Universidad Nacional de San Luis

Facultad de Ingeniería y Ciencias Agropecuarias

DESARROLLO DEL CONTROLADOR E INTERFAZ DE UN ROBOT COLABORATIVO DE 6 GRADOS DE LIBERTAD

Autor: Ianchina Diaz Diego Sebastian

Trabajo final de Ingeniería Mecatrónica

Director: Ing. Gabriel Iglesias

Codirector: Ing. Oscar Daniel Moran

Villa Mercedes, San Luis

Año 2026

DERECHO DE AUTOR

DEDICATORIA

Este trabajo está dedicado a mis padres, por su apoyo incondicional a lo largo de todos estos años, por el esfuerzo constante que realizaron para brindarme las oportunidades necesarias y por ser un pilar fundamental en cada etapa de mi formación. Su acompañamiento, dedicación y confianza han sido claves para alcanzar este objetivo.

Asimismo, a mi pareja, por su paciencia, comprensión y apoyo durante el desarrollo de este proyecto, y por su acompañamiento en los últimos años, especialmente en los momentos de mayor exigencia.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mis padres por su apoyo constante a lo largo de toda mi formación, y a mi pareja por su acompañamiento y comprensión durante estos últimos años.

Asimismo, agradezco al Laboratorio de Mecatrónica (LABME) por brindarme, durante estos años, no solo los recursos necesarios, sino también un espacio de aprendizaje y desarrollo que resultó clave en mi formación profesional.

A los Ingenieros Gabriel Iglesias y Daniel Morán, director y codirector de este trabajo respectivamente, por su orientación, acompañamiento y aportes durante el desarrollo del proyecto.

Quiero hacer un agradecimiento especial al técnico del laboratorio, Elio Ogas, por su acompañamiento constante en todos los proyectos desarrollados dentro del LABME. Su predisposición, experiencia y dedicación fueron fundamentales no solo durante este trabajo final, sino también a lo largo de las prácticas previas, contribuyendo de manera significativa a mi formación.

Asimismo, agradezco al Ing. Matías Funes por su apoyo en las etapas iniciales de mi formación dentro del laboratorio.

Finalmente, agradezco a Javier Bravin por su ayuda y recomendaciones en distintas cuestiones técnicas, que resultaron de gran utilidad durante el desarrollo del proyecto.

Agradezco a la Universidad Nacional de San Luis, y en particular a la Facultad de Ingeniería y Ciencias Agropecuarias, por la formación académica brindada a lo largo de la carrera. Asimismo, reconozco el compromiso de los docentes que, mediante su dedicación y profesionalismo, contribuyeron al desarrollo de los conocimientos que hicieron posible la realización de este trabajo.

RESUMEN

El presente trabajo final de grado tiene como propósito desarrollar un sistema completo que permita manejar y programar un brazo robótico de manera sencilla, pensado para el uso didáctico. Para ello, se desarrolló un controlador y una interfaz capaces de operar un robot colaborativo, es decir, un tipo de robot que puede ser guiado directamente con la mano por el usuario para indicarle los movimientos que se desean realizar. Con el fin de evaluar el funcionamiento del sistema desarrollado, se ensambló un robot utilizando un diseño estructural proporcionado por el laboratorio universitario donde se llevó a cabo este proyecto. Tanto el robot como el controlador fueron concebidos como herramientas educativas para el Laboratorio de Mecatrónica de la Facultad de Ingeniería y Ciencias Agropecuarias, con el objetivo de ofrecer a los estudiantes un equipo accesible que facilite la comprensión práctica de los conceptos básicos de control y movimiento de robots. El sistema creado permite trabajar de dos maneras principales. Por un lado, el usuario puede indicar mediante la interfaz a qué posición desea que el robot se desplace, ya sea cargando posiciones previamente guardadas o ingresando nuevas, lo que posibilita planificar tareas paso a paso. Por otro lado, el robot también puede ser movido directamente con la mano, permitiendo que el propio usuario le muestre físicamente el recorrido o las posiciones que desea repetir. El sistema registra estos movimientos y puede reproducirlos posteriormente, lo que lo convierte en una herramienta muy útil para la enseñanza, ya que permite experimentar de forma directa con diferentes formas de programar un robot. La interfaz desarrollada, basada en una página web accesible desde una computadora, hace posible que cualquier usuario pueda manejar el robot sin necesidad de conocimientos previos, ofreciendo una experiencia clara y ordenada. Finalmente, se realizaron diversas pruebas para comprobar el correcto funcionamiento tanto del controlador como del robot y la interfaz. Los resultados obtenidos mostraron que el sistema cumple con los objetivos planteados, permitiendo controlar el robot de manera fluida y registrar distintos tipos de movimientos para su posterior uso. Además, el proyecto demostró su utilidad como herramienta didáctica, ya que facilita que los estudiantes puedan interactuar con el robot, comprender su comportamiento y experimentar con diferentes formas de programar trayectorias. El sistema desarrollado sienta las bases para futuras mejoras y ampliaciones, permitiendo que nuevas generaciones de alumnos continúen explorando y aprendiendo a partir de un recurso práctico y adaptable a distintos niveles de formación.

