educativo Dobot Magician. El laboratorio remoto dispone de una interfaz que permite realizar control cartesiano, así como control articular o programar directamente el brazo robótico.

El servidor del laboratorio hace uso ReNoLabs está programado en Node.js, mientras que el software de control utiliza Python, y la interfaz web se basa en Easy JavaScript Simulations (EJsS). Para la visualización del robot se provee al laboratorio de dos cámaras laterales - una a cada lado del Dobot-, para que el alumno pueda seleccionar la más adecuada en cada caso. La herramienta de gestión de este laboratorio permite la gestión de usuarios, almacenar datos experimentales y contiene una sección de guía y ayuda.

En [27], el autor expone un entorno de aprendizaje de ROS disponible en línea, que permite completar tareas tanto en robots físicos como en simulaciones. Este laboratorio web utiliza VNC (Virtual Network Computing) y Docker para ofrecer estaciones de trabajo de escritorio en el navegador, facilitando así la experimentación con ROS sin necesidad de configurar ningún software localmente. Su funcionamiento se basa en el despliegue de contenedores Docker que contienen imágenes de Ubuntu con todas las herramientas y configuraciones necesarias de ROS. El acceso al sistema se realiza a través de VNC después de que el usuario se autentica y se verifica que ha reservado el espacio en el laboratorio. Además, se proporciona la visualización del robot a través de una cámara.

En [28], el autor presenta el desarrollo de un entorno de simulación en tiempo real para aplicaciones robóticas, que combina el sistema ROS con el motor gráfico de Unreal Engine 5. Se logra la integración mediante una API en Python que utiliza ROSLIBPY para la comunicación en tiempo real a través de websockets. El software resultante, denominado UROS API, se ejecuta en un servidor ASGI (Asynchronous Server Gateway Interface) y facilita la gestión de comunicaciones, incluyendo con clientes web. Además, se desarrolla una aplicación web, UROS APP, con React.js y Vite, que interactúa gráficamente con la API. UROS API almacena datos en una base de datos NoSQL como MongoDB, y UROS APP los visualiza usando Chart.js. El entorno utiliza recursos y modelos 3D disponibles en línea y en el marketplace de Unreal Engine. Se realizan pruebas de comunicación y control en un robot sumo de dos ruedas y en un robot industrial con 6 grados de libertad.

En [29], Se presenta un laboratorio remoto que utiliza el robot JeBot para llevar a cabo prácticas de robótica de forma local o remota. Este laboratorio simplifica la conexión a una instancia de Ubuntu con ROS y sus herramientas a través de un VNC, lo que brinda un control total sobre las acciones a realizar con el robot, como la programación directa y la simulación mediante Gazebo. A través de una herramienta de gestión, se concede acceso a esta instancia una vez que el usuario ha sido autenticado y ha reservado previamente su uso. La visualización del robot se realiza mediante dos cámaras conectadas a servomotores.

En [30], Se desarrolla un entorno de laboratorio remoto basado en la web para el control y monitoreo en tiempo real de un robot móvil en un entorno interior. En este laboratorio, un objetivo virtual y obstáculos virtuales están ubicados en cualquier lugar de la imagen de video tomada cámara. La meta, es que, el robot llegue al objetivo virtual evitando los obstáculos virtuales utilizando la ruta más corta.

El laboratorio utiliza un servidor Apache para la comunicación a través del protocolo HTTP con la herramienta de gestión. Además, se emplea un servidor de base de datos PostgreSQL para almacenar información tanto del servidor como de los usuarios.

The Construct Robotics Institute [31] es una plataforma comercial de aprendizaje en línea especializada en ROS y robótica, con una oferta de más de 40 cursos que abarcan desde los fundamentos básicos de ROS hasta la programación de vehículos autónomos. Incluso en su versión más básica, esta plataforma proporciona la capacidad de trabajar con ROS a través de simulación. Una vez seleccionado el tema a estudiar, se guía al usuario a través de una serie de pasos para alcanzar sus objetivos específicos. El entorno de aprendizaje incluye una terminal para ingresar comandos, una ventana de simulación para visualizar el comportamiento del robot y un editor de texto para desarrollar código.

En los antecedentes mencionados, se destaca el uso predominante de JavaScript y Python como los principales lenguajes de programación. Además, se opta por la programación directa de los robots y el control a través de parámetros. En lo que respecta a las opciones de visualización del robot, hay preferencia por el uso de cámaras. La mayoría de los robots empleados en estos laboratorios son manipuladores, por lo que su enfoque se centra en la planificación de trayectorias.

2.3 MARCO TEÓRICO

En esta sección, se abordan y definen conceptos clave fundamentales para comprender la propuesta de trabajo de grado. Estos elementos proporcionan al lector una contextualización integral de la investigación, dotándolo de los conocimientos necesarios para abordar de manera efectiva la temática propuesta.

2.3.1 LEGO MINDSTORM

LEGO MINDSTORM es una línea de juguetes de robótica diseñada para niños fabricada por la empresa LEGO. Estos kits son ampliamente utilizados en la educación en distintos niveles debido

a la versatilidad que ofrece su característica de modularidad. Esta capacidad permite modificar físicamente la estructura del robot mediante piezas ensamblables, que incluyen sensores y actuadores con lo que es posible interactuar y definir su comportamiento mediante algoritmos especificados a través de programación visual o basada en texto. Estos conjuntos, conocidos como bloques o bricks son programados a través de las herramientas suministradas por la compañía, aunque también existen entornos de desarrollo de terceros que admiten diversos lenguajes [32].

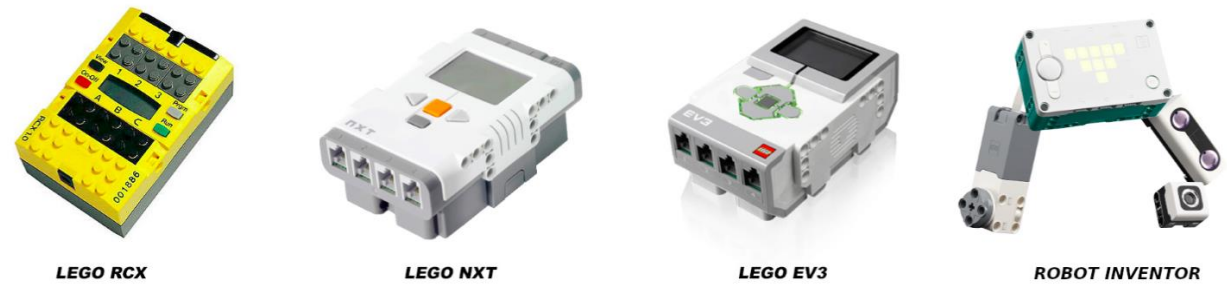


Figura 2.1. Todas las generaciones de bricks de la línea LEGO MINDSTORM..

Diferentes kits LEGO MINDSTORM han evolucionado a lo largo de cuatro generaciones con diseños distintos (Figura 2.1):

- **RCX (Primera generación):** Lanzado en septiembre de 1998, presentaba un diseño simple con un ladrillo amarillo de pantalla pequeña, tres entradas y tres salidas. Reflejaba la filosofía de LEGO centrada en la versatilidad y creatividad con un conjunto limitado de elementos.

- **NXT (Segunda generación):** Introducida en 2006, experimentó un cambio radical en diseño y electrónica. Se ampliaron las entradas y se incorporaron nuevos sensores. La comunicación con la computadora se realizaba mediante comunicación serial (USB) y Bluetooth.

- **EV3 (Tercera generación):** Lanzada en julio de 2013, incluye un bloque con 16 MB de memoria flash y 64 MB de memoria RAM, con una ranura para tarjetas mini SD de hasta 32 GB. Agrega opciones de conectividad como USB, Bluetooth y wifi a través del puerto USB. Destaca por el cambio a un sistema operativo abierto (Linux), marcando diferencia respecto a versiones anteriores.

- **ROBOT INVENTOR (Cuarta generación):** Lanzado en el otoño del 2022, es el último set de MINDSTORM disponible comercialmente antes de la discontinuación de esta línea de productos. Incluye cuatro motores medianos, dos sensores (sensor de distancia y sensor de color/luz), giroscopio de seis ejes, un acelerómetro, soporte para controles y control mediante dispositivos móviles. Además, cuenta con más de 902 elementos Lego Technic.

2.3.2 ROS (Robotic Operating System)

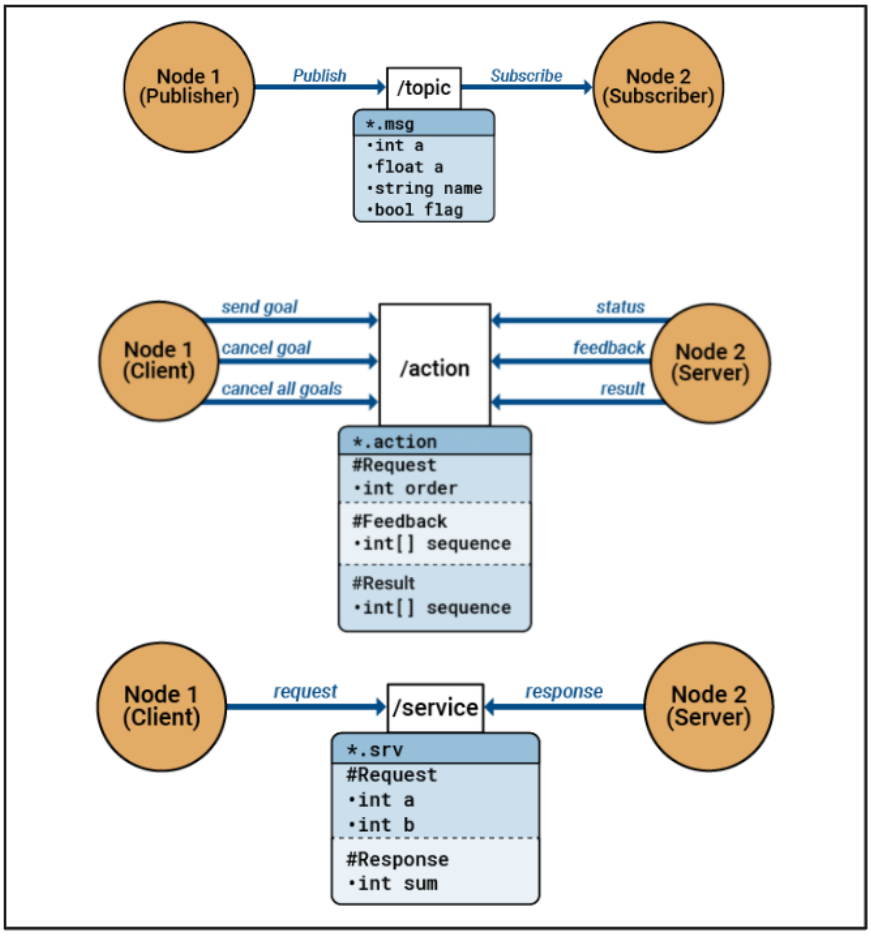


Figura 2.2. Mecanismos de comunicación: Tópicos, acciones y servicios.

ROS es un framework de código abierto ampliamente utilizado en el desarrollo de software para robots. Se define oficialmente como un conjunto de bibliotecas de software y herramientas que facilitan la creación de aplicaciones robóticas, desde controladores hasta algoritmos avanzados y poderosas herramientas de desarrollo [33]. Este sistema proporciona diversas funciones y servicios que facilitan la comunicación entre los distintos componentes de un sistema robótico, incluyendo sensores, actuadores, algoritmos de percepción y planificación, así como controladores.

Su diseño se centra en la modularidad y flexibilidad, permitiendo a los desarrolladores utilizar y combinar componentes según las necesidades específicas de sus aplicaciones. Una característica distintiva de ROS es su arquitectura basada en nodos. Estos nodos son unidades de ejecución independientes que pueden comunicarse entre sí mediante mensajes. Estos mensajes contienen información sobre datos sensoriales, comandos de actuación y otros tipos de información relevante para el funcionamiento y aplicaciones del robot. Cada nodo realiza una tarea específica y se comunica con otros nodos a través de tópicos, acciones y servicios.

En ROS, la comunicación nodal se realiza mediante tres mecanismos principales: tópicos, acciones y servicios (Figura 2.2).

- **Tópicos:** Canales unidireccionales para la transmisión eficiente de mensajes. Ideales para datos en tiempo real, como la obtención de información de sensores o el envío de instrucciones para actuadores.

- **Acciones:** Permiten tareas complejas con interacciones continuas. Compuestas por una meta, retroalimentación y resultado. Adecuadas para ejecutar acciones que implican interacciones sofisticadas entre los componentes de un sistema.

- **Servicios:** Facilitan una comunicación de solicitud-respuesta mediante una dinámica de servidor-cliente. Utilizados para tareas que no requieren una interacción continua, como la configuración de parámetros o la ejecución de cálculos especializados en respuesta a solicitudes puntuales.

2.3.3 EDUCACIÓN REMOTA

La educación remota o a distancia es un modelo educativo que permite a los estudiantes acceder a contenidos y participar en actividades académicas sin la necesidad de estar físicamente presentes en un entorno educativo tradicional. Esta modalidad utiliza tecnologías de la información y la comunicación para facilitar la transmisión de conocimientos, la interacción entre estudiantes y docentes, así como la realización de evaluaciones, todo ello a través de plataformas virtuales [34].

Características Clave:

- **Flexibilidad Temporal:** Permite acceso y tareas según la conveniencia del participante.
- **Accesibilidad Geográfica:** Elimina barreras, permitiendo la participación global.
- **Tecnologías de la Información:** Se apoya en plataformas en línea, videoconferencias y foros para facilitar la interacción.
- **Personalización del Aprendizaje:** Adapta el ritmo de estudio a las necesidades individuales.

2.3.4 LABORATORIO REMOTO

Los laboratorios remotos son entornos experimentales accesibles en línea que permiten a los usuarios realizar pruebas y experimentos sin estar físicamente presentes en el lugar de los equipos [35]. A través de interfaces en línea, los usuarios controlan y supervisan las operaciones. Estos laboratorios abarcan diversas disciplinas, como física, química, ingeniería y robótica. Su infraestructura incluye instrumentación, sensores y actuadores conectados a Internet para realizar experimentos en tiempo real. La principal ventaja es su accesibilidad global, brindando oportunidades de aprendizaje y colaboración a estudiantes, investigadores y profesionales, y democratizando el acceso a equipos costosos o ubicados en lugares remotos.

Los laboratorios remotos suelen contar con las siguientes características:

- **Acceso Remoto:** Permite la entrada a equipos y recursos del laboratorio a través de Internet desde ubicaciones distantes.
- **Interfaz en Línea:** Brinda una interfaz virtual que capacita a los usuarios para supervisar y controlar operaciones a través de dispositivos conectados.
- **Instrumentación Conectada:** Incluye instrumentación, sensores y actuadores conectados a la red para su manipulación remota.
- **Tiempo Real:** Facilita la ejecución de experimentos y la recopilación de datos en tiempo real, proporcionando una experiencia casi presencial.
- **Globalmente Accesible:** Disponible para estudiantes, investigadores o profesionales sin restricciones geográficas, fomentando la colaboración global.
- **Flexibilidad Horaria:** Permite realizar experimentos en horarios convenientes, ofreciendo flexibilidad temporal.
- **Seguridad y Control:** Implementa medidas de seguridad para proteger equipos y datos, con controles de acceso.
- **Amplia Variedad de Experimentos:** Ofrece una diversidad de experimentos, desde demostraciones simples hasta proyectos complejos, realizables de forma remota.

2.3.5 METODOLOGÍA SCRUM

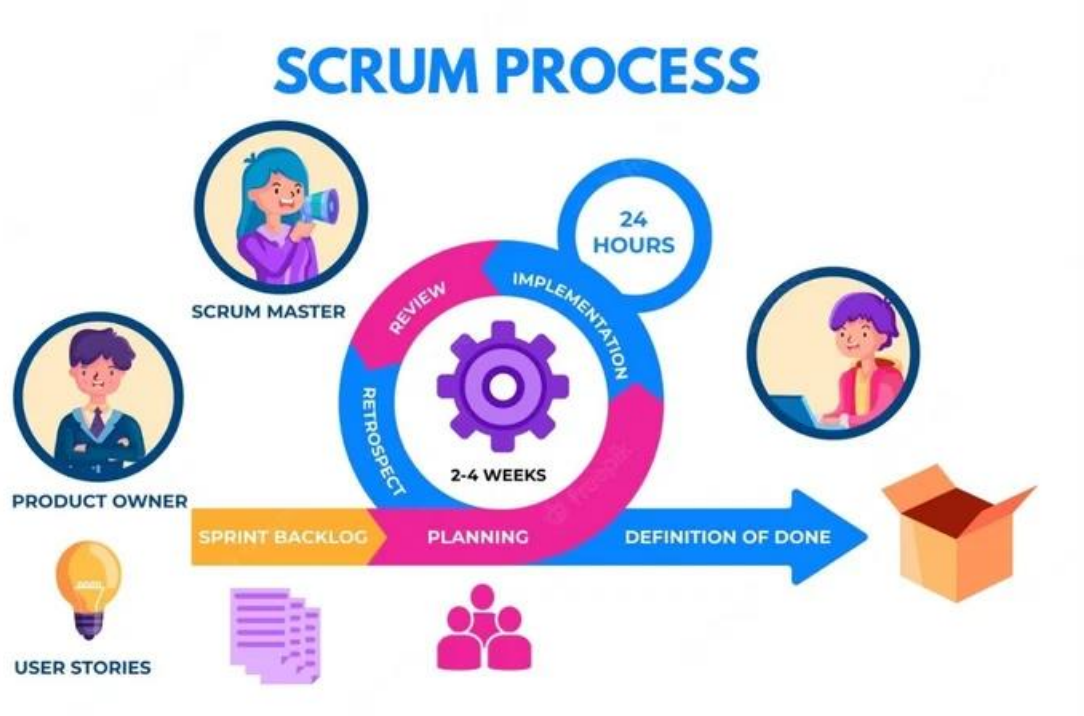


Figura 2.3. Proceso de la metodología SCRUM.

Scrum es un marco de trabajo ágil que se basa en principios de transparencia, inspección y adaptación para fomentar la colaboración y la entrega incremental de valor [36]. En Scrum, los proyectos se dividen en ciclos de trabajo llamados sprints, periodos fijos y cortos (generalmente de 1 a 4 semanas). Cada sprint incluye planificación, desarrollo y revisión para obtener retroalimentación (Figura 2.3).

El objetivo principal de Scrum es posibilitar una entrega temprana y continua de valor, fomentando la adaptación a medida que se obtiene retroalimentación del cliente y se realizan mejoras en el producto. Para ello, los equipos de Scrum son auto-organizados y multifuncionales, por lo que cada uno es responsable de unas tareas determinadas y de terminarlas en los tiempos acordados.

Los roles en el equipo Scrum

- **Product Owner:** Es la persona que decide qué cosas hay que hacer y en qué orden, hablando siempre con el cliente para saber qué es lo más importante. Se encarga de hacer una lista de tareas que el equipo irá completando poco a poco.

- **Scrum Master:** Ayuda al equipo a trabajar mejor y a seguir las reglas de esta metodología de trabajo. Si aparece algún problema, busca cómo resolverlo.

- **Equipo de Desarrollo:** Son los que hacen el trabajo: programar, diseñar, construir, probar, etc. Se organizan entre ellos, deciden cuánto tiempo les tomará cada tarea y se ayudan mutuamente para terminar a tiempo.

Artefactos de Scrum

Los artefactos de Scrum son herramientas para maximizar la transparencia del proyecto, asegurando que todos los miembros del equipo compartan una visión común del progreso.

- **Product Backlog**: Es la lista completa de tareas del proyecto, organizada por prioridades. El Product Owner es quien la gestiona, en coordinación con el cliente.

- **Sprint Backlog**: Es el conjunto de tareas seleccionadas del Product Backlog que el equipo desarrollará durante un Sprint, junto con un plan para completarlas.

- **Incremento**: Es el resultado final del trabajo realizado en el Sprint, sumado al valor acumulado de sprints anteriores. Representa una versión utilizable del producto.

2.3.6 DOCKER Y CONTENEDORES

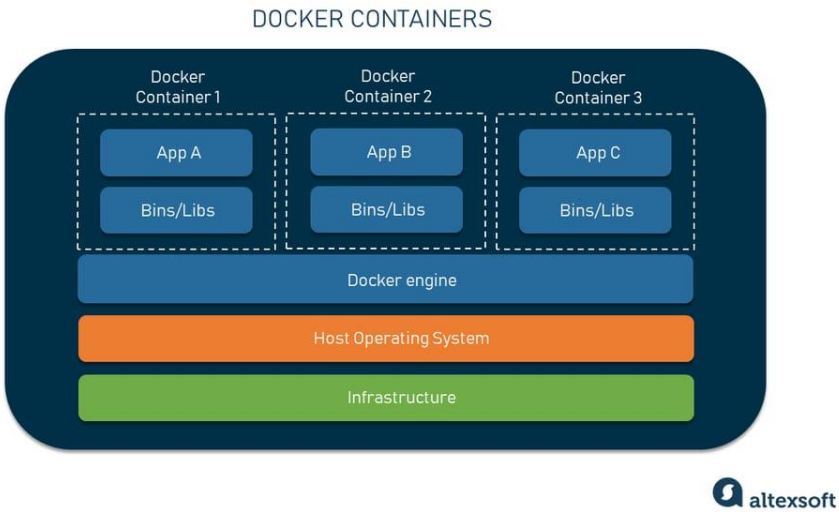


Figura 2.4. Contenedores de Docker.

Docker es una plataforma de software que permite crear, probar e implementar aplicaciones de forma rápida y eficiente. Utiliza una tecnología basada en contenedores, que encapsulan todo lo necesario para ejecutar una aplicación: código, bibliotecas, herramientas del sistema y ejecutables (Figura 2.4). Estas unidades estandarizadas, llamadas contenedores, garantizan que el software se comporte de manera consistente en cualquier entorno [37].

Con Docker los desarrolladores pueden crear entornos aislados que replican con precisión la configuración de producción, lo que facilita el desarrollo, las pruebas y el despliegue de aplicaciones. Además, al empaquetar todos los componentes necesarios en una sola imagen, se simplifica el proceso de distribución e instalación.

Docker también permite escalar e implementar aplicaciones de forma ágil en diferentes plataformas, con la certeza de que el código se ejecutará correctamente sin importar el entorno subyacente.

En el caso de ROS, su funcionamiento depende de versiones específicas de ciertos sistemas operativos. Por ejemplo, la mayoría de las distribuciones de ROS están diseñadas para funcionar únicamente con versiones específicas de Ubuntu. Esto representa un problema cuando se necesita trabajar con versiones antiguas de ROS, ya que suelen depender de sistemas operativos que han quedado sin soporte y que presentan problemas de compatibilidad con hardware moderno.

El uso de Docker resuelve este inconveniente al permitir empaquetar una versión específica de Ubuntu junto con ROS en un contenedor. De esta manera, es posible ejecutar versiones antiguas de ROS en cualquier sistema operativo moderno, sin preocuparse por incompatibilidades. Además, al contener todo en una única imagen, se eliminan los problemas de instalación y configuración, convirtiendo la aplicación en una solución prácticamente plug and play.

2.4 OBSERVACIONES FINALES

En la revisión de antecedentes se seleccionaron laboratorios relacionados con el trabajo desarrollado. En este caso, se priorizaron aquellos que hicieran uso de ROS, kits de Lego o algún tipo de robot, y que estuvieran mayormente basados en la web.

Se establecieron criterios de comparación basados en características relevantes para el proyecto, con el objetivo de identificar similitudes y diferencias entre las propuestas analizadas. Entre estos criterios se incluyeron: el tipo de robot utilizado, el uso de ROS, el lenguaje de programación, la forma de visualización del robot, la metodología de programación, los ejercicios propuestos, el uso de bases de datos, así como los sensores y actuadores integrados en el robot.

En los antecedentes locales se seleccionaron dos trabajos relevantes. El primero proviene de la Universidad del Valle, donde se utilizó el robot UVBots2, el cual es controlado a través de una interfaz web desarrollada en Python. El segundo antecedente corresponde a un proyecto de la Universidad EIA, basado en un manipulador robótico controlado también mediante una plataforma web, enfocado en la generación de trayectorias.

En cuanto a los antecedentes internacionales, hubo una mayor diversidad tecnológica en los antecedentes escogidos. Se analizaron proyectos que incorporan simulación, el uso de contenedores como Docker, la implementación de ROS, entornos programables y plataformas comerciales diseñadas específicamente para la enseñanza y el aprendizaje de la robótica.

La revisión de trabajos relacionados con laboratorios remotos ha permitido identificar aspectos clave para el desarrollo e implementación del laboratorio propuesto. Se reconocieron tecnologías y herramientas comúnmente utilizadas, destacando especialmente aquellas basadas en entornos web. Además, se tomaron en cuenta ideas para la implementación de ejercicios prácticos en el laboratorio, así como para la visualización remota del robot.

También se abordaron conceptos fundamentales sobre el uso del robot seleccionado y el framework ROS, lo que permitió contextualizar su integración. Además, se introdujo la tecnología Docker, basada en contenedores, que permite ejecutar aplicaciones en entornos aislados y reproducibles, con todas las dependencias necesarias incluidas. Finalmente, se describió la metodología que se empleará durante el desarrollo del laboratorio, la cual será Scrum, un marco ágil de trabajo que se basa en la organización del proyecto en ciclos cortos llamados sprints. Cada sprint permite planificar, desarrollar, revisar y ajustar el avance del proyecto de forma iterativa e incremental, facilitando la adaptación continua a cambios y promoviendo una mejora constante del proyecto.

CAPÍTULO 3.

DESARROLLO DEL LABORATORIO REMOTO PARA LA TELE-OPERACIÓN DE KITS LEGO MINDSTORM EV3

3. DESARROLLO DEL LABORATORIO REMOTO PARA LA TELE-OPERACIÓN DE KITS LEGO MINDSTORM EV3

3.1 INTRODUCCIÓN

Esta sección se describe el proceso de diseño del laboratorio remoto, desarrollado bajo la metodología ágil Scrum. El diseño contempla la identificación de los requerimientos funcionales y no funcionales, las historias de usuario, la elaboración de diagramas (funcional, relacional y de clases) que representan tanto la estructura lógica del software como la navegación del sistema por parte del usuario, y la modelación del diagrama de clases de la implementación.

Adicionalmente, se presenta la estructura general del laboratorio, detallando la organización de los módulos que lo integran, junto con sus interconexiones, mecanismos de comunicación, funcionalidades y posibles limitaciones técnicas.

3.2 DISEÑO LABORATORIO REMOTO

En esta sección se presenta el diseño del laboratorio, en el cual se detallan los requerimientos funcionales y no funcionales, así como los diagramas relacional y de clases. Además, se incluyen las historias de usuario, el diagrama funcional y el diagrama de navegación, elaborados conforme a la metodología de desarrollo ágil Scrum.

3.2.1 REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

Los requerimientos funcionales definen las características operativas que el laboratorio debe cumplir para asegurar su correcto funcionamiento. Estos se detallan en la tabla 3.1 y se dividen en dos categorías: los requerimientos del frontend, que describen las funciones de la interfaz de usuario, y los requerimientos del backend, que especifican las funcionalidades relacionadas con el robot.

Por otro lado, los requerimientos no funcionales, presentados en la tabla 3.2, describen las características del software, incluyendo las bibliotecas, los entornos de desarrollo y las versiones utilizadas. Además, especifican los requisitos de hardware necesarios para garantizar la compatibilidad con la aplicación.

Tabla 3.1. Requerimientos funcionales.

	Universidad del Valle **LABORATORIO REMOTO PARA LA TELE-OPERACIÓN DE KITS LEGO MINDSTORM EV3**	**Rev.:** **000**
Título: **ESPECIFICACIÓN DE REQUERIMIENTOS FUNCIONALES**	**Documento:** **ERF-001**	Página: **1 de 1**

REVISIÓN HISTÓRICA			
Rev.	**Descripción del Cambio**	**Autor**	**Fecha**
001	Construcción del documento	Christian Mercado	19/5/2024
002	Correcciones	Christian Mercado	2/6/2024
003	Revisión	Christian Mercado	30/6/2024

Ref #	**Funciones**	**Categoría**
1.0	La herramienta de gestión debe proporcionar un login al ingreso al aplicativo. Este debe tener los campos de nombre de usuario y contraseña.	**Front-end**
2.0	La herramienta de gestión debe ofrecer una aplicación para realizar pruebas en sensores, permitiendo seleccionar, probar y escoger entre diferentes modos de uso. Además, debe incluir características que proporcionen ayuda visual para demostrar los cambios en la variable del sensor en tiempo real.	**Front-end**
3.0	La herramienta de gestión debe incluir una sección de ayuda con información para nuevos usuarios y sobre los modos de uso.	**Front-end**
4.0	La herramienta de gestión debe permitir reservar acceso al laboratorio remoto. El usuario podrá reservar una sesión atada a un horario específico en el calendario.	**Front-end**
5.0	El robot móvil debe ser capaz de ser tele-operado mediante el teclado a través de la herramienta de gestión.	**Front-end**
6.0	Los usuarios deben ser almacenados en una base de datos. De igual forma, se debe verificar si un usuario ya está registrado antes de crear y dar acceso a la herramienta de gestión.	**Back-end**
7.0	La herramienta de gestión tener una sección de ejercicios propuestos. La sección de ejercicios propuestos debe contener actividades para realizar con el objetivo de poner en práctica conceptos básicos de robótica móvil.	**Front-end**
8.0	Cuando el tiempo de sesión termina, el usuario debe ser desconectado para dar paso al siguiente usuario.	**Front-end**
9.0	Se debe realizar una abstracción para la comunicación entre ROS-EV3 y el front-end usando rosbridge	**Back-end**
10.0	Después de que un usuario haya iniciado sesión en la herramienta de gestión, esta debe ofrecer un menú con diversas opciones de aplicativos disponibles.	**Front-end**
11.0	La herramienta de gestión debe permitir guardar datos de los sensores durante las sesiones de uso.	**Front-end**
12.0	La herramienta de gestión debe proporcionar una vista de la cámara del laboratorio como método de realimentación para el usuario.	**Front-end**

Tabla 3.2. Requerimientos no funcionales.

	Universidad del Valle **LABORATORIO REMOTO PARA LA TELE-OPERACIÓN DE KITS LEGO MINDSTORM EV3**	**Rev.:** **000**
Título: **ESPECIFICACIÓN DE REQUERIMIENTOS NO FUNCIONALES**	**Documento :** **ERF-001**	Página : **1 de 1**

REVISIÓN HISTÓRICA			
Rev.	**Descripción del Cambio**	**Autor**	**Fecha**
001	Construcción del documento	Christian Mercado	19/5/2024
002	Correcciones	Christian Mercado	2/6/2024
003	Revisión	Christian Mercado	30/6/2024

Ref #	**Descripción**	**Categoría**
1.0	El desarrollo de la herramienta de gestión se llevará a cabo utilizando JavaScript como lenguaje de programación principal, por lo que se empleará como entorno de ejecución Node.js v20.12.0	**Front-end**
2.0	El framework Express v4.19.2 se empleará como controlador en el patrón de diseño Modelo-Vista-Controlador (MVC) para la creación de la interfaz de usuario.	**Front-end**
3.0	Se usará el paquete rosbridge_suite v0.11.17 para establecer comunicación vía websocket entre la aplicación Back-end y Roscore v1.4.1	**Back-end**
4.0	Se usará PostgreSQL v12.18 como sistema de base de datos.	**Front-end**
5.0	Se usará Socket.io v4.7.5 para establecer comunicación vía websocket entre Express y la interfaz.	**Front-end**
6.0	Se usará el Lego MINDSTORM EV3 como plataforma robótica.	**Hardware**
7.0	Se usará una cámara que observe globalmente la escena del laboratorio para el seguimiento del robot.	**Hardware**
8.0	Se utilizará un Raspberry Pi Pico como dongle Wi-Fi	**Hardware**
9.0	Se usará Kinetic como distro de ROS.	**Back-end**

3.2.2 DIAGRAMA FUNCIONAL Y DE NAVEGACIÓN

La navegación del laboratorio se realiza principalmente mediante el uso del mouse, a través del cual el usuario interactúa con la interfaz web, accediendo a los distintos menús y seleccionando las opciones disponibles mediante clics. Por otro lado, el teclado también es utilizado, aunque no como medio de navegación, sino como herramienta de control durante la teleoperación del robot, cuando esta funcionalidad está habilitada.

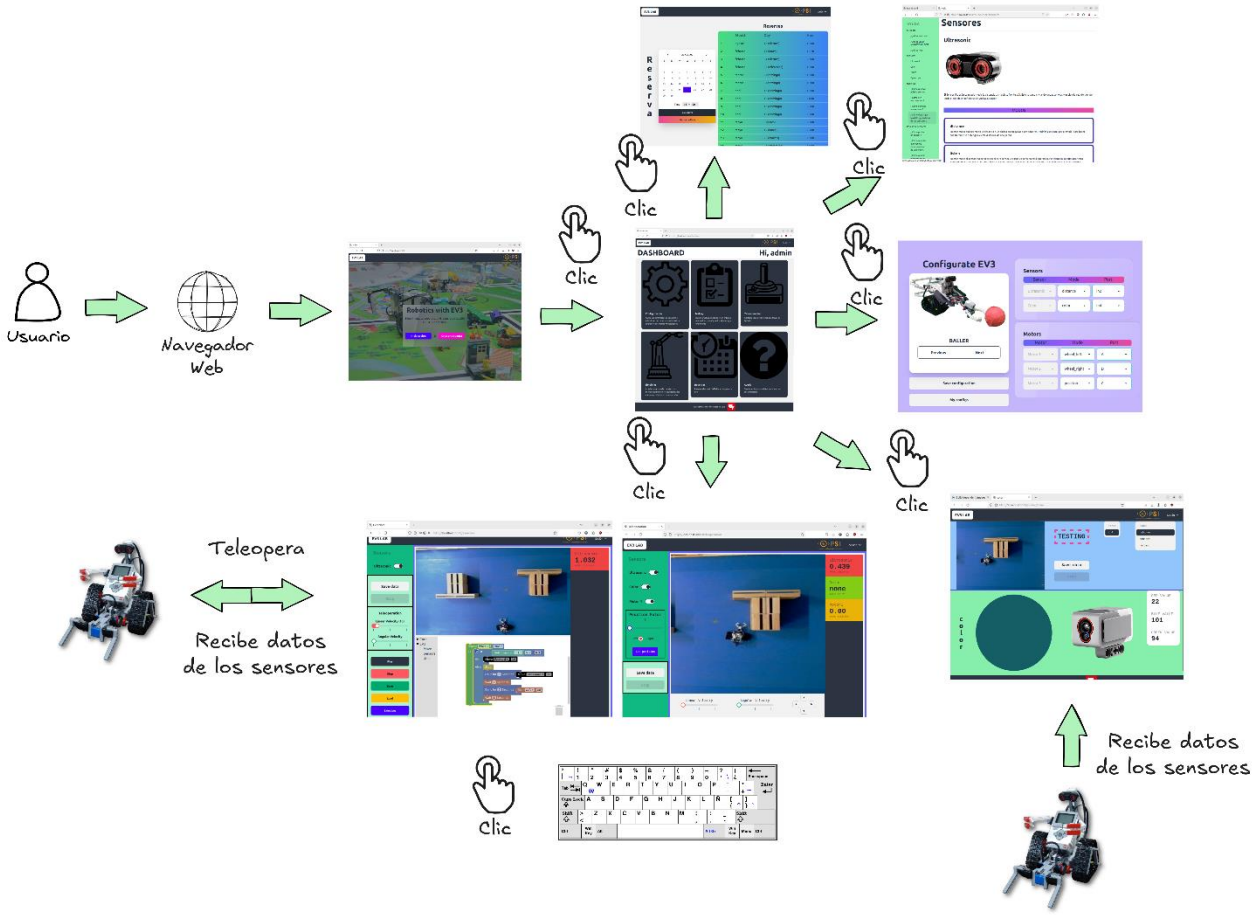


Figura 3.1. Diagrama de navegación.

En la Figura 3.1 se muestra el diagrama de navegación, el cual representa la ruta que sigue el usuario desde su ingreso al laboratorio hasta su recorrido por las distintas secciones. El acceso inicial se realiza a través del navegador web, y la navegación dentro del entorno se lleva a cabo principalmente mediante clics. Para funciones específicas, como la teleoperación del robot o el ingreso de texto, se hace uso del teclado.

3.2.3 DIAGRAMA RELACIONAL

Figura 3.2. Digrama relacional de base de datos.

El diagrama relacional de la base de datos (Figura 3.2) representa la estructura lógica del sistema de almacenamiento del laboratorio, esto incluyendo las tablas, sus atributos y las relaciones entre ellas. Este diagrama proporciona una visión organizada y detallada de cómo se gestionan y distribuyen los datos.

users: Esta tabla hace referencia a los usuarios registrados en el laboratorio.

- id: Identificador único del usuario.
- username: Vinculado al correo electrónico del usuario.
- password: Contraseña de la cuenta.
- name y last_name: Nombres y apellidos del usuario.
- config_id: Referencia a la configuración actual asociada al usuario.

reserves: Esta tabla almacena las reservas del laboratorio realizadas por los usuarios.

- id: Identificador único de la reserva.
- user_id: Relaciona la reserva con el usuario correspondiente.
- date: Fecha programada para la reserva.

- time: Hora asignada para el uso del laboratorio.

config: Esta tabla almacena las configuraciones personalizadas creadas por los usuarios.

- id: Identificador único de la configuración.
- user_id: Vincula la configuración con su respectivo usuario.
- robot_config: Tipo de configuración asociada al robot.
- sensorX_name, sensorX_mode, sensorX_port: Tipo, modo de operación y puerto del sensor configurado (hasta 4 sensores).
- motorX_mode, motorX_port: Modo de funcionamiento y puerto de conexión del motor (hasta 4 motores).
- config_name: Nombre asignado por el usuario a la configuración.

sensor_data: Esta tabla almacena la información registrada por los sensores durante el uso del laboratorio.

- id: Identificador único del conjunto de datos guardados.
- reserve_id: Identificador de la reserva con la que están vinculados los datos.
- data_name: Nombre asignado por el usuario al conjunto de datos almacenados.
- sensorX_data: Campos que contienen los datos registrados por los sensores (hasta 4 sensores).
- timestamp: Momento exacto en el que se registraron los datos.
- config_id: Identificador de la configuración utilizada por el usuario al momento de la captura de datos.

exercise_data: Esta tabla hace referencia a los ejercicios personalizados guardados por los usuarios durante el uso del laboratorio.

- id: Identificador único del ejercicio.
- user_id: Identificador del usuario al que está vinculado el ejercicio.
- exercise_name: Nombre asignado por el usuario al ejercicio guardado.
- exercise_json: Datos del ejercicio almacenados en formato JSON.

3.2.4 DIAGRAMA DE CLASES

En la Figura 3.3 se representa la estructura del objeto principal del laboratorio, correspondiente al robot utilizado. En dicho diagrama se muestran las clases, métodos, atributos y sus relaciones. La clase principal, denominada *EV3*, es una abstracción del robot físico empleado en el laboratorio y está compuesta por otras clases, reflejando la forma en que el robot se conecta físicamente a sus sensores y motores. Para representar estos dispositivos, se incluye una clase específica para los sensores y motores. Además, se incorpora una clase adicional encargada de gestionar parte de la teleoperación del robot.

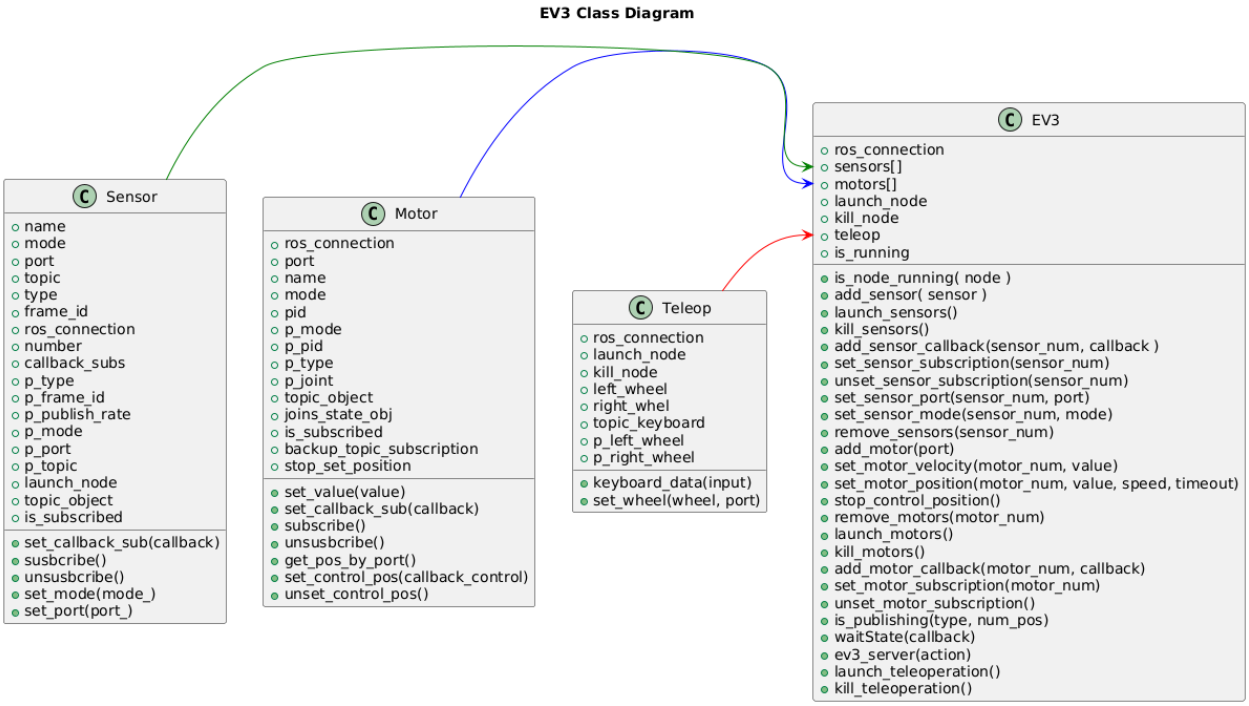


Figura 3.3. Diagrama de clase de objeto EV3.

Como se mencionó anteriormente, la clase EV3 está compuesta por tres subclases: Sensor, Motor y Teleop. Las clases Sensor y Motor representan los componentes físicos del robot, como su nombre lo indica, mientras que la clase Teleop se encarga de la gestión y configuración de la teleoperación del robot.

Clase Sensor

Atributos:

- name: Nombre del sensor (Color, Ultrasonic, Gyroscope, Touch).
- mode: Modo de operación del sensor.
- port: Puerto al que está conectado el sensor.

- topic: Parte del nombre del tópico donde se publica la información del sensor.
- type: Tipo de sensor.
- frame_id: Identificador del marco de referencia del sensor.
- ros_connection: Objeto de conexión con ROS vía rosbridge.
- number: Número identificador del sensor de ese tipo.
- callback_subs: Función que se ejecuta cuando se recibe información del sensor.
- p_type: Parámetro del tipo de sensor.
- p_frame_id: Parámetro del frame_id.
- p_publish_rate: Frecuencia de publicación de información del sensor. (No modificable).
- p_mode: Modo de operación actual del sensor.
- p_port: Puerto actual de conexión del sensor.
- p_topic: Nombre del tópico donde se publicará la información.
- launch_node: Objeto que representa el nodo encargado de lanzar otros nodos.
- topic_object: Objeto del tópico donde se publica la información del sensor.
- is_subscribed: Indica si el sensor está suscrito al tópico correspondiente.

Métodos:

- set_callback_subs(callback): Asigna la función que se ejecutará al recibir datos del sensor.
- subscribe(): Activa la suscripción al tópico del sensor.
- unsubscribe(): Desactiva la suscripción al tópico del sensor.
- set_mode(mode_): Cambia el modo actual del sensor.
- set_port(port_): Cambia el puerto al que está conectado el sensor.

Clase Motor

Atributos:

- ros_connection: Objeto de conexión con ROS vía rosbridge.
- port: Puerto al que está conectado el motor.
- name: Nombre del motor.
- pid: Valores del controlador PID utilizados por el nodo de control.
- p_mode: Modo de operación actual del motor.