Palabras claves — Arduino, Cobot, Dynamixel, Interfaz Grafica, Robot antropomórfico, ROS, TinyDB.

.ÍNDICE DE CONTENIDO

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO 1: Propuesta

1.1 Introducción

La robótica industrial y colaborativa ha experimentado un crecimiento significativo en los últimos años, impulsada por la necesidad de sistemas flexibles, seguros y capaces de interactuar de manera directa con las personas. En el ámbito académico, esta evolución ha generado un creciente interés por incorporar robots reales como herramientas de enseñanza, permitiendo a los estudiantes aplicar de forma práctica los conocimientos teóricos adquiridos en áreas como control, programación, electrónica y automatización.

En este contexto, los robots colaborativos representan una alternativa especialmente adecuada para el entorno educativo, ya que permiten modos de interacción más intuitivos, como la guía manual del robot, y reducen los riesgos asociados a su operación. Sin embargo, el acceso a este tipo de equipamiento suele estar limitado por el costo de los sistemas comerciales, lo que dificulta su utilización como recurso didáctico en laboratorios de universidades públicas.

El presente trabajo final de grado surge como respuesta a esta problemática y tiene como propósito el diseño y desarrollo de un sistema de control e interfaz gráfica para un robot colaborativo, orientado específicamente a su uso educativo. El proyecto fue concebido para el Laboratorio de Mecatrónica de la Facultad de Ingeniería y Ciencias Agropecuarias, con el objetivo de contar con una plataforma accesible que permita a los estudiantes experimentar con un robot real, comprender su funcionamiento y explorar distintas formas de programar y controlar sus movimientos.

Para alcanzar este objetivo, se desarrolló un controlador basado en el Robot Operating System (ROS), encargado de coordinar la comunicación entre los distintos componentes del sistema, realizar los cálculos necesarios y gestionar la ejecución de movimientos y rutinas. El sistema se complementa con un microcontrolador que controla directamente los actuadores del robot, una base de datos destinada al almacenamiento de puntos y trayectorias, y una interfaz gráfica basada en una plataforma web que facilita la interacción del usuario con el sistema de manera clara e intuitiva.

Asimismo, el robot desarrollado permite operar en dos modos principales: un modo de movimiento, en el cual el usuario define posiciones y rutinas a ejecutar, y un modo de lectura,

que posibilita guiar manualmente el robot para registrar posiciones o trayectorias completas. Esta dualidad de funcionamiento busca reforzar el aprendizaje práctico, permitiendo comparar distintas metodologías de programación de robots.

El informe se organiza de la siguiente manera: en primer lugar, se presenta el marco teórico, donde se describen los conceptos fundamentales relacionados con la robótica, ROS y las tecnologías empleadas. A continuación, se detalla el diseño y la arquitectura del sistema desarrollado, incluyendo el hardware, el software y la interfaz gráfica. Posteriormente, se describen las pruebas realizadas y los resultados obtenidos. Finalmente, se exponen las conclusiones del trabajo y se proponen posibles líneas de mejora y ampliación del sistema.

1.2 Objetivos

1.2.1 Objetivo general

- Diseñar y construir el controlador e interfaz de un robot colaborativo (COBOT) antropomórfico impreso en 3D, con capacidad de operación en dos modos distintos: "movimiento" y "lectura", explorando su aplicabilidad en entornos de fabricación avanzados y su potencial educativo en el ámbito universitario.

1.2.2 Objetivos específicos

- Definir y seleccionar los componentes electrónicos y de control necesarios para el funcionamiento del sistema, incluyendo actuadores, sensores, microcontroladores y elementos de comunicación.