- p_type: Tipo de motor.
- p_joint: Parámetro del joint correspondiente al motor.
- topic_object: Objeto del tópico para establecer la velocidad del motor.
- joint_state_object: Objeto del tópico de retroalimentación del motor, donde se publica su velocidad y posición.
- is_subscribed: Indica si el motor está suscrito al tópico de retroalimentación.
- backup_topic_subscription: Función que se ejecuta al recibir datos del tópico de retroalimentación.
- stop_set_position: Variable para detener el control de posición del motor.

Métodos:

- set_value(value): Establece la velocidad del motor.
- set_callback_subs(callback): Asigna la función que se ejecutará al recibir datos del motor.
- subscribe(): Activa la suscripción al tópico de retroalimentación.
- unsubscribe(): Desactiva la suscripción al tópico de retroalimentación.
- get_pos_by_port(): Devuelve la posición correspondiente al motor según el tópico de retroalimentación.
- set_control_pos(callback_control): Activa el control de posición del motor.
- unset_control_pos(): Desactiva el control de posición del motor.

Clase Teleop

Atributos:

- ros_connection: Objeto de conexión con ROS vía rosbridge.
- launch_node: Objeto del nodo responsable de lanzar otros nodos.
- kill_node: Objeto del nodo encargado de detener otros nodos.
- left_wheel: Motor asignado como rueda izquierda.
- right_wheel: Motor asignado como rueda derecha.
- topic_keyboard: Objeto del tópico que recibe el control por teclado.
- p_left_wheel: Parámetro para configurar la rueda izquierda en el nodo de control diferencial.
- p_right_wheel: Parámetro para configurar la rueda derecha en el nodo de control diferencial.

Métodos:

- keyboard_data(input): Procesa las entradas del teclado para teleoperar el robot.
- set_wheel(wheel, port): Configura la rueda y el puerto correspondiente para la teleoperación.

3.2.5 ESTRUCTURA DEL LABORATORIO REMOTO

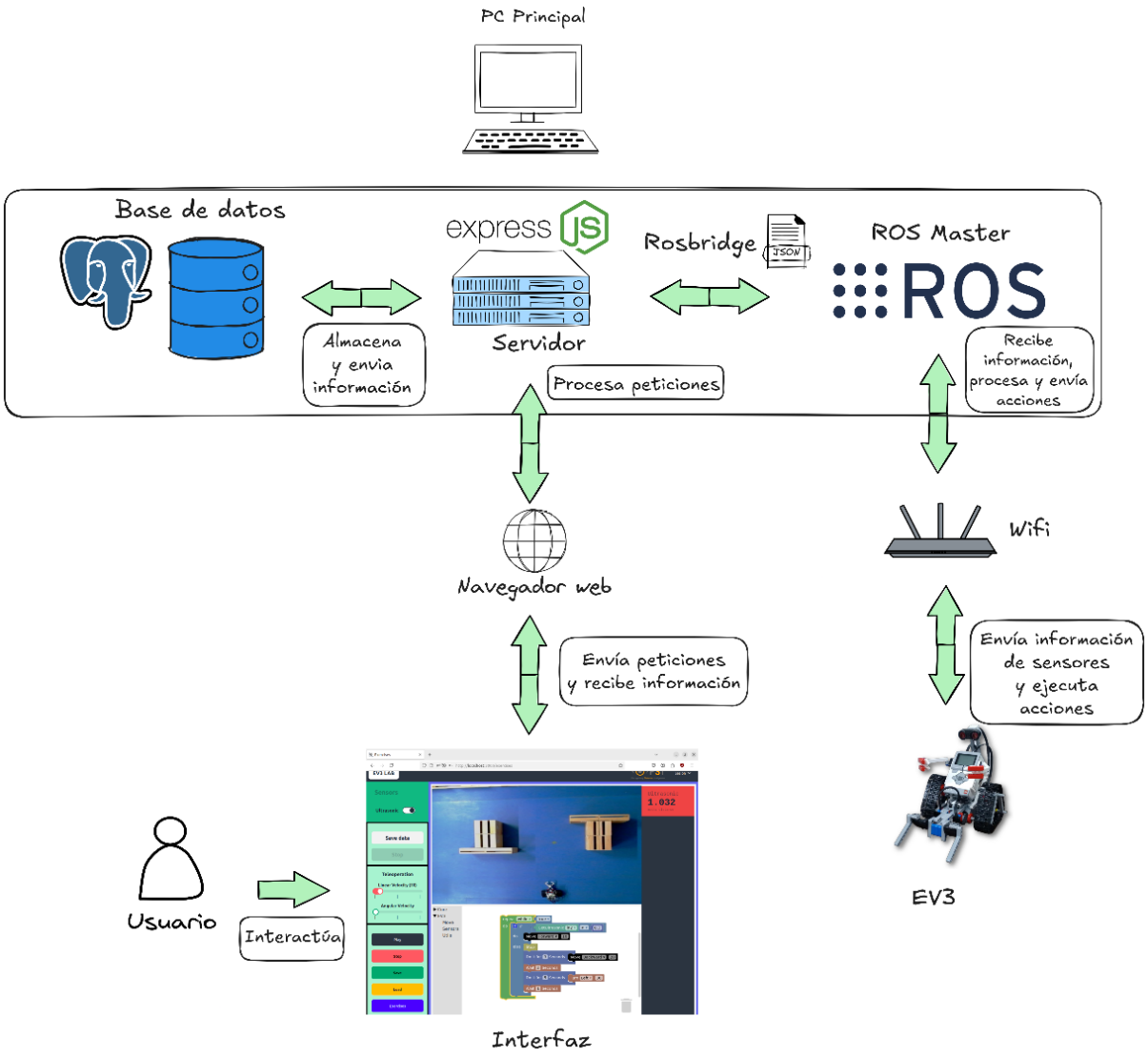


Figura 3.4. Estructura del laboratorio.

En la figura 3.4 se muestra la estructura general del laboratorio, este se basa en un PC principal que ejecuta el servidor encargado de desplegar la página web del laboratorio. Este servidor está desarrollado principalmente en Node.js, utilizando el framework Express. En el mismo equipo también se ejecutan la base de datos y el sistema operativo ROS.

El usuario accede al laboratorio a través de un navegador web, siempre y cuando esté conectado a la misma red local si se planea usarlo de forma local. La comunicación con el robot se realiza mediante ROS, donde el ROS Master se ejecuta en el PC principal y el robot EV3 se conecta a este a través de Wi-Fi.

Cuando el usuario interactúa con la interfaz web, las solicitudes se envían al servidor, el cual las procesa. Si es necesario ejecutar alguna acción en el robot, el servidor se comunica con ROS utilizando rosbridge para enviar los comandos correspondientes.

3.3 IMPLEMENTACIÓN DEL LABORATORIO REMOTO

En esta sección se explica en detalle el funcionamiento del laboratorio, incluyendo la comunicación entre sus distintos componentes, tanto en el front-end como en el back-end, así como la interacción con el robot. Se describen las diversas secciones que conforman el laboratorio y las acciones que el usuario puede realizar dentro del laboratorio.

3.3.1 COMUNICACIÓN ENTRE EL EV3 Y EL COMPUTADOR

El robot EV3 utiliza una imagen de Linux personalizada que incluye la instalación de ROS y los paquetes necesarios para su funcionamiento y comunicación. Esta imagen, basada en Yocto Linux, fue desarrollada por su autor original, quien se encargó de adaptarla, crear scripts de configuración para la comunicación con ROS y desarrollar los nodos encargados de gestionar la comunicación, así como el control de los sensores y actuadores del robot [38].

Para establecer la conexión entre el computador y el EV3, se emplea un dongle Wi-Fi. En el EV3, es necesario la asignación de la IP correspondiente a sí mismo y la del ROS MASTER, así como de interfaz de red relacionada el dongle wifi usado (Figura 3.4). Esto se hace a través de unos scripts ya definidos a través de consola del EV3.

- **`ros_master_set` IP_ROS_MASTER**
- **`ros_ip_set` INTERFAZ_DE_RED**

El computador funciona como ROS MASTER que es donde se ejecuta roscore, este es un conjunto de procesos esenciales que deben estar en ejecución para que funcione un sistema ROS. El ROS MASTER gestiona la información de registro de tópicos, servicios y nodos. Los nodos se conectan directamente entre sí, y el ROS MASTER solo proporciona información de ubicación, similar a un servidor DNS. Los nodos que se suscriben a un tema solicitan conexiones a los nodos que lo publican, estableciendo la conexión a través del protocolo TCPROS, que utiliza sockets estándar.

En el computador es necesario que esté declarado el hostname ev3dev con la dirección IP del robot.

En la imagen de Linux del EV3, existe un alias o shortcut llamado `ev3_manager` que ejecuta el binario del nodo `Ev3ControlNode`, este está basado en el paquete `controller_manager`, que es paquete de ROS que proporciona una infraestructura de control con realimentación para

la administración de controladores. El `Ev3ControlNode` se responsabiliza de inicializar, lanzar y detener los controladores relacionados con los sensores y motores.

Para empezar a cargar algún controlador de un sensor o motor es necesario entonces lanzar un nodo tipo `spawner` del paquete `controller_manager` desde el computador, especificando el nombre del sensor y apuntando hacia la dirección IP asignada al robot.

Los launch_files necesarios para estas operaciones se encuentra en el paquete `ev3_sensors` del workspace `catkin_ws` del computador. Los launch_files son los encargados de lanzar todos los nodos.

La estructura de la carpeta del paquete es así:

```
└── ev3_sensors/
    ├── CMakeLists.txt
    ├── package.xml
    └── launch/
        ├── camera.launch
        ├── color.launch
        ├── color.yaml
        ├── gamepad.launch
        ├── generic.launch
        ├── gyro.launch
        ├── gyro.yaml
        ├── motors.launch
        ├── motors.yaml
        ├── teleop.launch
        ├── teleop.yaml
        ├── touch.launch
        ├── touch.yaml
        ├── ultrasonic.launch
        └── ultrasonic.yaml
```

Los *launch files* con nombres de sensores son ejemplos utilizados para lanzar los respectivos controladores. Por otro lado, los archivos YAML definen ciertos parámetros fijos, como el tipo de controlador que se debe cargar. Cuando se lanza un nodo de un sensor dentro del laboratorio, se utiliza el archivo launch llamado **generic**. Para los motores, se emplea el archivo **motors**, y para la teleoperación, el archivo **teleop**.

Un ejemplo de un launch file para cargar el controlador del sensor de ultrasonido es el siguiente:

```
<launch>
      <group ns="$(arg ev3_hostname)">
            <!-- Load joint controller configurations from YAML file to parameter server
-->
            <rosparam   file="$(find   h4r_ev3_control_launch)/config/ultrasonic.yaml"
command="load"/>
            <!-- load the controllers -->
            <node  name="ev3_sensor_spawner"  pkg="controller_manager"  type="spawner"
respawn="false"
       output="screen" args="Ev3UltraSonic"/>
```

```
        </group>
</launch>
```

En este caso se observa que el nombre que se le asigna al nodo es **`ev3_sensor_spawner`**, que es de tipo **`spawner`** y proviene del paquete **`controller_manager`**. Se da como argumento (**`args`**) el nombre del tipo de controlador, en este caso es para el sensor ultrasónico, así que se pone **`Ev3UltraSonic`** y como namespace (**`ns`**), se pasa la dirección IP del EV3 para el argumento con nombre **`ev3_hostname`** al ejecutar el launch.

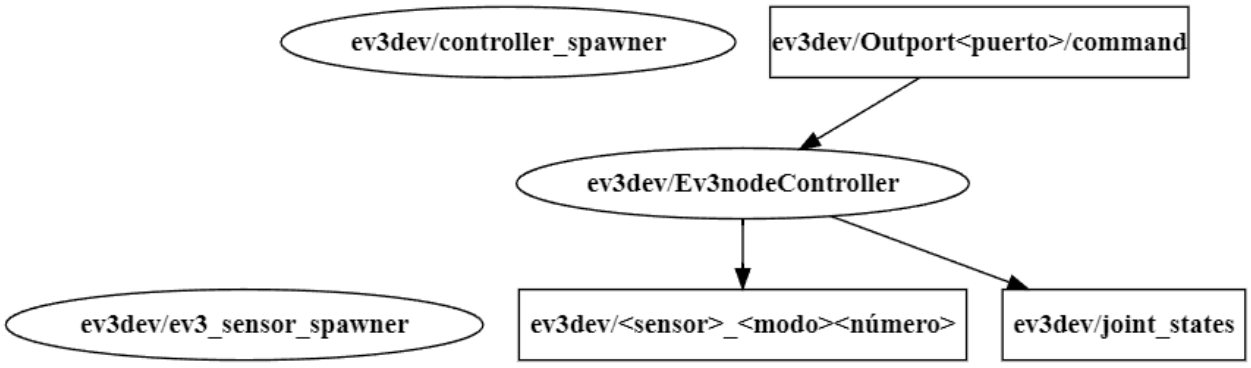


Figura 3.5. Nodos generados para sensores y motores.

En la Figura 3.5 se pueden observar los nodos que se generan al ejecutar un archivo launch. En la imagen se muestran dos nodos principales: ev3_sensor_spawner, encargado de los sensores, y controller_spawner, utilizado para los motores. Ambos nodos son de tipo *spawner* y pertenecen al paquete controller_manager; su función es cargar los controladores correspondientes para sensores y actuadores. El nodo principal es EV3nodeController, el cual actúa como punto central en donde se cargan y ejecutan dichos controladores. Cuando se carga un controlador para un sensor, el nombre del tópico en el que se publica la información sigue la estructura: <sensor>_<modo>_<número>. Por otro lado, la información conjunta de todos los motores se publica en el tópico joint_states. Finalmente, el tópico OutPort<puerto> es utilizado para aplicar el valor de velocidad correspondiente a cada motor de acuerdo con el puerto dónde está conectado.

3.3.2 COMUNICACIÓN ENTRE EL SERVIDOR Y LA GUI

El servidor web del laboratorio está construido con el framework `Express` para Node.js, que se encarga de procesar las solicitudes y responder con el contenido solicitado. Para comunicarse directamente con ROS, se utiliza roslibjs, una biblioteca que permite interactuar con ROS a través de la web. Junto con el paquete `rosbridge`, `roslibjs` establece una conexión mediante WebSockets (Figura 3.4).

`Rosbridge` proporciona una interfaz JSON para ROS, lo que permite a cualquier cliente enviar JSON para publicar o suscribirse a temas de ROS, realizar llamadas a servicios de ROS, entre otras funcionalidades; es con este nodo con el que `roslibjs` hace comunicación directa, cuando se establece comunicación con ROS desde la web (Figura 3.6).

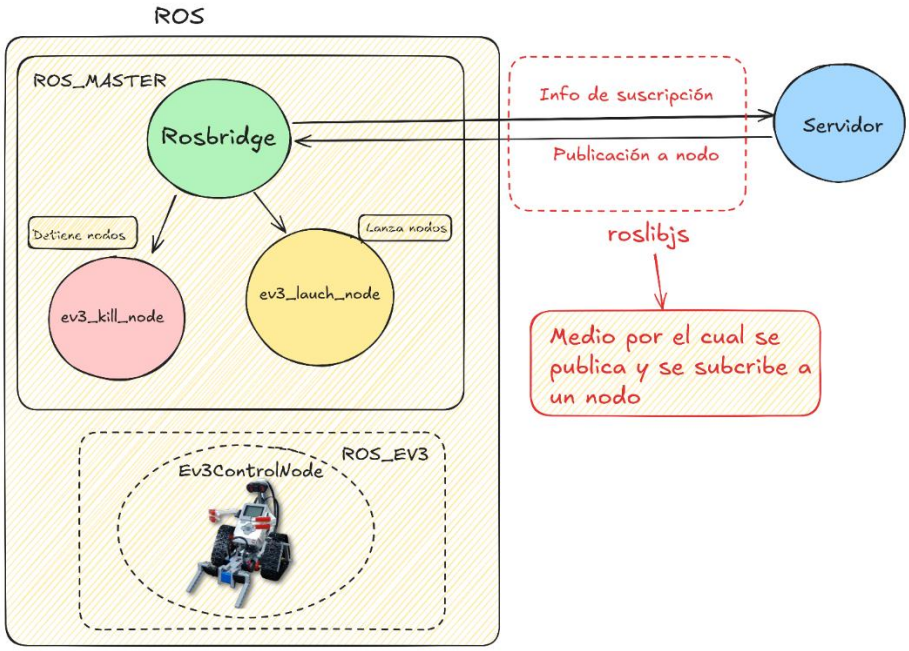


Figura 3.6. Proceso de comunicación entre el servidor de Express con ROS por medio de rosbridge.

Además de los `launch_files` diseñados para el lanzamiento de sensores y motores, se desarrolló un paquete adicional llamado `ev3_pc` que incluye dos nodos, `ev3_launch_node` y `ev3_kill_node` (Figura 3.6). El primero se encarga de gestionar el lanzamiento de los `launch_files`, mientras que el segundo está destinado a detener nodos específicos. Ambos nodos funcionan de manera similar: reciben una cadena de texto como argumento, la cual indica qué launch file debe ser ejecutado o qué nodo debe ser detenido. Una vez procesada esta información, se genera un nuevo proceso independiente del nodo original y se lleva a cabo la acción correspondiente.

La comunicación entre estos nodos y el sistema se realiza a través de `rosbridge`, que facilita la conexión desde el servidor web. Dentro de la estructura del servidor web se encuentra una clase denominada `EV3`, que actúa como un objeto que agrupa toda la información relevante del robot actual, incluyendo la conexión con ROS y los métodos necesarios para ejecutar diversas acciones.

La interfaz de usuario accede a esta clase una vez establecida la comunicación a través de WebSocket, utilizando `Socket.IO`, una biblioteca que permite la comunicación bidireccional, de baja latencia y basada en eventos entre el cliente y el servidor. Tras la autenticación del usuario, se establece la conexión, lo que le permite ejecutar acciones remotas sobre el robot. A través de esta conexión con `Socket.IO` también se recibe toda la información proveniente de los sensores. Dado que Socket.IO es flexible respecto al tipo de datos que puede manejar, los datos de los sensores suelen enviarse en el formato original con el que el nodo los publica en su tópico correspondiente. El tipo de mensaje varía según el sensor: puede tratarse de un valor booleano,

un entero, un número flotante, o combinaciones de estos junto con una marca de tiempo (*timestamp*).

La estructura de los mensajes para los sensores para cada modo son los siguientes:

touch:

- normal: mensaje tipo **std_msgs/Bool**

```
bool data
```

ultrasonic:

- distance: mensaje tipo **sensor_msgs/Range**

```
std_msgs/Header header
uint8 radiation_type
float32 field_of_view
float32 min_range
float32 max_range
float32 range
```

- listen: mensaje tipo **std_msgs/Bool**

```
bool data
```

color:

- rgb_raw: mensaje tipo **std_msgs/ColorRGBA**

```
float32 r
float32 g
float32 b
float32 a
```

- ambient, reflect: mensaje tipo **sensor_msgs/Illuminance**

```
std_msgs/Header header
float64 illuminance
float64 variance
```

- color: mensaje tipo **std_msgs/UInt8**

```
uint8 data
```

gyroscope:

- angle, rate: mensaje tipo **sensor_msgs/Imu**

```
std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
```

```
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
```

Para las demás acciones basadas en acceso de a la base de datos o autenticación, se manejan directamente con peticiones GET y POST a través del servidor web (Figura 3.7). En general, las solicitudes al servidor se gestionan mediante un objeto JSON que contiene un valor numérico utilizado para identificar el tipo de petición. Si la solicitud requiere información adicional, estos datos se extraen del resto del mensaje contenido en el mismo objeto JSON. En cuanto a las respuestas, se manejan de forma similar: se utiliza un objeto JSON cuando la solicitud requiere datos específicos como respuesta; de lo contrario, se retorna únicamente un código de estado HTTP para indicar el resultado de la operación.

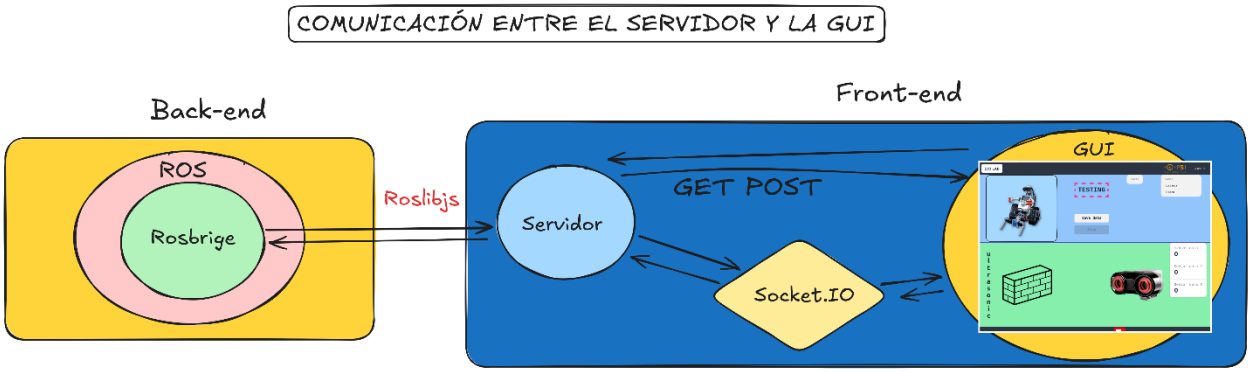


Figura 3.7. Comunicación entre ROS, el servidor y la interfaz (GUI).

3.3.3 DASHBOARD DE INICIO

Figura 3.8. Dashboard del laboratorio para acceder a las diferentes secciones.

Cuando un usuario se ha autenticado correctamente con sus credenciales de inicio a través de la página principal, el servidor web lo dirigirá a la sección del Dashboard (Figura 3.8), donde se encuentran las demás secciones del laboratorio:

- **Configuración:** En esta sección el usuario puede escoger o armar una configuración del robot.
- **Testing:** En esta sección, el usuario puede probar los sensores o motores que están seleccionados en la configuración actual.
- **Teleoperación**: En esta sección, el usuario puede controlar el robot de forma remota y visualizar la información de los sensores configurados en la selección actual.
- **Ejercicios:** En esta sección, el usuario puede practicar las guías de ejercicios dispuestas para el laboratorio, o si lo requiere, solo crear programas basados en bloques de código para la configuración actual.
- **Reservar**: En esta sección, además de mostrar las reservas del día, el usuario puede reservar el uso del laboratorio por periodos de una hora, eliminar una reserva o acceder a los datos guardados durante una reserva anterior.
- **Ayuda:** En esta sección, el usuario puede consultar información sobre el funcionamiento del laboratorio y el robot.

3.3.4 MÓDULO DE CONFIGURACIÓN

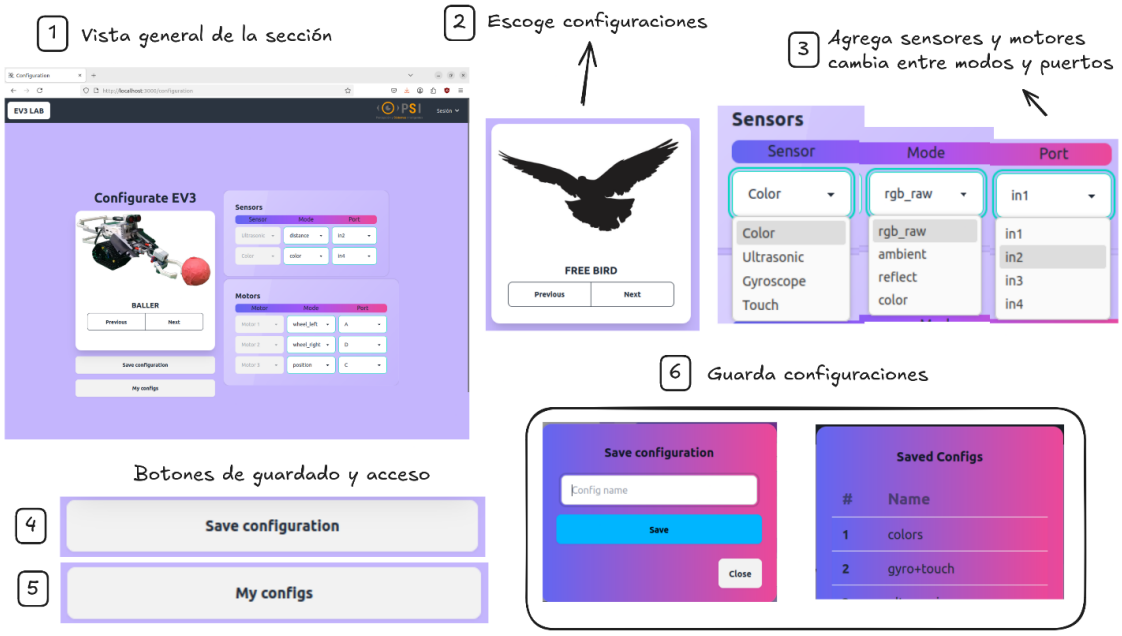


Figura 3.9. Sección de configuración.

La figura 3.9 muestra la sección de configuración. En el primer apartado se presenta una vista general de toda la sección. En el segundo, se muestra un ejemplo de una configuración predefinida. El tercer apartado corresponde al área destinada a la configuración de sensores y motores. En la cuarta y quinto apartado se muestra el botón de acción de guardado y acceso a las configuraciones. Finalmente, en el sexto apartado se presenta una vista de las configuraciones guardadas.

En esta sección, al ingresar, el usuario encontrará inicialmente una serie de configuraciones predefinidas, junto con los sensores y motores utilizados en la configuración actual, así como botones para guardar o seleccionar configuraciones previamente almacenadas. Algunas de estas configuraciones corresponden a las construcciones utilizadas en las guías de laboratorio.

Cuando el usuario selecciona una de las opciones predefinidas, podrá modificar los puertos y elegir entre diferentes modos para los sensores o motores.

Figura 3.10. Sensores y motores del EV3.

Entre los sensores disponibles se encuentran el sensor de toque (`touch`), el sensor de ultrasonido (`ultrasonic`), el sensor de color (`color`) y el sensor giroscópico (`gyroscope`) (Figura 3.10).

El sensor de toque (`touch`) posee un único modo de funcionamiento:

- **normal**: detecta si el sensor está siendo presionado o no.

El sensor de ultrasonido (`ultrasonic`) cuenta con dos modos de operación:

- **distance**: mide la distancia entre el sensor y un objeto.
- **listen**: detecta señales ultrasónicas externas.

El sensor de color (`color`) dispone de los siguientes modos:

- **rgb_raw**: lee los valores de color crudos en formato RGB (rojo, verde, azul).
- **ambient**: mide la intensidad de la luz ambiental.
- **reflect**: mide la intensidad de la luz reflejada.
- **color**: detecta hasta ocho colores predefinidos.

El sensor giroscópico (`gyroscope`) tiene dos modos de funcionamiento:

- **angle**: mide el ángulo de rotación en grados.
- **rate**: mide la velocidad de rotación en grados por segundo.

Para los motores se encuentran disponibles dos modos de funcionamiento: `wheel` y `position`.

- **wheel**: se utiliza cuando el motor actúa como rueda.
- **position**: permite controlar el motor en función de un ángulo específico.

Además de las configuraciones predefinidas, existe una opción llamada `Free Bird` (Figura 3.10 apartado 2), que, a diferencia de las anteriores, permite al usuario seleccionar y añadir cualquier sensor o motor, así como definir el modo de funcionamiento y el puerto al que estarán conectados.

Para guardar una configuración, basta con presionar el botón `Save Configuration` (Figura 3.10 aparatado 4) y asignar un nombre. Esta configuración se almacenará en la base de datos y estará vinculada al usuario actual.

Para acceder a una configuración previamente guardada, el usuario debe hacer clic en el botón `My Configs` (Figura 3.10 apartado 5). El aplicativo mostrará todas las configuraciones guardadas por el usuario, permitiendo seleccionar o eliminar alguna de ellas.

3.3.5 MÓDULO DE RESERVAS

La figura 3.11 muestra una vista general de la sección de reservas y sus botones principales. En el apartado 2 y 3 se encuentran los botones `Do Reserve` y `My Reservations`, que permiten realizar una nueva reserva y consultar las reservas existentes, respectivamente. En el apartado 4 se visualizan las reservas realizadas, las cuales aparecen al presionar el botón `My Reservations`. Finalmente, en el apartado 5 se muestra el menú que se despliega al hacer clic sobre una reserva, desde el cual es posible eliminarla o acceder a los datos almacenados relacionados con ella.

Figura 3.11. Sección de reservas.

Para utilizar el robot, el usuario debe realizar una reserva previa del laboratorio. Este proceso se lleva a cabo accediendo a la sección de reservas, disponible desde el dashboard de la interfaz.

Una vez dentro, el usuario encontrará un calendario interactivo, diseñado para seleccionar el día y la hora en que desea realizar la reserva. Al hacer clic sobre un día específico, el sistema muestra las reservas ya existentes para esa fecha (Figura 3.11 apartado 1).

Las reservas tienen una duración fija de una hora, y los espacios disponibles se generan a partir de la hora actual y en intervalos de media hora hacia adelante. Es decir, solo se puede reservar en franjas que estén dentro del rango permitido desde el momento en que se accede al sistema.

Si el usuario intenta reservar un horario que ya ha sido asignado a otra persona, el sistema le notificará que el espacio no está disponible. En caso contrario, si la franja horaria está libre, el laboratorio confirmará que la reserva se ha realizado exitosamente.

Para realizar una reserva, el usuario debe presionar el botón `Do Reserve` (Figura 3.11 apartado 2), lo que envía una petición al servidor para registrar la solicitud. Una vez procesada, la información de la reserva se almacena en la base de datos.

Si el usuario desea consultar las reservas que ha realizado previamente, puede hacerlo presionando el botón `My Reservations` (Figura 3.11 apartado 3). Al hacerlo, se desplegará una lista con todas sus reservas que realizado hasta ese momento. Al seleccionar una de ellas, el usuario podrá eliminar la reserva o consultar los datos asociados a dicha sesión.

Durante el uso del laboratorio, el usuario tiene la posibilidad de guardar datos generados por los sensores. Estos registros también se almacenan en la base de datos y se vinculan directamente a la reserva correspondiente.

Para acceder a esta información, el usuario debe hacer clic en el botón `Saved Sensor Data` (Figura 3.11 apartado 4), el cual despliega todos los datos registrados durante la sesión de laboratorio. El sistema realiza una petición al servidor, que consulta la base de datos para obtener toda la información relacionada con la reserva seleccionada. Una vez obtenida la información se presenta en la interfaz web, utilizando el nombre asignado por el usuario.

Los datos pueden ser descargados en formato .CSV, o si se desea, también pueden ser eliminados desde la misma interfaz.

3.3.6 MÓDULO DE PRUEBAS

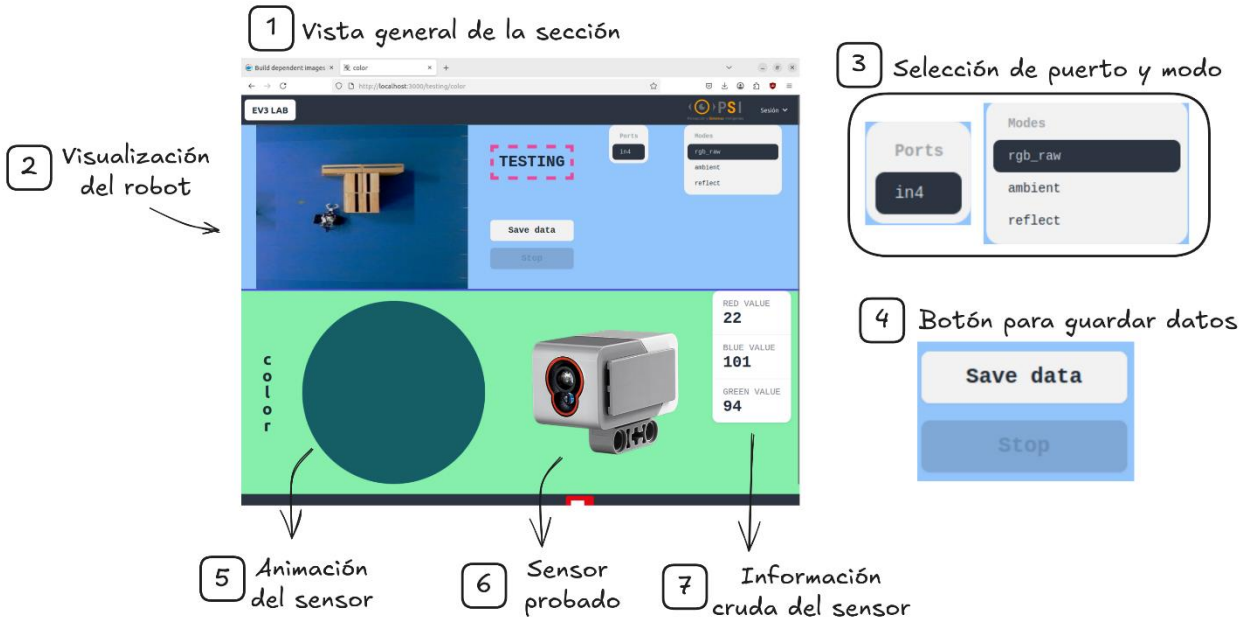


Figura 3.12. Sección de pruebas

En esta sección, al hacer clic sobre la opción correspondiente en el dashboard, el usuario accede al módulo de sensores disponibles para prueba. Las opciones que se muestran están basadas en la configuración previamente guardada, la cual es leída mediante una petición del laboratorio a la base de datos.

La figura 3.12, en el apartado 1, se muestra una vista general de la sección de pruebas. En el apartado 2 se presenta la visualización del robot a través de la cámara. El apartado 3 contiene las opciones para seleccionar los puertos del sensor, así como el modo de funcionamiento. En el apartado 4 se encuentra el botón para la toma de datos del sensor. Finalmente, los apartados 5, 6 y 7 muestran, respectivamente, una animación representativa del funcionamiento del sensor, el sensor que está siendo probado y la información correspondiente a dicho sensor.

Una vez dentro del módulo, el sistema procede a lanzar el nodo correspondiente al sensor seleccionado automáticamente. Antes de esto, se verifica que no exista un nodo activo con el mismo nombre; en caso contrario, el nodo existente es detenido para evitar conflictos y se lanza una nueva instancia. Para los sensores el nombre del nodo es `ev3_sensor_spawner`.

En el servidor, se configura una instancia del robot a través de una clase llamada `EV3`, que representa un modelo abstracto del robot físico. Esta instancia se ajusta con base en la configuración del usuario. Para el módulo de pruebas, esto implica agregar el sensor seleccionado a la instancia del robot. Toda la comunicación entre la interfaz web y el servidor se realiza mediante WebSockets, utilizando la biblioteca `Socket.IO`. A través de `Socket.IO` se envían cadenas de texto como parámetros, las cuales hacen referencia a los métodos del objeto **`EV3`** que deben ejecutarse en el servidor. El siguiente es un ejemplo de un mensaje de la interfaz al servidor para agregar un sensor de ultrasonido conexión al puerto 2 y en modo `distance` vía Socket.IO:

```
socket.emit('add_sensor', 'ultrasonic');
socket.emit('set_sensor_port', 'in2');
socket.emit('set_sensor_mode', 'distance');
```

Una vez que los nodos se lanzan correctamente, la interfaz se desbloquea y el usuario puede comenzar a visualizar en pantalla la información generada por el sensor. Entre los parámetros configurables, el usuario puede seleccionar el puerto al que está conectado el sensor, siempre que el tipo de sensor coincida. También puede cambiar el modo de funcionamiento del sensor; en ese caso, el nodo actual es detenido y relanzado con el nuevo modo seleccionado. Visualmente, esta sección presenta animaciones representativas del funcionamiento del sensor.

Para explicar mejor lo anterior se expone el siguiente ejemplo:

Suponiendo que un usuario ha realizado una reserva en un horario disponible. Si este usuario decide utilizar la configuración **Baller** (Figura 3.13) para su práctica, al hacer clic sobre el módulo de pruebas en la dashboard, se desplegarán las opciones de sensores disponibles, de acuerdo con la configuración seleccionada. En este caso, al haber elegido la configuración **Baller**, estarán disponibles los sensores de **ultrasonido**, **color** y un **motor** (Figura 3.13). Esta información es extraída desde la base de datos asociada al usuario.

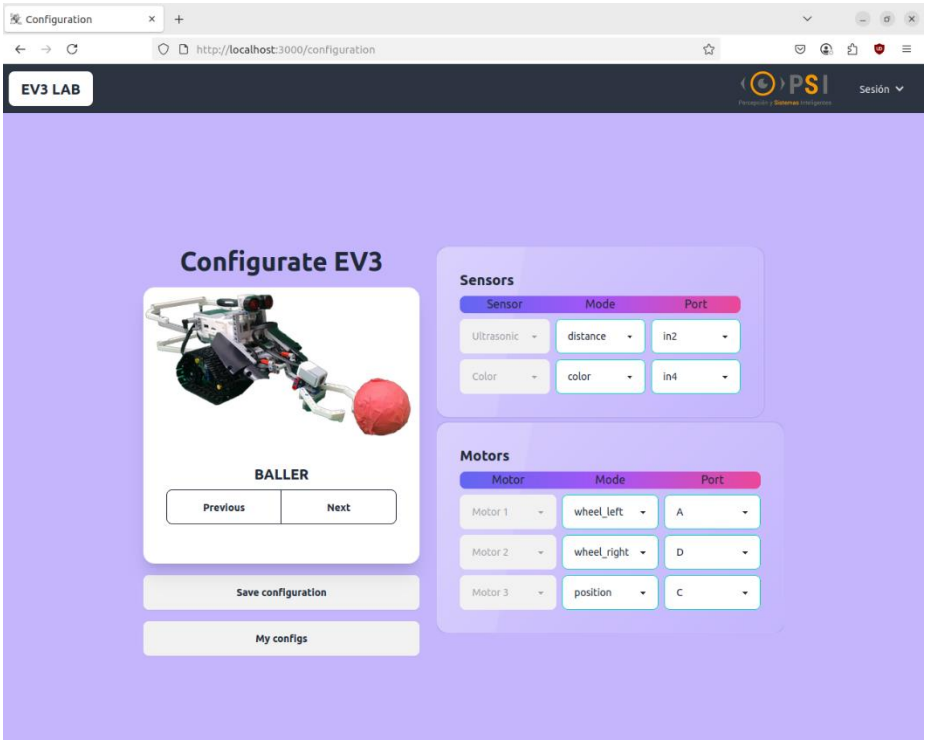


Figura 3.13. Configuración Baller para el EV3.

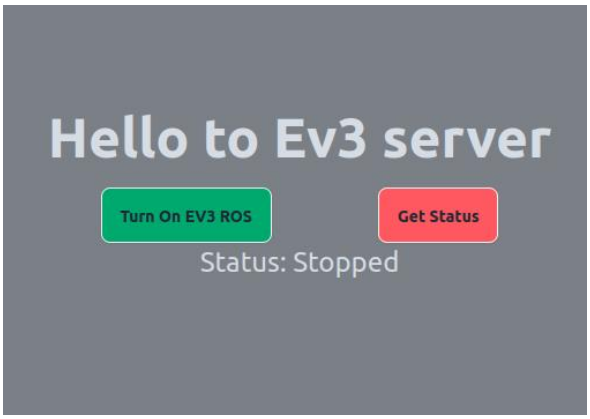


Figura 3.14. Web server del EV3.

Si, por ejemplo, se elige el sensor de ultrasonido, la interfaz gráfica se bloqueará inicialmente para evitar interacciones que interfieran con el lanzamiento de los nodos en el EV3. Primero, el sistema verifica que exista conexión con el robot EV3. Esto lo hace el servidor mediante una petición HTTP al servidor web que corre en el robot (Figura 3.14), cuya dirección es **`http://EV3_IP:8000/`**. Si no hay conexión, el usuario será redirigido al dashboard. En caso contrario, el proceso continuará, pero no sin antes verificar que el nodo `Ev3ControlNode` se encuentre en ejecución. Este nodo es lanzado automáticamente por el web server del robot desde el servidor del laboratorio, en caso de que no esté activo.

Luego, se lee la configuración del usuario para obtener el puerto al que está conectado el sensor de ultrasonido. En la instancia del objeto EV3 —que representa al robot físico— se limpian los sensores previamente asociados y se agrega el sensor de ultrasonido con el puerto correspondiente. Para configurar este puerto, se llama al método **`set_mode(mode_)`** del sensor dentro de la instancia del objeto del robot. Este método establece internamente el parámetro **`/ev3dev/Ev3Ultrasonic0/port`** y es aquí donde establece el puerto de conexión. La estructura de este parámetro sigue el formato: **`/ev3dev/Ev3 + <nombre_del_sensor> + <número_del_sensor> + /port.`**

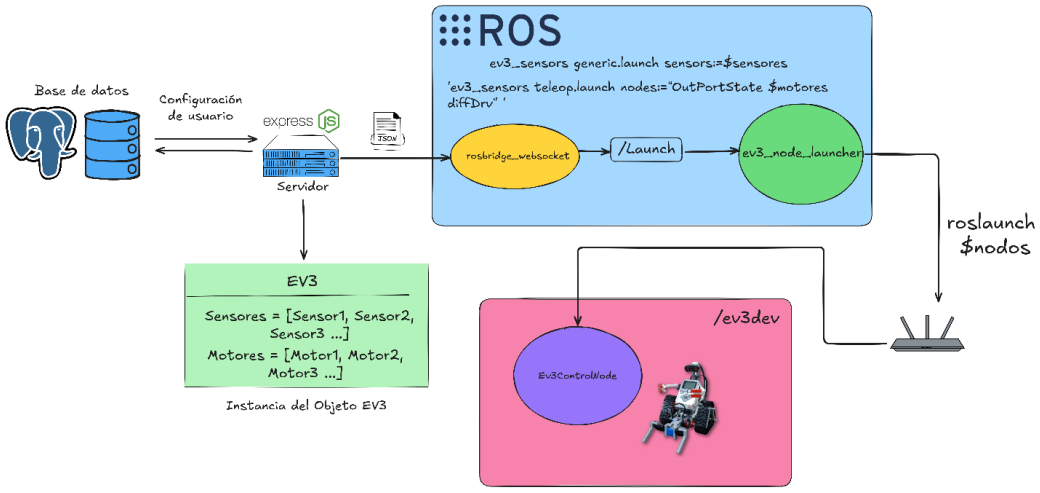


Figura 3.15. Lanzamiento de nodos.

A continuación, se define la función que será llamada cada vez que lleguen datos desde el sensor, y se lanza el nodo correspondiente. Esto lo realiza el servidor llamando al método **`launch_sensors()`** del objeto EV3, el cual publica en el tópico **`/launch`** el comando:

```
ev3_sensors generic.launch sensors:=ultrasonic
```

Este mensaje es recibido por el nodo **`ev3_node_launcher`** (Figura 3.15), que crea un nuevo proceso vinculado a ese nodo. Se espera entonces a que el nodo esté activo, lo que se comprueba monitoreando la disponibilidad del tópico que transmite los datos del sensor. El formato de este tópico es **`/<sensor>_<modo><número del sensor>`**, por ejemplo: **`/ultrasonic_distance0`**, donde:

- **`ultrasonic`** es el nombre del sensor.
- **`distance`** es el modo por defecto.
- `0` indica que es el primer sensor de ese tipo agregado.

Una vez que el tópico está disponible, el objeto EV3 se suscribe a él. Cada vez que el sensor publique datos, esta información es transmitida del servidor a la interfaz mediante un web socket,

donde se procesa y muestra al usuario. En el caso del sensor de ultrasonido, se visualiza una barra animada que varía de acuerdo con la distancia medida (Figura 3.16).