- Integrar el sistema de control con la estructura del robot disponible en el LABME, realizando el montaje del hardware indispensable para su operación y pruebas

- Desarrollar la arquitectura de software del controlador, asegurando la comunicación entre los distintos dispositivos, el procesamiento de datos y la respuesta adecuada del sistema

- Implementar los modos de operación “movimiento” y “lectura”, creando los procedimientos necesarios para registrar posiciones, reproducir trayectorias y ejecutar movimientos indicados por el usuario.

- Seleccionar e implementar la base de datos adecuada para el proyecto, integrándola al sistema y desarrollando el módulo encargado de gestionar las funciones de consulta, registro y modificación de la información.

- Diseñar y programar una interfaz web que permita interactuar de forma sencilla con el robot, contemplando funciones de programación, control y visualización.

- Ensamblar y ajustar el robot utilizado como plataforma de pruebas, garantizando su correcto funcionamiento mecánico y electrónico.

- Realizar pruebas individuales de cada módulo y pruebas de integración del sistema completo, identificando mejoras necesarias y optimizando el rendimiento general.
- Evaluar el desempeño del sistema desarrollado en escenarios representativos del uso didáctico, analizando su facilidad de uso, exactitud, seguridad y capacidad para apoyar actividades de enseñanza.

1.3 Alcances y limitaciones

El proyecto abarca el diseño y desarrollo del controlador de un robot colaborativo de seis grados de libertad, así como el montaje y puesta en funcionamiento del robot utilizado como plataforma de validación. Incluye además la implementación de una interfaz de usuario basada en una plataforma web que permite programar, registrar y reproducir movimientos y trayectorias. El alcance cubre la integración del hardware necesario, el desarrollo del software de control y la realización de pruebas funcionales que aseguren el correcto desempeño del sistema.

La aplicación del sistema se encuentra limitada exclusivamente al ámbito didáctico, académico y de experimentación dentro del laboratorio universitario. Debido a los componentes empleados y a los objetivos del proyecto, el robot y su controlador no están destinados a cumplir con los requisitos de precisión, robustez, seguridad o capacidad de carga propios de aplicaciones industriales.

1.4 Marco teórico

1.4.1 Cobots

Los robots colaborativos (cobots) (Figura Nº1) son manipuladores diseñados explícitamente para compartir espacio y tareas con operadores humanos, priorizando la seguridad y la facilidad de interacción por sobre las exigencias de producción masiva de los robots industriales tradicionales. Estos sistemas se caracterizan por una arquitectura mecánica y de control que reduce la masa y el momento de inercia en los eslabones, incorpora limitación de potencia y fuerza (power & force limiting) y emplea sensores y estrategias de detección de colisiones para minimizar el riesgo en contactos físicos con personas. Estos rasgos permiten que los cobots ofrezcan una mayor flexibilidad operativa y una integración más simple en entornos de trabajo compartidos, al tiempo que facilitan reconfiguraciones rápidas y su uso en contextos de enseñanza y prototipado [1].

Desde la perspectiva dinámica, la reducción de la inercia y la optimización del reparto de masas en la cadena cinemática influyen directamente en la respuesta dinámica del manipulador: brazos con menor inercia producen menores pares requeridos para aceleraciones equivalentes, lo que reduce las demandas sobre los actuadores y mejora la capacidad del control para obtener movimientos suaves a bajas velocidades. En aplicaciones colaborativas, esto es crítico porque la detección de interacción física y la regulación de fuerza dependen de la relación entre la dinámica propia del robot y el ancho de banda de sus controladores; por ello, el diseño mecánico (longitud de eslabones, ubicación de masas, elección de actuadores) y la estrategia de control inmediato (control de posición, control de esfuerzo o control híbrido) deben concebirse de forma complementaria [2].

Una funcionalidad central de muchos cobots es la programación por demostración o “hand-guiding”, mediante la cual un operador guía físicamente el efector final para marcar posiciones o trayectorias que el sistema registra y reproduce posteriormente. Esta técnica reduce drásticamente la barrera de entrada para la programación de tareas, puesto que el usuario no necesita formular trayectorias ni parámetros de control de forma explícita: el robot capta las poses o los puntos intermedios (via-points) y emplea algoritmos de interpolación y planificación para generar trayectorias ejecutables. Investigaciones recientes han avanzado en métodos que mejoran la precisión del hand-guiding —por ejemplo, compensando peso e inercia del efector o estimando fuerzas de guía—, así como en estrategias que combinan hand-guiding

con interfaces no-codificadas (no-code) y aprendizaje asistido para ampliar la usabilidad en entornos industriales y educativos [3].