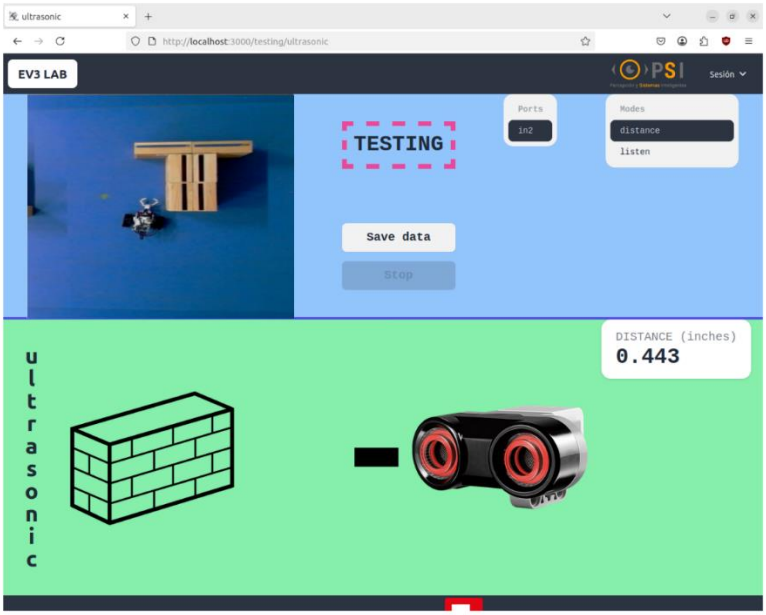


Figura 3.16. Visualización del sensor ultrasónico en la sección de pruebas.

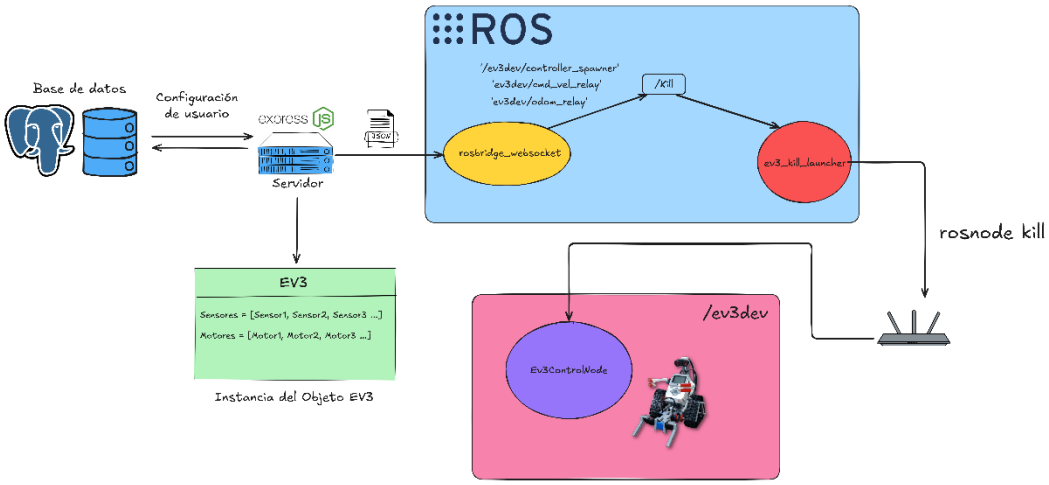


Figura 3.17. Detención de nodos.

Si el usuario desea cambiar el modo de funcionamiento del sensor, puede seleccionar otra opción desde la interfaz. Al hacerlo, se actualiza el modo del sensor dentro del objeto EV3, y el nodo activo es detenido mediante el método **`kill_sensors()`**, que publica en el tópico **`/kill`** el texto:

```
ev3dev/ev3_sensor_spawner
```

Este mensaje es procesado por el nodo **`ev3_kill_node`** (Figura 3.17), el cual ejecuta el comando:

```
rosnode kill ev3dev/ev3_sensor_spawner
```

Esto detiene el nodo que controla el sensor de ultrasonido. Posteriormente, se relanza el nodo con el nuevo modo seleccionado. Por ejemplo, el sensor de ultrasonido puede operar en el modo listen. Para aplicar este cambio, se modifica el parámetro correspondiente:

```
/ev3dev/Ev3ultrasonic0/mode
```

Una vez actualizado el modo, se lanza nuevamente el nodo, repitiendo el procedimiento inicial

3.3.7 MÓDULO DE EXPERIMENTACIÓN Y TELEOPERACIÓN

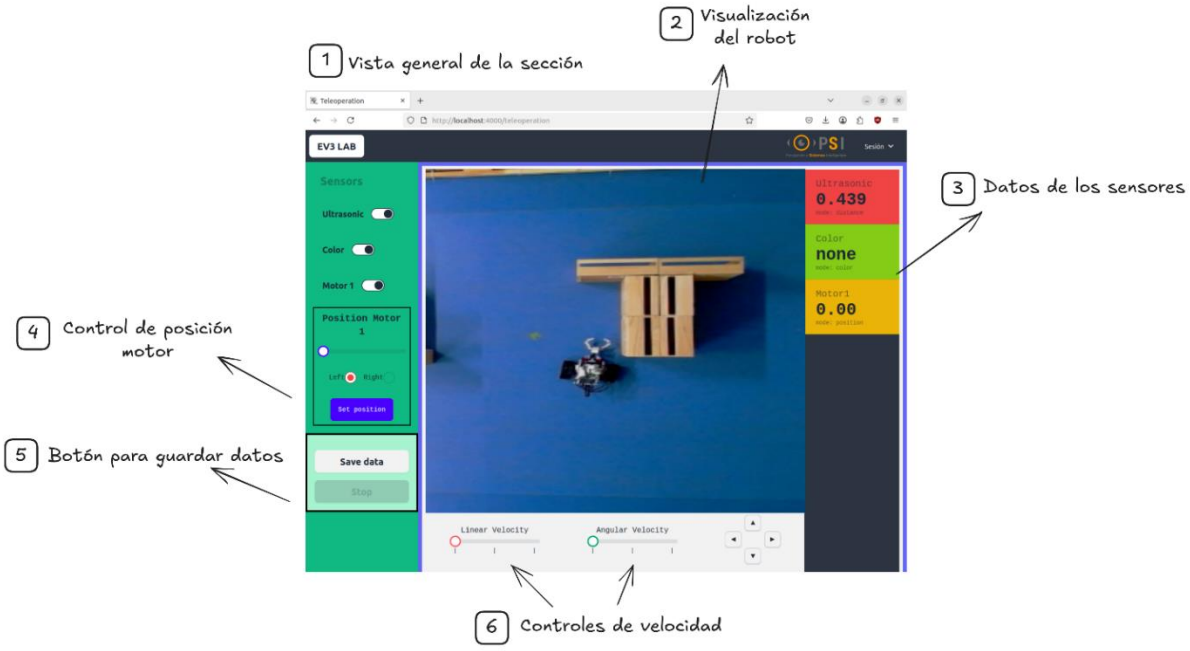


Figura 3.18. Sección de teleoperación.

Cuando un usuario accede a las secciones de teleoperación y ejercicios (Figura 3.18 y Figura 3.19 respectivamente), la herramienta de gestión inicia el proceso de preparación para establecer la comunicación con el robot, durante el cual la interfaz se bloquea. En primer lugar, se verifica que haya conexión con el `EV3` se verifica la conexión con el EV3 a través de su servidor web, siguiendo el mismo procedimiento utilizado en la sección de pruebas. Luego, se lee la configuración del usuario, a partir de la cual se determina qué sensores y motores se utilizarán, los puertos a los que están conectados y los modos en que se van a configurar. A través de Socket.IO, se invocan los métodos del objeto **EV3** mediante cadenas de texto, de forma similar a lo realizado en la sección de pruebas, con el objetivo de actualizar el objeto que representa al robot con la información correspondiente.

El objeto de la clase `EV3` lanza los nodos necesarios, pero no antes de detener todos los nodos que controlan los sensores y motores invocando los métodos de `launch_sensors()` y `launch_motors()`, con el fin de evitar posibles conflictos. Estos métodos se encargan de publicar un mensaje al tópico **`/launch`** del nodo `ev3_node_launch` que describe el *launch file* que debe ejecutarse; en este caso, los *launch files* corresponden a los sensores y motores. A continuación, se presentan dos ejemplos de los mensajes que se publican en el tópico **`/launch`** para lanzan los *launch files* correspondientes a los sensores y a los motores, respectivamente.

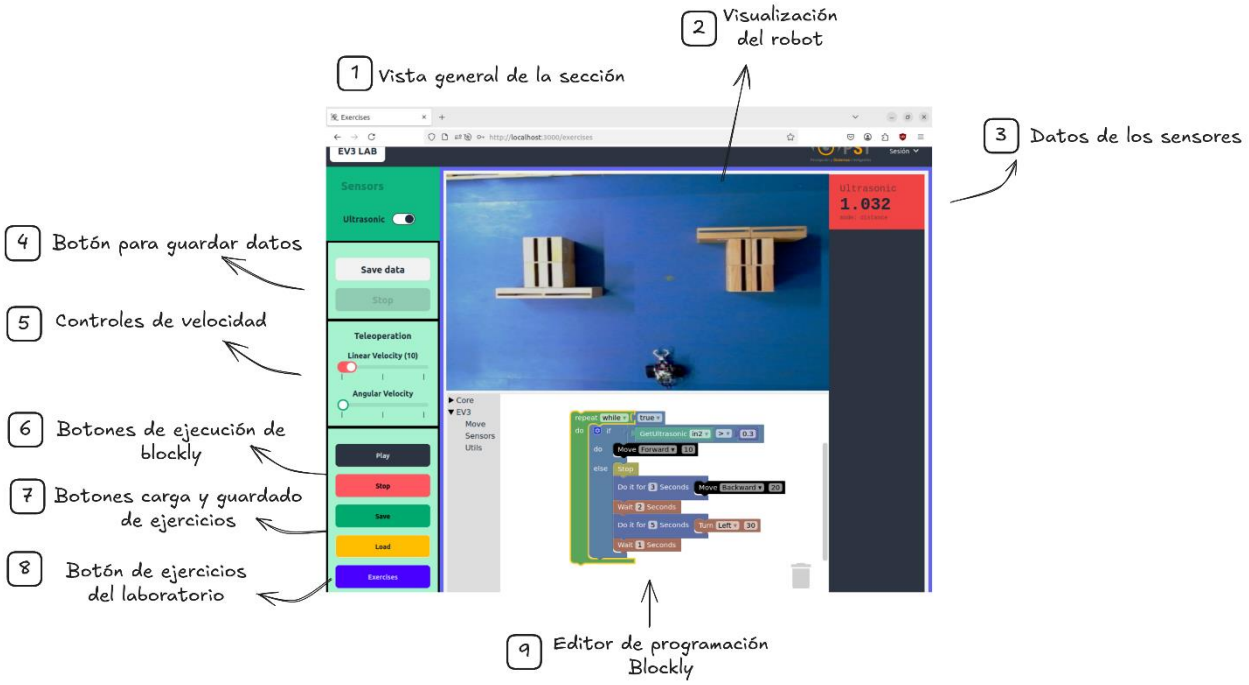


Figura 3.19 Sección de ejercicios.

```
ev3_sensors generic.launch sensors:=<sensores>
ev3_sensors motors.launch motors:=<motores>
```

Una vez que se asegura que los nodos y los tópicos correspondientes han sido lanzados correctamente, la interfaz se desbloquea. Ahora, la información de los sensores es visible y el usuario puede proceder a tele-operar el robot mediante el teclado.

Para acceder a los ejercicios, el usuario puede hacer clic en el botón `Exercises` (Figura 3.19 apartado 8), que muestra todos los ejercicios disponibles. Al seleccionar un ejercicio, este se carga en el espacio de trabajo de `Blockly`, siempre y cuando el ejercicio utilice los mismos sensores que la configuración actual.

`Blockly` es un editor de programación visual desarrollado por Google que permite crear programas mediante bloques de arrastrar y soltar. Se utiliza para generar la lógica de los ejercicios, devolviendo una cadena de texto que luego se ejecuta en el entorno de JavaScript. Aunque `Blockly` ofrece bloques básicos para programación, se han creado bloques

personalizados para el laboratorio, específicamente diseñados para el manejo del robot (Figura 3.19 apartado 9).

Para ejecutar un ejercicio, se dispone de un botón `Play`, y para detener su ejecución, se utiliza el botón `Stop`. La lógica de los ejercicios se procesa en el navegador, mientras que las acciones necesarias en el robot se envían a través de `Socket.IO` al servidor web y, posteriormente, a la clase EV3.

En esta sección también es posible teleoperar el robot utilizando las teclas W, A, S y D (Figura 3.19 apartado 5). Al teleoperar el robot, la clase EV3 gestiona los comandos de teleoperación a través del nodo `differential_drive`. Este nodo es usado para el control de tracción diferencial, qué es una técnica común utilizada en robots móviles que permite a un robot moverse controlando las velocidades independientes de sus dos ruedas motrices. La clase `EV3` entonces le envían comandos de velocidad a este nodo para controlar el desplazamiento del robot.

Las figuras 3.18 y 3.19 muestran las secciones de teleoperación y ejercicios, respectivamente. Ambas comparten características comunes, como el apartado 2, que incluye la visualización del robot a través de la cámara, controles de velocidad para la teleoperación, visualización de datos de los sensores y un botón para el guardado de datos.

La diferencia principal entre ambas secciones radica en su funcionalidad. En la sección de teleoperación, se ofrece una vista más amplia del robot y se implementa el control de posición de motores, siempre que alguno de ellos esté configurado para este modo.

Por otro lado, la sección de ejercicios incluye elementos adicionales, como botones para la ejecución de lógica mediante Blockly, botones para cargar y guardar ejercicios creados por el usuario, un botón para cargar ejercicios predefinidos por el laboratorio, y el espacio del editor Blockly donde se programa mediante bloques (Figura 3.19 apartados 6, 7, 8 y 9 respectivamente).

Siguiendo con el ejemplo presentado en la sección de pruebas, donde el usuario tiene la configuración **Baller** (Figura 3.13), una vez finalizada la prueba de sensores, se dispone a teleoperar el robot.

Teleoperación

Cuando usuario decide ahora ingresar a la sección de Ejercicios o Teleoperación, se ejecutará un procedimiento similar al de la sección de pruebas.

Inicialmente, la interfaz se bloqueará para evitar cualquier interacción del usuario. Luego, se verificará la conexión con el robot, contactando su servidor web y comprobando el estado de ejecución del nodo **`Ev3ControlNode`**. Si el nodo no está activo, el servidor del laboratorio lo lanzará automáticamente. A continuación, se cargará la configuración almacenada en la base de datos del usuario.

En este punto hay una diferencia importante respecto a la sección de pruebas: ya no se trabaja con un solo sensor, sino con todos los dispositivos definidos en la configuración. Por tanto, es necesario agregar todos los sensores y actuadores a la instancia del robot. Por ejemplo, si se utiliza la configuración **Baller**, se añadirán los sensores de ultrasonido, color y un motor en modo posición.

Los comandos correspondientes son generados desde la interfaz y enviados al servidor a través de WebSocket, donde se encuentra la instancia del objeto EV3.

Antes de iniciar nuevos procesos, se detienen los posibles nodos activos de usos anteriores, invocando los métodos **`kill_sensors()`** y **`kill_teleoperation()`.** Este último funciona de manera similar a **`kill_sensors()`**, pero se aplica específicamente a los nodos relacionados con la teleoperación. Publica en el tópico **`/kill`** los identificadores de los nodos a detener:

Una vez detenidos los nodos, se configuran nuevamente los sensores y sus correspondientes funciones de procesamiento de mensajes. A continuación, se lanzan los sensores con el método **`launch_sensors()`**, de forma similar a como se hace en la sección de pruebas. Esto activa el nodo **`ev3_sensor_spawner`**.

Los sensores se configuran de acuerdo con los modos definidos por el usuario. En la configuración Baller, por ejemplo, el sensor de ultrasonido se establece en modo **`distance`** y el sensor de color en modo **`color`**. Como resultado, los tópicos correspondientes serían:

- **`/ultrasonic_distance0`**
- **`/color_color0`**

Simultáneamente, se lanza el launch configurado para los motores, el cual crea el nodo **`controller_spawner`**, responsable del control motor. Posteriormente, se invoca el método **`launch_teleoperation()`**, que publica en el tópico `/launch` el siguiente comando:

```
ev3_sensors teleop.launch nodes:="OutPortState +motores+ diffDrv"
```

Este comando ejecuta el roslaunch con los nodos definidos, donde motores representa todos los motores configurados. En la configuración Baller, hay tres motores: dos en modo **`wheel`** (ruedas) y uno en modo position. Los motores en modo **`wheel`** serán utilizados como ruedas del robot.

Este launch lanza el nodo **`controller_spawner`**, encargado de gestionar los motores, y además activa el controlador de manejo diferencial (**`diff_drive_controller`**), parte del paquete **`controller_manager`** de ROS. Este controlador requiere la configuración de parámetros como nombre de las ruedas, radio de las ruedas, separación entre ellas etc…

Estos valores se encuentran definidos en un archivo **YAML** asociado al ***launch***. En la instancia del robot, las ruedas se establecen mediante los parámetros:

- **`/ev3dev/diffDrv/left_wheel`**
- **`/ev3dev/diffDrv/right_wheel`**

En este ejemplo, las ruedas están conectadas a los puertos A y B, por lo que se configuran como **Joint_A** y **Joint_B**.

El estado de los motores se publica en el tópico **`/joint_state`**, donde se transmite información de **velocidad** y **posición**. Estos datos son utilizados tanto para el motor en modo posición como para el controlador de manejo diferencial.

Luego de lanzar el nodo de teleoperación, se realizan las suscripciones a todos los tópicos correspondientes, tanto de sensores como de motores. Para este último, la suscripción se hace al tópico **`/joint_state`**, filtrando la información para el motor en modo posición.

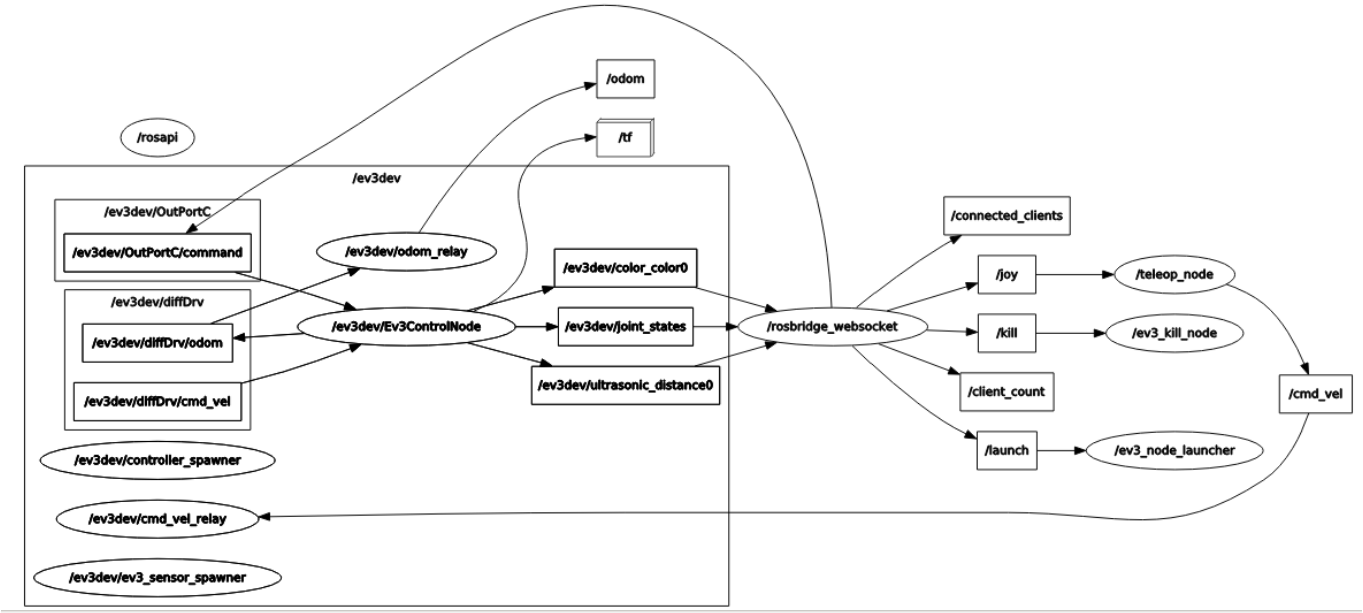


Figura 3.20. Nodos y tópicos del proceso.

La Figura 3.20 muestra los nodos utilizados y los tópicos generados durante el proceso del uso del laboratorio para este ejemplo. Los nodos incluyen **`/ros_bridge_websocket`**, perteneciente al paquete **`rosbridge_server`**, encargado de habilitar la comunicación entre ROS y aplicaciones externas mediante el protocolo WebSocket. El nodo **`ev3dev/controller_spawner`** es responsable de cargar el controlador de tracción diferencial (**`diff_drive_controller`**) del EV3. El nodo **`/cmd_vel_relay`** redirige los comandos de velocidad desde el tópico estándar **`/cmd_vel`** hacia **`/ev3dev/diffDrv/cmd_vel`**, mientras que el nodo **`/ev3dev/odom_relay`** redirige la odometría desde **`/ev3dev/diffDrv/odom`** hacia **`/odom`**. También se ejecuta el nodo **`/ev3dev/ev3_sensor_spawner`**, que se encarga de inicializar y administrar los sensores conectados al EV3. Además, el nodo **`ev3_kill_node`** permite finalizar la ejecución de otros nodos, y **`ev3_launch_node`** con el que se ejecutan nuevos nodos mediante **`launch files`**.

Una vez completadas estas configuraciones, se **desbloquea la interfaz**, permitiendo al usuario interactuar.

Control del robot vía teclado

En las secciones de Ejercicios y Teleoperación, es posible controlar el robot mediante el teclado, utilizando las teclas W, A, S y D.

Cada vez que el usuario presiona una de estas teclas, se publica un mensaje en el tópico **`/cmd_vel`** con la siguiente estructura:

```
{
  linear: {x: 0, y: 0, z: 0},
  angular: {x: 0, y: 0, z: 0}
}
```

Por ejemplo, al presionar la tecla **W**, se modifica el valor de `linear.x` para que el robot avance:

```
{
  linear: {x: velocidad, y: 0, z: 0},
  angular: {x: 0, y: 0, z: 0}
}
```

La variable velocidad es proporcional a un valor entre **0 y 100**, que el usuario puede ajustar desde la interfaz. Este valor se normaliza a un rango de **0.0 a 1.0** para ROS. Un valor positivo hace que el robot avance, y uno negativo que retroceda.