Figura Nº1 Cobot UR

Fuente: tomado de [4]

1.4.2 Robot Operating System (ROS)

Robot Operating System (ROS) es un meta-sistema de software de código abierto ampliamente utilizado en robótica para el desarrollo de aplicaciones modulares, escalables y reutilizables. A pesar de su nombre, ROS no es un sistema operativo en el sentido tradicional, sino un middleware que provee servicios esenciales tales como abstracción de hardware, comunicación entre procesos, gestión de paquetes, visualización y herramientas de desarrollo [5] [6].

La arquitectura de ROS se basa en un enfoque distribuido, donde múltiples procesos cooperan para ejecutar tareas complejas. Esta característica resulta especialmente adecuada para sistemas robóticos, en los que sensores, actuadores, algoritmos de control y planificación deben interactuar de manera concurrente y desacoplada [5] [6].

1.4.3 Encoders

Los encoders son sensores de posición angular o lineal que proporcionan la información necesaria para el control y la estimación del estado en manipuladores y sistemas mecatrónicos. Existen dos clasificaciones fundamentales: encoders incrementales, que generan impulsos

relativos al movimiento y requieren homing para recuperar la referencia tras un corte de alimentación; y encoders absolutos, que entregan una única palabra digital que identifica la posición sin necesidad de referencia tras un reinicio. La elección entre uno u otro depende de requisitos de aplicación como la necesidad de conocer la posición tras pérdida de energía, la resolución demandada y la complejidad de la electrónica de lectura [8].

Desde el punto de vista de la tecnología de lectura, los encoders pueden ser ópticos, magnéticos o eléctricos (resolvers), entre otros. Los encoders ópticos ofrecen, en general, mayor resolución y precisión, pero son más sensibles a condiciones ambientales adversas (polvo, humedad, vibraciones); los encoders magnéticos y los resolvers, en cambio, presentan mayor robustez para entornos industriales rigurosos y mayor tolerancia a contaminantes y variaciones de temperatura, aunque a veces con menor resolución nominal. Por ello, la selección de la tecnología ha de ponderar precisión frente a robustez operativa según el entorno y el presupuesto [8].

En aplicaciones robóticas la información del encoder se integra en lazo cerrado de control (control de posición y velocidad) y en algoritmos de planificación y seguridad. Consideraciones prácticas relevantes incluyen la resolución (puntos por revolución o bits), la linealidad / precisión absoluta, la tasa máxima de muestreo, el retardo de comunicación, la inmunidad a ruido y la capacidad de operar después de un corte de energía (encoders absolutos). En robots colaborativos y plataformas didácticas suele preferirse una combinación que ofrezca suficiente resolución para control y suavidad de trayectorias con redundancia o filtrado que mejore robustez ante lecturas erráticas [8].

1.4.4 Base de datos

Las bases de datos son sistemas de almacenamiento de información estructurada que permiten guardar, recuperar y manipular datos de forma eficiente. Tradicionalmente, los sistemas de gestión de bases de datos relacionales (RDBMS) han sido dominantes y se caracterizan por organizar los datos en tablas con filas y columnas interrelacionadas mediante claves, proporcionando consistencia y capacidades transaccionales estrictas bajo las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Este modelo es especialmente apropiado cuando se requiere garantizar integridad referencial y realizar consultas complejas mediante un lenguaje estándar de consulta estructurado, como SQL [9].

En contraste, las bases de datos no relacionales, comúnmente denominadas NoSQL, surgen como un paradigma alternativo donde los datos no necesariamente se estructuran en tablas ni se accede mediante SQL como lenguaje primario de consultas. En lugar de ello, estos sistemas adoptan modelos de almacenamiento más flexibles —como documento, clave-valor, columna o gráfico— que permiten manejar datos semiestructurados o no estructurados, y facilitan la escalabilidad horizontal en aplicaciones modernas con grandes volúmenes de datos y patrones de acceso variables [9].

La elección entre bases de datos relacionales y no relacionales depende de los requerimientos del proyecto. Las bases de datos relacionales suelen ser preferidas cuando los datos tienen una estructura rígida y relaciones fuertes entre entidades, y se requiere mantener una alta coherencia y transacciones complejas. En cambio, las bases de datos no relacionales son recomendadas cuando se trabaja con esquemas flexibles o cambiantes, se desea una alta escalabilidad horizontal y se tolera un modelo de consistencia más laxa en favor del rendimiento y la adaptabilidad [9].