Para rotación (teclas **A** o **D**), se usa el campo `angular.z`:

```
{
  linear: {x: 0, y: 0, z: 0},
  angular: {x: 0, y: 0, z: velocidad}
}
```

Programación del robot

Si el usuario desea practicar algún ejercicio o programar el robot, debe dirigirse al área de programación por bloques, disponible únicamente en la sección de ejercicios. Por ejemplo, si el usuario selecciona el ejercicio 4.1, ejercicios de la configuración **Baller**; se cargará automáticamente el diagrama de bloques correspondiente.

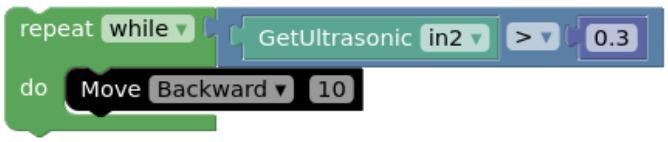


Figura 3.21. Ejercicio 4.1

En el ejercicio 4.1 (Figura 3.21) se propone un programa que hace que el robot avance mientras la distancia medida por el sensor ultrasónico sea mayor a 0.3 metros. `Blockly` ofrece bloques

lógicos, bucles, operaciones matemáticas, listas y variables. En este caso, se utiliza principalmente el bloque **`do while`**, que ejecuta acciones mientras se cumpla una condición.

La condición del bucle es que la distancia medida por el sensor ultrasónico conectado al puerto **in2** sea mayor a 0.3 m. Si se cumple, el robot avanza. Para este ejercicio se utilizan dos bloques diseñados específicamente para el laboratorio: **`GetUltrasonic`** y **`Move`**.

- **`GetUltrasonic:`** retorna la distancia medida por el sensor, la misma que se muestra en la interfaz y se publica en el nodo del sensor.
- **`Move:`** simula la lógica utilizada en la teleoperación por teclado, publicando en el tópico **`/cmd_vel`** el valor correspondiente a la velocidad lineal (en este caso el valor de 10). En este bloque también se selecciona la orientación; en el ejercicio se usa la opción "backwards" (hacia atrás), ya que la construcción del robot tiene la dirección de los motores invertida.

Cuando la condición deja de cumplirse, se detiene la publicación en el tópico **`/cmd_vel`**, el robot se detiene y el programa finaliza. Blockly convierte esta lógica en código JavaScript que se ejecuta directamente en el navegador del usuario. El código generado es el siguiente:

```
while (GetUltrasonicDistanceValue("in2") > 0.3) {
  await new Promise((resolve) => {
    setTimeout(resolve, 30);
  });
  if (stopBlocklyExecutionCode) { break; }
  moveEV3(-10);
}
```

Las funciones **`GetUltrasonicDistanceValue()`** y **`Move()`** implementan la lógica mencionada. La variable **`stopBlocklyExecutionCode`** se utiliza para detener la ejecución al presionar el botón **`Stop`** de la interfaz.

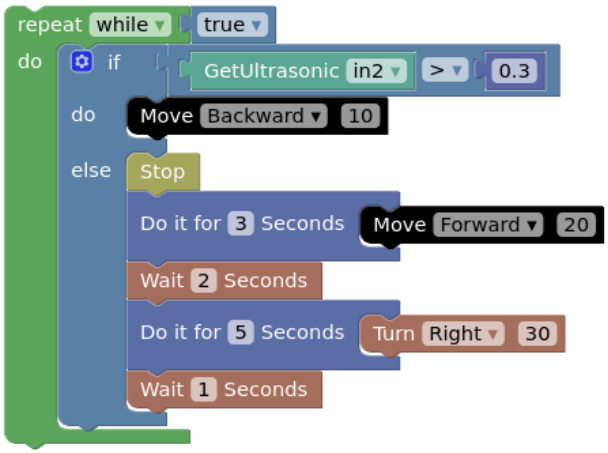


Figura 3.22. Ejercicio 4.2

A continuación, el usuario decide realizar el ejercicio 4.2 (Figura 3.22), que propone una lógica para evasión de obstáculos. La secuencia es la siguiente:

- Si la distancia medida es mayor a 0.3 m, el robot avanza.
- En caso contrario, se detiene, retrocede durante 3 segundos, espera 2 segundos, gira a la derecha durante 5 segundos con velocidad 30, y luego espera 1 segundo.

Nuevos bloques en este ejercicio:

- **`Turn`**: publica en el tópico **`/cmd_vel`** una velocidad angular.
- **`Do it for`**: ejecuta una acción durante un tiempo determinado.
- **`Wait`**: simplemente espera el tiempo especificado.

El código JavaScript generado es el siguiente:

```
while (true) {
  await new Promise((resolve) => {
    setTimeout(resolve, 30);
  });
  if (stopBlocklyExecutionCode) { break; }
  if (GetUltrasonicDistanceValue("in2") > 0.3) {
    moveEV3(-10);
  } else {
    stopEV3();
    await doItForTime(3, moveEV3, 20);
    await wait(2);
    await doItForTime(5, turnEV3, -30);
    await wait(1);
  }
}
```

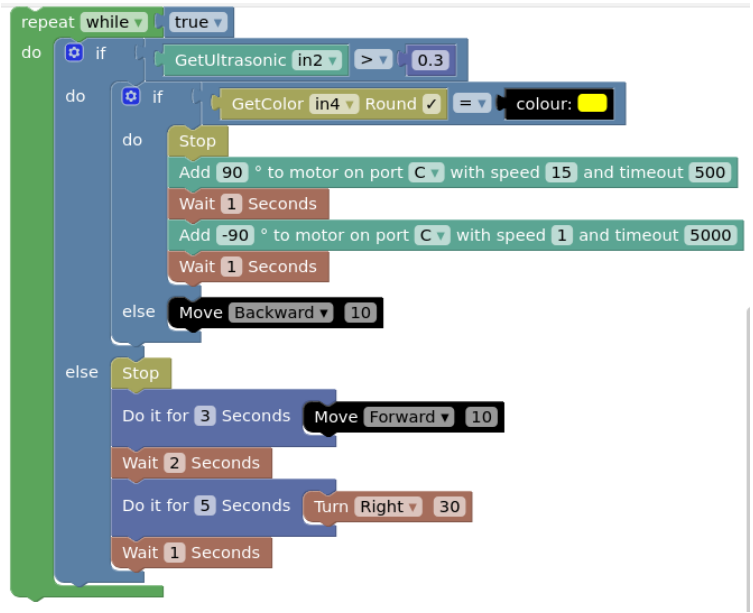


Figura 3.23. Ejercicio 4.3

Por último, el usuario escoge el ejercicio 4.3 (Figura 3.23), donde el objetivo es que el robot evite obstáculos y lance pelotas de color amarillo. La lógica es la siguiente:

- Si la distancia medida por el sensor ultrasónico es mayor a 0.3 m:
- Se lee el color detectado por el sensor conectado al puerto in4.
- Si el color es amarillo, el robot se detiene, gira el motor conectado al **puerto C** (en modo posición) 90 grados con velocidad 15 y un tiempo de espera de 500 ms. Luego espera 1 segundo, regresa -90 grados con velocidad 1 y espera otros 5 segundos. Finalmente, espera 1 segundo más.
- Si el color no es amarillo, el robot simplemente avanza.
- Si la distancia es menor o igual a 0.3 m, se ejecuta la misma rutina de evasión del ejercicio 4.2.

Nuevos bloques utilizados:

- **`GetColor`**: retorna el valor del color detectado por el sensor.
- **`Add angle° to motor on port X`**: controla la posición de un motor utilizando un pequeño lazo de control basado en la información publicada en el tópico **`/joint_state`**.

Como no se pudo usar un nodo de control de posición directamente, se implementó un control propio. Se publica la velocidad en el tópico correspondiente al motor (por ejemplo, **`/OutPortC/command`**) y se ajusta hasta que el error angular sea menor a 0.1. Si no se alcanza la posición en el tiempo establecido, el control se detiene automáticamente.

El código generado es el siguiente:

```
while (true) {
  await new Promise((resolve) => {
    setTimeout(resolve, 30);
  });
  if (stopBlocklyExecutionCode) { break; }
  if (GetUltrasonicDistanceValue("in2") > 0.3) {
    if (GetColorColorValue("in4") == '#ffff00') {
      stopEV3();
      await SetMotorPosition(0, 90, 15, 500);
      await wait(1);
      await SetMotorPosition(0, -90, 1, 5000);
      await wait(1);
    } else {
      moveEV3(-10);
    }
  } else {
    stopEV3();
    await doItForTime(3, moveEV3, 10);
    await wait(2);
    await doItForTime(5, turnEV3, -30);
    await wait(1);
  }
}
```

En el código generado se pueden observar las funciones `SetMotorPosition()` para el control de posición del motor, y `GetColorColorValue()` para la lectura del sensor de color, cuyo valor se compara con el código hexadecimal correspondiente al color amarillo (#ffff00).

3.4 CONCLUSIONES

A pesar de algunos inconvenientes durante el desarrollo, el laboratorio cumple con los requerimientos funcionales establecidos inicialmente en la fase de diseño. Entre los requerimientos satisfechos se encuentran: la autenticación de usuarios, la reserva del laboratorio, la configuración y el uso de sensores y motores, la teleoperación del robot, la ejecución de ejercicios propuestos y la recolección de datos provenientes de los sensores.

El lanzamiento de nodos en el EV3 debe realizarse de manera controlada, ya que se trata de un hardware con recursos limitados. Ejecutar múltiples nodos simultáneamente puede generar problemas de rendimiento o inestabilidad en el sistema, por lo que es fundamental gestionar su despliegue con precaución para evitar sobrecargar el sistema operativo del EV3.

A pesar del alto consumo de recursos de ROS, su uso como eje central de control del robot proporciona una amplia variedad de herramientas y ventajas para el desarrollo y la integración de sistemas robóticos. ROS facilita la comunicación entre nodos, el manejo de sensores y actuadores, y permite explotar herramientas que la comunidad de ROS ha desarrollado.

Los nodos desarrollados para el EV3 presentan algunos fallos que afectan su funcionalidad y revelan ciertas limitaciones. Uno de los principales problemas está relacionado con el sensor de color en su modo **`rgb_raw`**, el cual provoca una ralentización considerable del sistema operativo del EV3. Además, si el sensor se conecta antes de encender el robot, puede impedir que el sistema operativo inicie correctamente.

Otro inconveniente se presenta al utilizar el sensor de color en modo color, ya que esto provoca que el script **`ev3_manager`** el componente principal responsable de la comunicación entre el EV3 y ROS, falle y se cierre inesperadamente. Inicialmente, este problema requería reiniciar manualmente el script. Sin embargo, se implementó una solución mediante el servidor, que detecta si el nodo no está en ejecución y lo relanza automáticamente.

Asimismo, se identificó como una limitación el hecho de que los nodos de ROS no permiten realizar control directo de posición sobre los motores. Para mitigar esta carencia, se propuso un sistema de control basado en la información de posición proporcionada por el propio nodo de control de motores.

La versatilidad de ser un laboratorio basado en una interfaz web permite al usuario despreocuparse por la instalación de dependencias y configuraciones en su propio equipo. Esta característica facilita significativamente el acceso y uso del entorno, pudiendo ser utilizada desde cualquier computador conectado a la red.

La incorporación de Blockly para la programación basada en bloques brinda mayor libertad y accesibilidad en la experimentación con el robot. Combinado con el entorno de ROS, constituye una herramienta didáctica para el aprendizaje de la robótica, sin profundizar mucho en conceptos internos de ROS y programación.

CAPÍTULO 4.

PRUEBAS DE FUNCIONAMIENTO DEL LABORATORIO

4. PRUEBAS DE FUNCIONAMIENTO DE LABORATORIO

4.1 INTRODUCCIÓN

En esta sección se presentan una serie de pruebas realizadas en el laboratorio. Primero, se describen las pruebas de integración llevadas a cabo por el autor del proyecto, seguidas de pruebas realizadas con estudiantes, orientadas a evaluar el cumplimiento de los requerimientos funcionales. Se explicará en qué consistió cada prueba, se presentarán los resultados obtenidos y se realizará un análisis correspondiente.

4.2 PRUEBAS DE INTEGRACIÓN

Esta sección presenta una evaluación de la funcionalidad del laboratorio, la cual se llevó a cabo en dos etapas: primero, mediante pruebas realizadas por el autor del proyecto, y luego mediante una prueba con estudiantes haciendo recorrido a través de todo el laboratorio, esto, con el fin de analizar la aceptación y la percepción del proyecto.

4.2.1 PRUEBAS DE FUNCIONALIDAD

Las pruebas de funcionalidad fueron realizadas por el autor de este trabajo con el objetivo de evaluar el cumplimiento de los requerimientos funcionales (Tablas 4.1 – 4.6). Los requerimientos funcionales son: La autenticación de usuarios, la configuración de sensores y motores, la reserva del laboratorio, la prueba de sensores, así como la recolección de datos, la teleoperación del robot y la ejecución de ejercicios propuestos.

Tabla 4.1. Prueba de integración - Acceso al laboratorio.

PRUEBA DE INTEGRACIÓN – ACCESO AL LABORATORIO	
Requerimientos funcionales de la prueba: • Permitir crear una cuenta de usuario. • Permitir iniciar sesión a un usuario.	
Tipo de Prueba	Integración
Hardware Requerido	Computador
Software Requerido	Navegador web
Objetivo	Esta prueba tiene como objetivo verificar el acceso al laboratorio por medio del navegador web y el funcionamiento del manejo de usuarios.
ACCESO Y MANEJO DE USUARIOS	

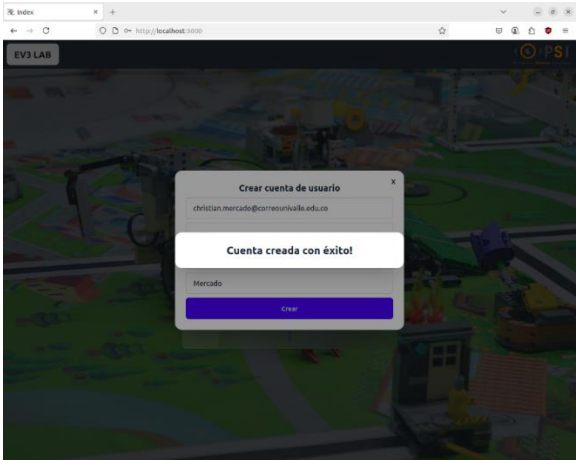
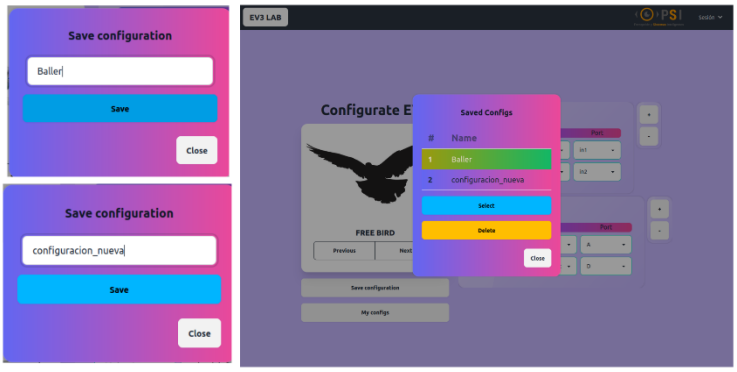
Datos de la Prueba	• Datos de usuario. • Login del usuario. • Clave de acceso.
Procedimiento	1. Ingreso al laboratorio por medio del navegador web 2. Creación de una cuenta 3. Ingreso con las credenciales de la cuenta creada.
Resultado Esperado	a. Acceso a la página web del laboratorio. b. Creación de la cuenta nueva c. Ingreso del usuario con sus credenciales.
Resultado Obtenido	a. Ingreso al laboratorio por medio del navegador web. b. Se creó una cuenta de usario existosamente ingresando los datos requeridos (correo, contraseña, nombre y apellidos)  c. Con las credenciales de la cuenta creada (correo y contraseña) se pudo ingresar al labotratorio. 

Tabla 4.2. Prueba de integración – Gestión de configuraciones de usuario.

PRUEBA DE INTEGRACIÓN – GESTIÓN DE CONFIGURACIONES
Requerimientos funcionales de la prueba: • Permitir al usuario escoger entre configuraciones predefinidas. • Permitir al usuario crear una configuración propia. • Seleccionar una configuración

• Eliminar una configuración	
Tipo de Prueba	Integración
Hardware Requerido	Computador
Software Requerido	Navegador web, Linux, Docker
Objetivo	Esta prueba tiene como objetivo verificar el funcionamiento de la sección de configuración del laboratorio, verificando que el usuario pueda escoger una configuración, crear una propia, guardarla o eliminarla.
GESTIÓN DE CONFIGURACIONES DE USUARIO	
Datos de la Prueba	• Hora y fecha de la reserva.
Procedimiento	1. Escoger una configuración predefinida y guardarla 2. Crear una configuración nueva y guardarla. 3. Seleccionar la primera configuración guardada. 4. Eliminar la segunda configuración.
Resultado Esperado	a. Configuraciones guardadas y selección de configuración. b. Eliminación de configuración exitosa.
Resultado Obtenido.	a. Se ingresó a la seccion de configuraciones, se creó una configuración y se guardó. La configuracion es gurdada exitosamente.  b. Se selecciona la configuración anteriormente hecha y se le da eliminar. La configuración es eliminada exitosamente.

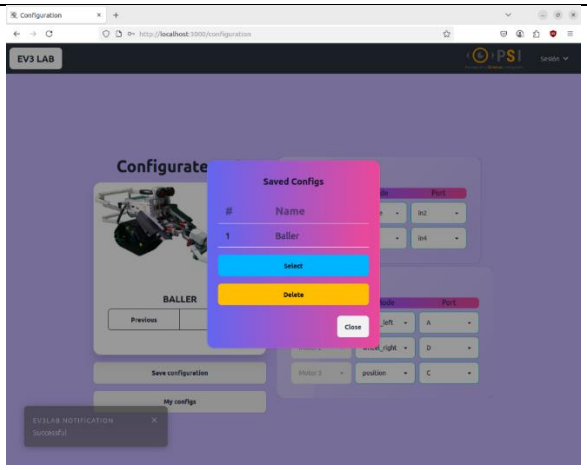
	

Tabla 4.3. Prueba de integración - Gestión de reservas.

PRUEBA DE INTEGRACIÓN – GESTIÓN DE RESERVAS	
Requerimientos funcionales de la prueba: • Permitir al usuario reservar el laboratorio en un horario disponible. • Mostrar al usuario sus reservas. • Seleccionar y eliminar una reserva.	
Tipo de Prueba	Integración
Hardware Requerido	Computador
Software Requerido	Navegador web
Objetivo	Esta prueba tiene como objetivo validar que el usuario puedo realizar la reserva del laboratorio en el horario deseado.
GESTIÓN DE RESERVAS	
Datos de la Prueba	• Hora y fecha de la reserva.
Procedimiento	1. Ingresar a la sección de reservas. 2. Realizar una reserva el día 12 de junio de 2025 a las 10:00 3. Eliminar la reserva.
Resultado Esperado	a. Reserva realizada exitosamente en la fecha indicada. b. Eliminación de la reserva.
Resultado Obtenido	a. Se ingresa a la sección de reserva y se realiza una reserva para el dia 12 de junio de 2025 a las 10:00. Esta se realiza con éxito.

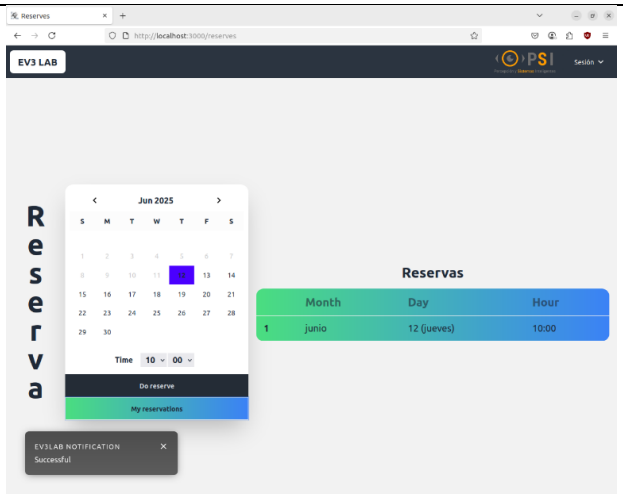
	 b. Se ingresa a las reservas hechas, se seleccionó la reserva anteriomente hecha y se le dió eliminar. La reserva se eliminó exitosamente.
Comentarios	Ninguno.

Tabla 4.4. Prueba de integración - Prueba de sensores.

PRUEBA DE INTEGRACIÓN – PRUEBA DE SENSORES	
Requerimientos funcionales de la prueba: • Permitir seleccionar los sensores que se desean probar. • Permitir ejecutar pruebas sobre los sensores seleccionados. • Proporcionar una visualización de los cambios en las variables del sensor. • Permitir al usuario guardar datos del sensor.	
Tipo de Prueba	Integración
Hardware Requerido	Computador, Lego MINDSTORM EV3
Software Requerido	Navegador, Linux, Docker
Objetivo	Esta prueba tiene como objetivo verificar que el usuario pueda comprobar el funcionamiento de los sensores incluidos en la configuración del usuario.
PRUEBA DE SENSORES	
Datos de la Prueba	NO APLICA
Procedimiento	1. Ingresar a la sección de pruebas. 2. Escoger un sensor a probar. 3. Verificar el funcionamiento del sensor. 4. Guardar datos del sensor.
Resultado Esperado	a. Visualización de la de la animación e información del sensor. b. Datos guardados del sensor.
Resultado Obtenido	a. Se escogió como sensor a probar el sensor de ultrasonido. En la interfaz se cargo este sensor, se visulizó de su valor medido y la animacion respectiva.

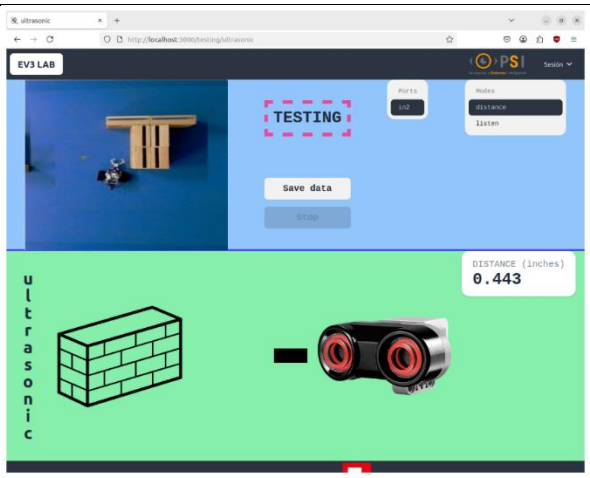
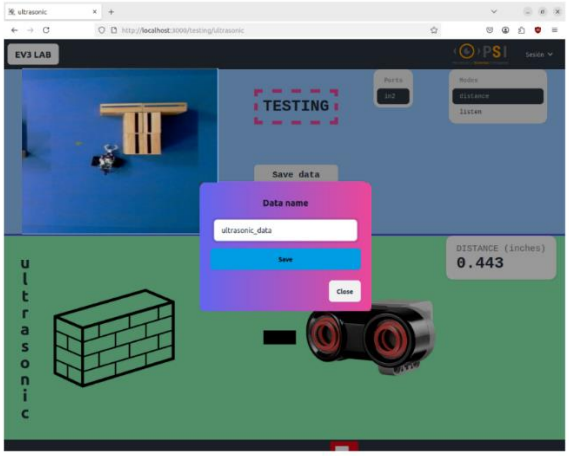
	 b. Se le dio clic en el bóton Save data, s guardó la información del sensor y esta pudo ser descargada en la sección de reservas. 
Comentarios	Ninguno.

Tabla 4.5. Prueba de integración - Teleoperación

PRUEBA DE INTEGRACIÓN – TELEOPERACIÓN	
Requerimientos funcionales de la prueba: • La herramienta de gestión debe permitir el control del robot móvil mediante entradas del teclado. • La herramienta de gestión debe permitir variar tanto la velocidad lineal como angular.	
Tipo de Prueba	Integración
Hardware Requerido	Computador, Lego MINDSTORM EV3
Software Requerido	Navegador, Linux, Docker
Objetivo	Esta prueba tiene como objetivo verificar que el usuario puede teleoperar el robot mediante el teclado, comprobando su capacidad de responder a comandos de velocidad lineal y angular.
TELEOPERACIÓN	

Datos de la Prueba	NO APLICA
Procedimiento	1. Ingresar a la sección de teleoperación. 2. Teleoeperar el robot con las teclas W,A,S,D. 3. Variar tanto la velocidad lineal como la angular.
Resultado Esperado	a. Desplazamiento del EV3 en todas las direcciones. b. Variaciones de velocidad al modificar los valores correspondientes a la velocidad lineal y velocidad angular.
Resultado Obtenido	a. Al ingresar a la sección teleoperación, modificar los valores de la velocidad lineal y velocidad angular a un valor inicial. Presionando las W,A,S,D el robot respondió desplazándose. b. Se modificó el valor de las velocidades a diferentes valores y el robot respondió de acuerdo con la variación hecha.
Comentarios	Ninguno.

Tabla 4.6. Prueba de integración - Ejercicios.

PRUEBA DE INTEGRACIÓN - EJERCICIOS	
Requerimientos funcionales de la prueba: • La herramienta de gestión debe incluir una sección específica destinada a ejercicios prácticos. • La herramienta de gestión debe permitir guardar, cargar, y eliminar ejercicios propios del usuario. • La herramienta de gestión debe permitir modificar ejercicios prácticos.	
Tipo de Prueba	Integración
Hardware Requerido	Computador, Lego MINDSTORM EV3
Software Requerido	Navegador, Linux, Docker
Objetivo	Esta prueba tiene como objetivo verificar que el usuario puede escoger entre los ejercicios prácticos definidos, hacer sus propios ejercicios, guardarlos y eliminarlos.
PRUEBA DE EJERCICIOS	
Datos de la Prueba	NO APLICA
Procedimiento	1. Ingresar a la sección de ejercicios, cargar un cargar un ejercicio y ejecutarlo. 2. Modificar este ejercicio y guardarlo. 3. Eliminar el ejercicio.
Resultado Esperado	a. Carga del ejercicio y lógica ejecutada. b. Modificación hecha y ejercicio guardado. c. Eliminación del ejercicio.
Resultado Obtenido	a. Se ingresa a la sección de ejercicios, se escoge un ejercicio propuesto y se ejecuta. El robot realiza las acciones que describe la lógica. b. Se modifica el ejercicio anterior y se guarda. Se revisa los ejercicios del usuario y efectivamente el ejercicio se encuentra guardado. c. Se selecciona este ejercicio guardado y se le da en eliminar. El ejercicio es eliminado.
Comentarios	Ninguno.

4.2.2 PRUEBAS CON ESTUDIANTES

Para la realización de estas pruebas, se diseñó inicialmente una encuesta tipo Likert utilizando Google Forms. Este tipo de preguntas permite medir el nivel de acuerdo o desacuerdo de los usuarios respecto a la funcionalidad de la aplicación.

Una vez finalizada la encuesta, se invitó a seis estudiantes de la Universidad del Valle a probar el funcionamiento del laboratorio. Para ello, se instaló el laboratorio en una de las salas de cómputo de la Escuela de Ingeniería Eléctrica y Electrónica. A cada participante se le entregó un robot LEGO MINDSTORMS EV3 con una figura previamente ensamblada, y se les indicó realizar un recorrido completo por todas las secciones del laboratorio.

4.2.3 ANÁLISIS DE RESULTADOS

Dado que se trataba de un recorrido por el laboratorio, fue necesario crear una cuenta de usuario e ingresar al sistema. Por ello, primero se les pidió a los estudiantes que crearan una cuenta y accedieran al laboratorio con ella.

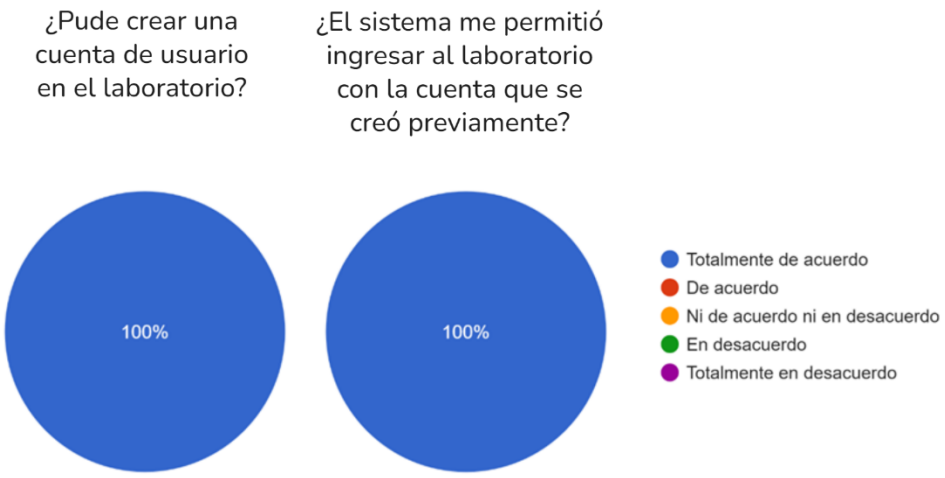


Figura 4.1. Resultado de encuesta - Creación de cuenta e ingreso al laboratorio.

En las figuras 4.1 se observa que el 100 % de las personas encuestadas manifestaron estar totalmente de acuerdo en que lograron registrarse en el laboratorio e ingresar a este utilizando sus credenciales. Lo anterior, confirma el buen funcionamiento de esta de característica.

Después de completar el registro e ingreso al laboratorio, se les pidió a los estudiantes seleccionar la configuración predefinida correspondiente a la figura ensamblada del EV3 con la que estaban trabajando. Posteriormente, se les solicitó crear una configuración propia desde cero. Finalmente, se les indicó guardar ambas configuraciones.

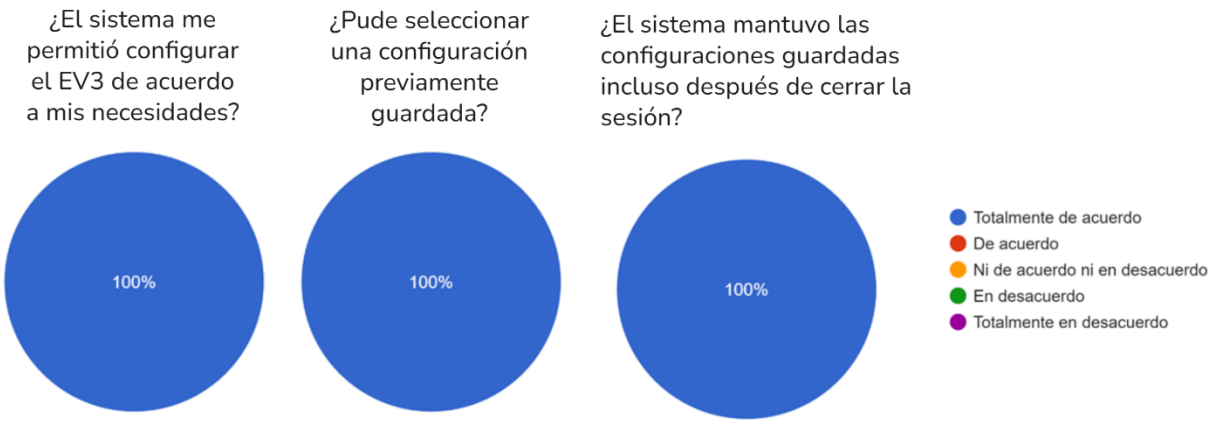


Figura 4.2. Resultado de encuesta - Configuraciones de usuario.

Como se muestra en la Figura 4.2, el 100 % de los encuestados manifestaron estar totalmente de acuerdo en lograron armar la configuración que necesitaba, seleccionar una configuración previamente guardada y confirmar que su configuración se mantenía almacenada después de cerrar la sesión.

Estos resultados reflejan una alta satisfacción con la funcionalidad de gestión de configuraciones del laboratorio, evidenciada por una coincidencia total en la opción "Totalmente de acuerdo" en las tres situaciones evaluadas.

Una vez finalizada la prueba de las funcionalidades de configuración, se solicitó a los estudiantes realizar una reserva del laboratorio como paso previo para acceder a las siguientes secciones.

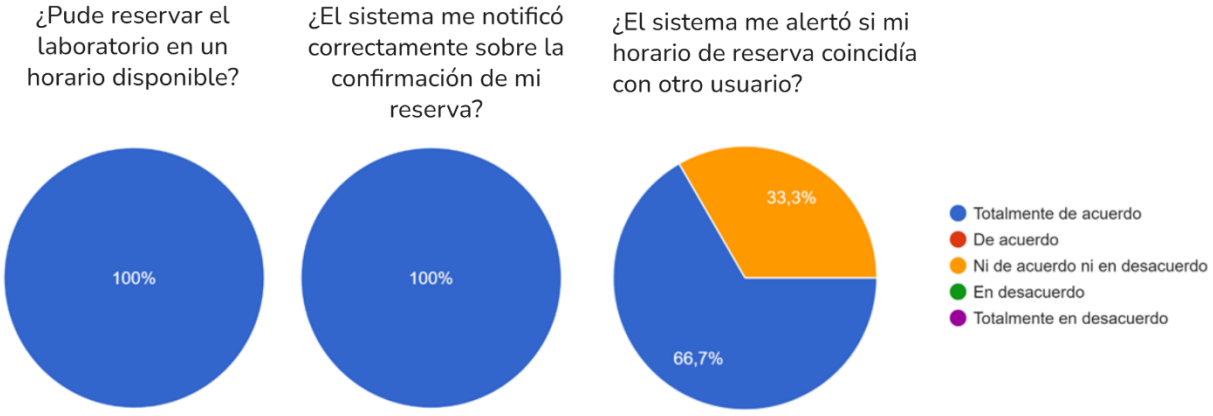


Figura 4.3. Resultado de la encuesta - Reserva del laboratorio.

En la Figura 4.3 se observa que el 100 % de los estudiantes manifestaron estar totalmente de acuerdo en que lograron reservar en el horario de su elección, y que el sistema les notificó que la reserva se realizó con éxito. Además, el 66,7 % indicó estar totalmente de acuerdo en que el sistema les informó cuando el horario seleccionado ya estaba ocupado, mientras que el 33,3 % expresó una postura neutral, seleccionando la opción "ni de acuerdo ni en desacuerdo" respecto

a esta funcionalidad. Estos resultados sugieren que el sistema de reservas cumple adecuadamente con su función principal, aunque existe un margen de mejora en la claridad o efectividad de las notificaciones cuando los horarios ya están ocupados.

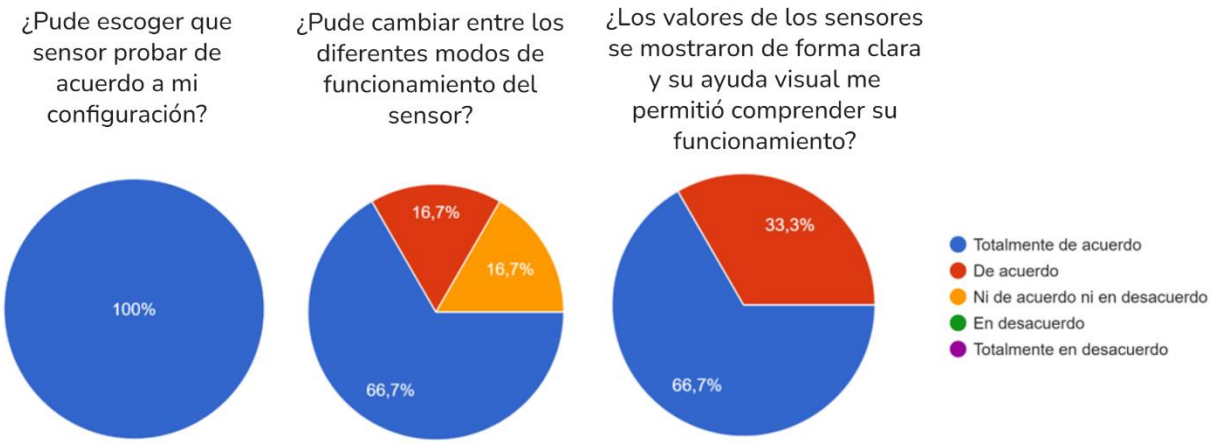


Figura 4.4. Resultado encuesta - Prueba de sensores.

Después de realizar la reserva del laboratorio, se indicó a los estudiantes que accedieran a la sección de pruebas para evaluar alguno de los sensores incluidos en su configuración.

En la Figura 4.4 se observa que el 100 % de los estudiantes manifestó haber podido seleccionar un sensor perteneciente a su configuración para realizar pruebas. Por otro lado, el 66,7 % indicó estar totalmente de acuerdo en que lograron cambiar el modo de operación del sensor utilizado, mientras que un 16,7 % estuvo de acuerdo y otro 16,7 % se mostró neutral, seleccionando la opción "ni de acuerdo ni en desacuerdo". Finalmente, el 66,7 % de los estudiantes señaló estar totalmente de acuerdo en que los valores del sensor fueron correctamente mostrados y que la animación correspondiente les ayudó a comprender su funcionamiento; el 33,3 % restante indicó estar de acuerdo con esta afirmación.

Estos resultados indican que, en general, la funcionalidad de prueba de sensores fue bien recibida y comprendida por la mayoría de los estudiantes. Sin embargo, la presencia de respuestas menos favorables en algunas áreas sugiere que podría mejorarse el funcionamiento o la claridad de ciertas funciones, como el cambio de modo del sensor y la interpretación de los resultados mostrados.

Tras finalizar las pruebas de sensores, se solicitó a los estudiantes acceder a la sección de teleoperación con el fin de evaluar el funcionamiento de esta característica.

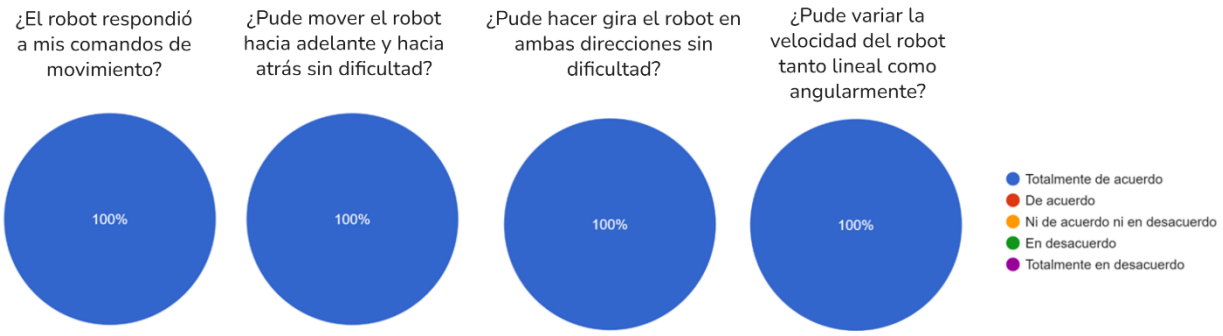


Figura 4.5. Resultado encuesta - Prueba de teleoperación.

Como se observa en la Figura 4.5, el 100 % de los estudiantes indicó estar totalmente de acuerdo en que el robot respondió correctamente a los comandos de movimiento, incluyendo desplazamiento hacia adelante y hacia atrás, giros, y la variación de la velocidad tanto lineal como angular.

Estos resultados evidencian que la funcionalidad de teleoperación del robot funciona de manera correcta y cumplió con la expectativa en cuanto respuesta y control.

Por último, después de teleoperar el robot, se indicó a los estudiantes que ingresaran a la sección de ejercicios para realizar algunos de ellos.

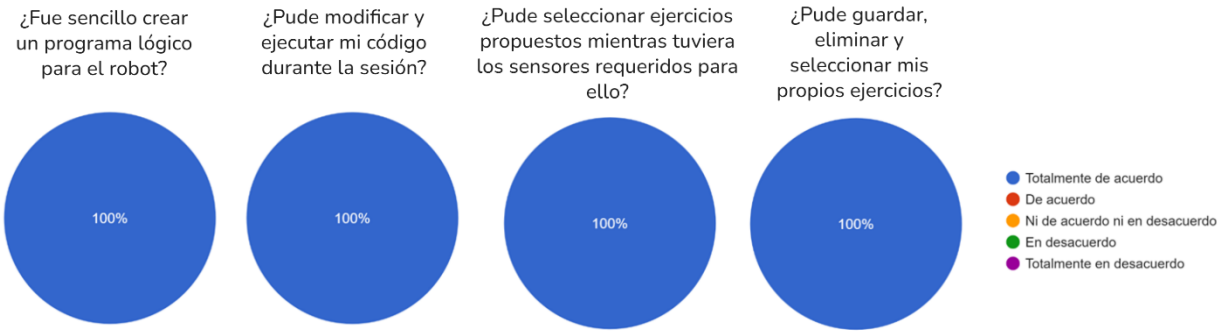


Figura 4.6. Resultado de encuesta - Realización de ejercicios.

Como se observa en la Figura 4.6, el 100 % de los estudiantes manifestó estar totalmente de acuerdo en que fue fácil crear un programa lógico para el robot, modificarlo y ejecutarlo, así como seleccionar ejercicios y gestionar sus propios ejercicios, es decir, guardarlos, seleccionarlos y eliminarlos.

Los resultados muestran que los estudiantes encontraron fácil crear, modificar y gestionar ejercicios, lo que refleja buen funcionamiento de esta característica.

4.3 CONCLUSIONES

Las pruebas de integración permitieron corroborar el correcto funcionamiento del laboratorio, tanto en el acceso como en las diferentes secciones del sistema. Además, estas pruebas ayudaron a identificar posibles problemas que pudieron ser corregidos antes de su uso generalizado por parte de los estudiantes.

Los resultados de la encuesta aplicada para evaluar las funcionalidades del laboratorio muestran que, en 16 de 19 preguntas el 100% de los encuestados expresó estar totalmente de acuerdo con las afirmaciones presentadas, esto indica que las funcionalidades evaluadas están cumpliendo con lo esperado. Por lo tanto, se puede afirmar que los requerimientos funcionales sometidos a prueba fueron alcanzados satisfactoriamente. Entre dichos requerimientos se incluyen: la autenticación de usuarios, la reserva del laboratorio, la configuración y uso de sensores y motores, la teleoperación del robot, la ejecución de ejercicios propuestos y la recolección de datos provenientes de los sensores.

La experiencia de los estudiantes durante el uso del laboratorio permitió confirmar una observación previa: si bien las funcionalidades relacionadas con el uso de sensores, la teleoperación y la ejecución de ejercicios fueron bien valoradas —con un 100% de respuestas de "totalmente de acuerdo" en 9 de las 12 preguntas correspondientes a estos apartados—, el hardware antiguo del EV3 presenta importantes limitaciones. Por ejemplo, el proceso de inicialización de los nodos puede tardar entre 1 y 2.5 minutos, lo cual retrasa el inicio de las actividades. Además, la batería del EV3, especialmente al operar con los motores activos, tiene una duración promedio de solo 2 a 2.5 horas, lo que restringe considerablemente la duración de las sesiones prácticas.

Por otro lado, la programación del EV3 mediante bloques fue muy bien valorada por los estudiantes. Aunque esta funcionalidad no formaba parte de los requerimientos funcionales iniciales —que se centraban en configuraciones fijas con parámetros modificables—, el uso de bloques proporciona una mayor libertad para experimentar, lo cual enriqueció la experiencia de los usuarios.

CAPÍTULO 5.

CONCLUSIONES

5. CONCLUSIONES

5.1 CONCLUSIONES

El presente proyecto tuvo como objetivo principal la creación de un laboratorio remoto para la teleoperación de kits LEGO Mindstorms EV3. El punto de partida fue la revisión de antecedentes tanto a nivel local como internacional, con el fin de identificar y extraer requerimientos funcionales y no funcionales. Este análisis se centró en trabajos previos relacionados con laboratorios remotos, especialmente aquellos que empleaban robots reales en lugar de simulación y hacían uso del framework ROS.

En los proyectos analizados se evidenció un uso predominante de tecnologías web, lo cual se tomó como base para la arquitectura del presente proyecto. Asimismo, se examinaron diversas características clave con el propósito de establecer similitudes y diferencias entre las propuestas estudiadas. Entre los criterios considerados se incluyeron: el tipo de robot utilizado, la integración con ROS, los lenguajes de programación empleados, los métodos de visualización del robot, las estrategias de programación implementadas, los ejercicios ofrecidos, el uso de bases de datos, y los sensores y actuadores integrados.

Como resultado del análisis de antecedentes, se definieron los siguientes requerimientos funcionales: autenticación de usuarios, sistema de reservas para el uso del laboratorio, configuración y control de sensores y motores, teleoperación de los robots, ejecución de ejercicios predefinidos y recolección de datos provenientes de los sensores.

Una vez definidos los requerimientos funcionales, se establecieron los requerimientos no funcionales con base en las necesidades derivadas de los primeros. Posteriormente, se dio paso a las etapas de diseño e implementación del laboratorio.

Se definió la arquitectura del sistema, la cual se basa en un PC principal que ejecuta el servidor web, desarrollado en Node.js con el framework Express, junto con la base de datos y el sistema operativo ROS. Los usuarios acceden al laboratorio a través de un navegador web, siempre que estén conectados a la misma red local.

La comunicación con el robot EV3 se realiza mediante ROS, donde el ROS Master se ejecuta en el servidor principal. Cuando el usuario interactúa con la interfaz web, sus acciones son procesadas por el servidor, el cual utiliza rosbridge para enviar los comandos correspondientes al robot. En cumplimiento del requerimiento de persistencia de datos, se diseñó el diagrama relacional de la base de datos y se definió una clase encargada de representar al robot en el área del servidor.

El desarrollo del laboratorio se organizó en módulos, cada uno de los cuales responde a los requerimientos funcionales definidos a partir del análisis de antecedentes.

Se desarrolló una dashboard que permite acceder a las diferentes sesiones del laboratorio, luego que el usuario ha creado una cuenta y ha accedido al laboratorio con sus credenciales.

Se desarrolló una sección de configuración, donde el usuario puede definir los sensores y motores a utilizar, puertos de conexión y modos de funcionamiento según sus necesidades.

También se desarrolló una sección de reservas, que permite reservar el uso del laboratorio en bloques de una hora.

Se desarrolló una sección de pruebas de sensores, en la que los usuarios pueden verificar el funcionamiento de los sensores según la configuración realizada y en la que se presentan los datos de los sensores de forma visual y su valor crudo.

Se desarrolló una sección dedicada a la teleoperación del robot que permite controlar el robot a distancia mediante el teclado, de acuerdo con la configuración establecida previamente por el usuario.

Además, se desarrolló una sección de ejercicios, en la cual los usuarios pueden realizar prácticas guiadas con actividades preestablecidas, o bien programar el robot mediante un entorno gráfico basado en bloques de programación.

Finalmente, se integró la recolección de datos provenientes de los sensores en todas las secciones prácticas del laboratorio, permitiendo su descarga para uso posterior.

Para evaluar la funcionalidad del laboratorio, se llevó a cabo un proceso de pruebas de integración dividido en dos etapas. La primera etapa consistió en pruebas realizadas por el autor del proyecto, con el objetivo de verificar el funcionamiento general del sistema, incluyendo el acceso al laboratorio y el correcto desempeño de cada una de sus secciones. Estas pruebas permitieron identificar y corregir problemas técnicos antes de su uso por parte de los estudiantes.

En la segunda etapa, se realizaron pruebas con estudiantes, esto con el fin de evaluar la experiencia de uso. Para ello, se diseñó una encuesta tipo Likert mediante Google Forms, orientada a medir el grado de acuerdo o desacuerdo de los participantes respecto a la funcionalidad de las distintas secciones del laboratorio. Los resultados mostraron una alta aceptación: en 16 de las 19 preguntas, el 100% de los encuestados expresó estar totalmente de acuerdo con las afirmaciones planteadas. Esto sugiere que las funcionalidades evaluadas cumplieron con las expectativas establecidas.

Por lo anterior, se concluye que los requerimientos funcionales sometidos a prueba fueron alcanzados satisfactoriamente. No obstante, también se identificaron algunas limitaciones asociadas al hardware utilizado. El kit EV3, al tratarse de una plataforma con varios años en el mercado, presenta restricciones como baja capacidad de procesamiento y limitada duración de batería, lo cual puede afectar su desempeño en sesiones prolongadas.

5.2 TRABAJOS FUTUROS

Para mejorar y complementar este proyecto, como trabajos futuros, se plantea los siguientes:

- **Soporte para múltiples robots:** El laboratorio fue diseñado originalmente para ser utilizado con un solo robot de forma simultánea. Como mejora, se podría añadir soporte para la configuración y gestión de múltiples robots, permitiendo al usuario seleccionar entre varias opciones disponibles.
- **Corrección de errores en el código fuente de los nodos del EV3:** El código original de los nodos del EV3 presenta errores relacionados con el sensor de color y el control de los motores, lo que afecta la estabilidad del uso del robot. En este trabajo, el desarrollo de nuevos nodos quedaba fuera del alcance de los objetivos planteados, ya que se pretendía utilizar directamente el código existente. No obstante, como posible mejora futura, sería recomendable corregir estos errores para lograr un funcionamiento más robusto y estable del robot.
- **Adición de otros periféricos:** Los nodos originales no ofrecen soporte para el uso de los altavoces ni de la pantalla del EV3. Por ello, la integración de estas funcionalidades constituye una mejora potencial que podría abordarse en trabajos futuros.
- **Migración de los nodos de ROS 1 a ROS 2:** Actualmente, los nodos desarrollados para el EV3 son compatibles únicamente con las versiones Indigo y Kinetic de ROS (ROS 1). Estas versiones dependen de distribuciones de Ubuntu ya obsoletas, lo cual obliga a utilizar sistemas operativos antiguos que pueden presentar incompatibilidades con hardware moderno. Además, ROS 1 ha dejado de recibir soporte oficial, lo que limita el acceso a actualizaciones y nuevas herramientas. Migrar los nodos a ROS 2 permitiría aprovechar las últimas características del ecosistema.

BIBLIOGRAFÍA

[1] R. A. A. Escobar, G. Y. A. Soto, E. P. B. Teran, M. E. C. Mesa, y L. M. C. Sandoval, «La Robótica Pedagógica como Herramienta para la Construcción de Aprendizajes Significativos en el Aula», VII Coloqui Internacional de Educacion, 2016.

[2] C. A. Jara, F. A. Candelas, S. T. Puente, and F. Torres, “Hands-on experiences of undergraduate students in Automatics and Robotics using a virtual and remote laboratory,” Comput Educ, vol. 57, no. 4, 2011, doi: 10.1016/j.compedu.2011.07.003.

[3] E. S. Tendero, R. C. Gutiérrez, and J. A. G. Somoza, 'Robótica en la enseñanza de conocimiento e interacción con el entorno. Una investigación formativa en Educación Infantil', Revista interuniversitaria de formación del profesorado, vol. 33, no. 94th ed., p. 11-28, 2019.

[4] S. J. Daniel, “Education and the COVID-19 pandemic,” Prospects (Paris), vol. 49, no. 1–2, 2020, doi: 10.1007/s11125-020-09464-3.

[5] M. Hassan, “Online teaching challenges during COVID-19 pandemic,” International Journal of Information and Education Technology, vol. 11, no. 1, 2020, doi: 10.18178/ijiet.2021.11.1.1487.

[6] A. S. Alves Gomes, J. F. Da Silva, and L. R. De Lima Teixeira, “Educational Robotics in Times of Pandemic: Challenges and Possibilities,” in 2020 Latin American Robotics Symposium, 2020 Brazilian Symposium on Robotics and 2020 Workshop on Robotics in Education, LARS-SBR-WRE 2020, 2020. doi: 10.1109/LARS/SBR/WRE51543.2020.930714

[7] R. De Abreu Alves E Silva, C. F. Joventino, J. H. M. Pereira, and L. P. Correa, “Teaching Robotics in Pandemic Times Through Remote Education,” in 2021 Latin American Robotics Symposium, 2021 Brazilian Symposium on Robotics, and 2021 Workshop on Robotics in Education, LARS-SBR-WRE 2021, 2021. doi: 10.1109/LARS/SBR/WRE54079.2021.9605454.

[8] J. P. A. Ioannidis, “The end of the COVID-19 pandemic,” European Journal of Clinical Investigation, vol. 52, no. 6. 2022. doi: 10.1111/eci.13782.

[9] H. Khong, I. Celik, T. T. T. Le, V. T. T. Lai, A. Nguyen, and H. Bui, “Examining teachers’ behavioural intention for online teaching after COVID-19 pandemic: A large-scale survey,” Educ Inf Technol (Dordr), vol. 28, no. 5, 2023, doi: 10.1007/s10639-022-11417-6.

[10] A. Z. Al Rawashdeh, E. Y. Mohammed, A. R. Al Arab, M. Alara, and B. Al-Rawashdeh, “Advantages and disadvantages of using E-learning in university education: Analyzing students’ perspectives,” Electronic Journal of e-Learning, vol. 19, no. 2, 2021, doi: 10.34190/ejel.19.3.2168.

[11] N. Davidovitch and E. Eckhaus, “ECONOMICS OF TIME: ADVANTAGES OF E-LEARNING IN PROPORTION TO THE TIME UTILIZED AND THE TRADEOFF BETWEEN WORK AND STUDIES,” Economics and Sociology, vol. 15, no. 2, 2022, doi: 10.14254/2071-789X.2022/15-2/14.

[12] D. Pang, S. Cui, and G. Yang, “Remote Laboratory as an Educational Tool in Robotics Experimental Course,” International Journal of Emerging Technologies in Learning, vol. 17, no. 21, pp. 230–245, 2022, doi: 10.3991/ijet.v17i21.33791.

[13] S. Michieletto, S. Ghidoni, E. Pagello, M. Moro, and E. Menegatti, “Why teach robotics using ROS,” Journal of Automation, Mobile Robotics and Intelligent Systems, 2014, doi: 10.14313/jamris_1-2014/8.

[14] O. Gervais and T. Patrosio, “Developing an Introduction to ROS and Gazebo Through the LEGO SPIKE Prime,” 2022. doi: 10.1007/978-3-030-82544-7_19.

[15] D. Roldán-Álvarez, S. Mahna, and J. M. Cañas, “A ROS-based Open Web Platform for Intelligent Robotics Education,” 2022. doi: 10.1007/978-3-030-82544-7_23.

[16] U. U. S. Rajapaksha, C. Jayawardena, and B. A. Macdonald, “Design, Implementation, and Performance Evaluation of a Web-Based Multiple Robot Control System,” Journal of Robotics, vol. 2022, 2022, doi: 10.1155/2022/9289625.

[17] B. Stefanuto, L. Piardi, A. O. Junior, M. Vallim, and P. Leitao, “Remote Lab of Robotic Manipulators through an Open Access ROS-based Platform,” in IEEE International Conference on Industrial Informatics (INDIN), Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/INDIN51400.2023.10218202.

[18] K. N. AlQarzaie and S. A. AlEnezi, “Using LEGO MINDSTORMS in Primary Schools: Perspective of Educational Sector,” International journal of online and biomedical engineering, vol. 18, no. 1, 2022, doi: 10.3991/ijoe.v18i01.27579.

[19] N. Rosillo, N. Montés, J. P. Alves, and N. M. F. Ferreira, “A generalized matlab/ROS/robotic platform framework for teaching robotics,” in Advances in Intelligent Systems and Computing, 2020. doi: 10.1007/978-3-030-26945-6_15.

[20] S. Schiffer and A. Ferrein, “Erika—early robotics introduction at kindergarten age,” Multimodal Technologies and Interaction, vol. 2, no. 4, 2018, doi: 10.3390/mti2040064.

[21] S. García, S. Rueda, B. Florián Gaviria, and B. Bacca Cortés, “Programmable mobile robots as hands-on platform for basic programming,” Sistemas y Telemática, vol. 15, no. 41, 2017, doi: 10.18046/syt.v15i41.2455.

[22] S. D. Jimenez Melo, «Desarrollo laboratorio virtual de robótica, para el manipulador 4R de la Universidad EIA». 2020. Accedido: 8 de noviembre de 2023. [En línea]. Disponible en: https://repository.eia.edu.co/handle/11190/2658

[23] Y. Rosas, «Diseño de una Plataforma Interoperable ROS - LEGO MINDSTORMS EV3», Arequipa, 2018.

[24] O. Sans-Cope, E. Danahy, D. Hannon, C. Rogers, J. Albo-Canals, y C. Angulo, «CORP. A Collaborative Online Robotics Platform», 2022. doi: 10.1007/978-3-030-82544-7_22.

[25] G. Farias, E. Fabregas, E. Peralta, H. Vargas, S. Dormido-Canto, and S. Dormido, "Development of an Easy-to-Use Multi-Agent Platform for Teaching Mobile Robotics," IEEE Access, vol. 7, 2019, doi: 10.1109/ACCESS.2019.2913916.

[26] J. Chacon, D. Goncalves, E. Besada, y J. A. López-Orozco, «Un laboratorio remoto de código abierto y bajo coste para el brazo robótico educativo Dobot Magician», *Revista Iberoamericana de Automática e Informática industrial*, vol. 20, n.º 2, 2022, doi: 10.4995/riai.2022.17477.

[27] D. Krūmiņš *et al.*, «Open Remote Web Lab for Learning Robotics and ROS With Physical and Simulated Robots in an Authentic Developer Environment», *IEEE Transactions on Learning Technologies*, vol. 17, pp. 1313-1326, 2024, doi: 10.1109/TLT.2024.3381858.

[28] A. Atahualpa Guerrero, «Desarrollo de un Laboratorio Virtual de Robots Utilizando un Motor Gráfico para Videojuegos», *Universitat Politècnica de València.*, oct. 2023, Accedido: 30 de abril de 2025. [En línea]. Disponible en: https://riunet.upv.es/handle/10251/198914

[29] S. Solak, Ö. Yakut, y E. Dogru Bolat, «Design and Implementation of Web-Based Virtual Mobile Robot Laboratory for Engineering Education», *Symmetry (Basel)*, vol. 12, n.o 6, 2020, doi: 10.3390/sym12060906.

[30] R. A. Maldonado Mercado, «Laboratorio de robótica Remoto-Presencial», 2022.

[31] «The Construct Robotics Institute», https://www.theconstruct.ai/.

[32] E. Tello-leal, T. Y. Guerrero-Melendez, y V. P. Saldivar-Alonso, «Revisión de la plataforma robótica LEGO Mindstorms para aplicaciones educativas y de investigación», *Revista S&T*, vol. 11, n.º 26, 2013.

[33] M. Quigley et al., "ROS: an open-source Robot Operating System," in ICRA workshop on open source software, 2009.

[34] J. García Sánchez y P. Jáuregui Arias, «Educación a distancia y mundos virtuales», *miradas (Pereira)*, vol. 1, n.º 2, 2019, doi: 10.22517/25393812.22051.

[35] Herrera D.C, «Importancia de los laboratorios remotos y virtuales en la educación superior Importance of Remote and Virtual Labs in Higher Education», *Documentos De Trabajo ECBTI*, vol. 1, n.º 1, 2020.

[36] Miguel Ángel, “Scrum: qué es y cómo funciona este marco de trabajo”, 2022. https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html (consultado el 7 de diciembre de 2023).

[37] D. Boada, “¿Qué es Docker? Funcionamiento y componentes básicos,” Tutoriales Hostinger, Feb. 20, 2025. https://www.hostinger.com/co/tutoriales/que-es-docker (consultado el 25 de mayo de 2025).

[38] C. Holl, «H4R_EV3 ROS nodes». https://hacks4ros.github.io/h4r_ev3_ctrl/index.html

6. ANEXOS

6.1 ESTRUCTURA DE CARPETAS DE LA CARPETA ANEXOS

En la figura 6.1 se presenta como están organizados los documentos y archivos pertenecientes al desarrollo del trabajo de grado. Se puede acceder a esta documentación a través del siguiente enlace:
https://drive.google.com/drive/folders/14a6_uYEXoHOImfe7sLIR5DGfT0j0Fb2b?usp=drive_link

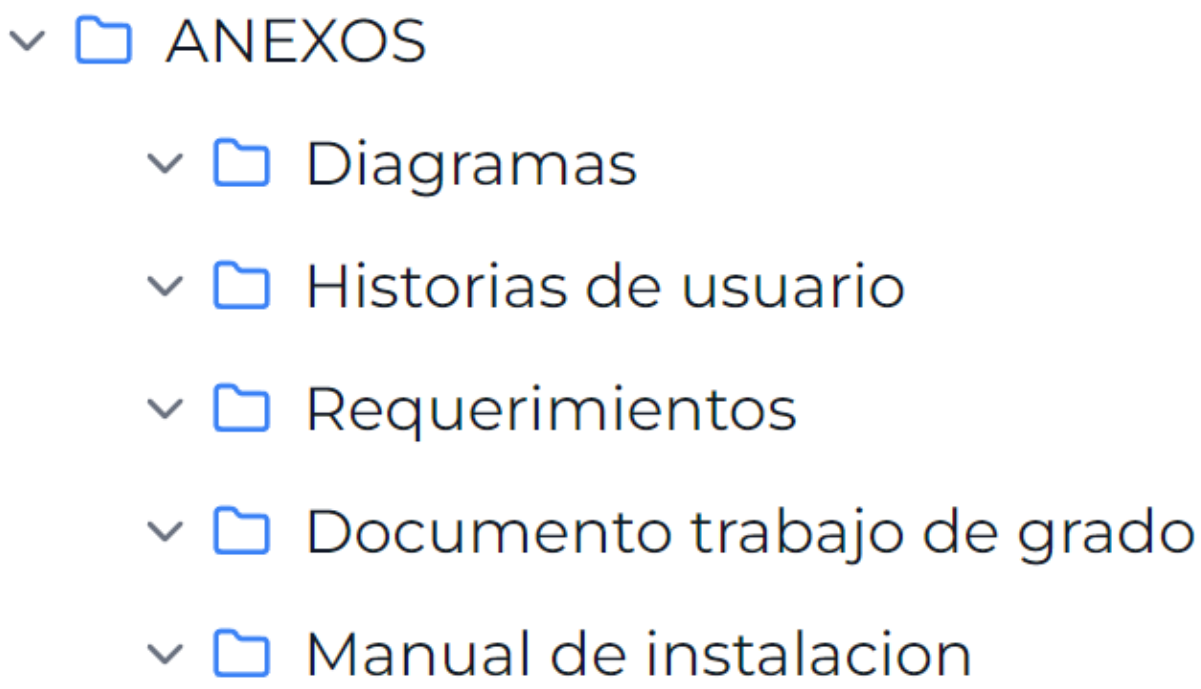


Figura 6.1. Estructura de la carpeta ANEXOS.

- **Diagramas**: Esta carpeta contiene imágenes correspondientes al diagrama de clases, diagrama relacional, diagrama de nodos y la estructura del laboratorio.
- **Historias de usuario:** Incluye un documento que describe las historias de usuario definidas para el proyecto.
- **Requerimientos:** Contiene un documento con las tablas de requerimientos funcionales y no funcionales del laboratorio.
- **Documento del trabajo de grado:** Esta carpeta contiene una copia del documento principal del trabajo de grado.

- **Manual de instalación:** Contiene una guía con instrucciones para la instalación del laboratorio y la preparación del kit LEGO Mindstorms EV3.

6.2 PLANIFICACIÓN DE SPRINTS

En esta sección se presenta la planificación del desarrollo de este trabajo de grado, basada en la metodología ágil Scrum.

6.2.1 SPRINT PLANNING

Objetivo: Definición de metas, tareas y tiempos de trabajo.

Actividades:

Definir las metas de los Sprints: Definición de lo que se quiere lograr en el Sprint.

Descomposición de tareas: Descomposición de las metas de los Sprints en pequeñas tareas a través de las historias de usuario.

Asignar la duración de los Sprints: Cada Sprint tendrá una duración estimada de entre 2 y 6 semanas.

Tabla 6.1 Sprints del proyecto.

Sprint	Descripción	Duración
Login y configuración	Desarrollo de la lógica de entrada al laboratorio, creación de usuarios y configuración del robot.	6 semanas
Reservación del laboratorio	Implementación del mecanismo de reserva del laboratorio	4 semanas
Sección de pruebas	Implementación de la sección destinada a la prueba de sensores.	6 semanas
Sección de tele-operacion	Desarrollo e integración de la funcionalidad de tele-operación, junto con la sección de la interfaz destinada a su uso.	4 semanas
Sección de ejercicios	Implementación de la sección que permite a los usuarios practicar con ejercicios.	6 semanas
Sección de ayuda	Desarrollo de la sección de ayuda destinada a orientar a los usuarios en el uso adecuado del laboratorio remoto.	2 semanas

6.2.2 EJECUCIÓN DEL PLAN DE TRABAJO

Objetivo: Ejecutar las tareas planificadas para el sprint actual y dar seguimiento al progreso de este.

Actividades:

- **Daily Scrum:** Reuniones breves cada semana para discutir el avance del sprint y los posibles obstáculos encontrados.
- **Ejecución de tareas:** Avance en las tareas asignadas para el Sprint.
- **Seguimiento del progreso:** Registro y monitoreo del progreso utilizando una herramienta de seguimiento de metodologías ágiles.

Tabla 6.2 Actividades de ejecución del plan de trabajo.

Actividad	Descripción	Duración	Fecha limite
Daily Scrum	Reuniones breves vía Google Meet	30 minutos	Cada semana
Ejecución de tareas	Desarrollo de tareas del Sprint	Dia completo	Diario
Seguimiento del progreso	Registro del avance de las tareas del sprint	No aplica	Final de la tarea

6.2.3 REVISIÓN Y RETROSPECTIVA

Objetivo: Evaluar el éxito del sprint e identificar posibles mejoras.

Actividades:

- **Sprint Review:** Revisión del trabajo realizado en el Sprint, junto con pruebas que demuestran su correcto funcionamiento.
- **Sprint Retrospective**: Evaluación del producto resultante del Sprint con el propósito de detectar posibles áreas de mejora en el producto.

Tabla 6.3. Actividades de revisión.

Actividad	**Descripción**	**Duración**	**Fecha limite**
Sprint review	Evaluar el trabajo completo	30 minutos	Final del Sprint
Sprint Retrospective	Discutir mejoras	30 minutos	Final del Sprint