En el contexto de sistemas robóticos, incluidos los robots colaborativos didácticos, generalmente no se enfrentan grandes requerimientos de transacciones complejas ni de integridad referencial estricta como en sistemas bancarios o empresariales. Por ello, cuando la base de datos se utiliza para almacenar trayectorias, posiciones guardadas o rutinas de movimiento, un modelo de datos no relacional y ligero puede ser más eficiente y flexible que un RDBMS tradicional. Esto es especialmente cierto en proyectos educativos o prototipos donde los datos a almacenar no siguen un esquema estrictamente fijo y donde la simplicidad de implementación es una ventaja [9].

1.5 Estado del arte

El desarrollo de robots colaborativos y de los sistemas de control asociados ha experimentado un crecimiento sostenido en los últimos años, impulsado por la necesidad de soluciones flexibles, seguras y reconfigurables tanto en entornos industriales como educativos. En este contexto, el avance de tecnologías de software y hardware ha permitido la implementación de sistemas robóticos cada vez más accesibles, modulares y escalables.

En el ámbito del software, el uso de plataformas como ROS se ha consolidado como un estándar de facto en investigación y desarrollo en robótica, debido a su arquitectura distribuida

basada en nodos y su capacidad de integración con múltiples dispositivos y lenguajes de programación. Esta característica permite desacoplar los distintos componentes del sistema, facilitando su mantenimiento, reutilización y escalabilidad.

Sobre esta plataforma, herramientas específicas como MoveIt proporcionan soluciones avanzadas para la planificación y ejecución de trayectorias en manipuladores robóticos. Este tipo de frameworks integra algoritmos de cinemática, planificación libre de colisiones y control de movimiento, reduciendo significativamente la complejidad asociada al desarrollo de aplicaciones robóticas reales.

En relación con las interfaces de usuario, las tecnologías web modernas han comenzado a incorporarse en aplicaciones de control y monitoreo de sistemas robóticos, particularmente en entornos educativos y de prototipado. En este sentido, React permite desarrollar interfaces dinámicas basadas en componentes, accesibles desde distintos dispositivos y con una elevada capacidad de interacción, favoreciendo la usabilidad del sistema.

Desde el punto de vista del hardware, los actuadores inteligentes como los motores Dynamixel han ganado relevancia en aplicaciones robóticas experimentales y educativas, debido a su capacidad de integrar en un único dispositivo el accionamiento, el sensado de posición y la comunicación. Esta característica simplifica el diseño de sistemas de múltiples grados de libertad y facilita su integración con plataformas de control de mayor nivel. Asimismo, el uso de microcontroladores como Arduino continúa siendo una solución ampliamente adoptada en etapas de prototipado, debido a su simplicidad, bajo costo y versatilidad en la interacción con dispositivos físicos.

Por otra parte, en aplicaciones donde el almacenamiento de datos no requiere estructuras complejas ni altos niveles de concurrencia, las bases de datos no relacionales representan una alternativa eficiente. En este contexto, soluciones como TinyDB permiten gestionar información en formato ligero, resultando adecuadas para sistemas de pequeña escala y aplicaciones didácticas, donde se prioriza la simplicidad de implementación y la flexibilidad en el manejo de datos.

1.5.1 Ecosistema ROS

Sobre la base conceptual presentada en el marco teórico, en esta sección se abordan las principales herramientas y tecnologías utilizadas actualmente en la implementación de sistemas robóticos, profundizando en su aplicación práctica dentro del ecosistema ROS

1.5.1.1 Nodos, tópicos y su relación

1.5.1.1.1 Nodos

Un nodo en ROS es un proceso ejecutable que realiza una tarea específica dentro del sistema, como la lectura de sensores, el control de actuadores o la planificación de movimientos [5] [6]. Cada nodo se diseña típicamente para cumplir una función bien definida, siguiendo el principio de modularidad.

Esta separación funcional facilita el mantenimiento, la depuración y la reutilización de código, además de permitir que nodos desarrollados en diferentes lenguajes de programación (principalmente C++ y Python) coexistan dentro del mismo sistema [5] [6].

1.5.1.1.2 Tópicos

Los tópicos constituyen el principal mecanismo de comunicación asíncrona en ROS. Un tópico es un canal nombrado a través del cual los nodos intercambian información siguiendo el modelo publicador–suscriptor.

- Un nodo publicador envía mensajes a un tópico determinado.
- Un nodo suscriptor recibe los mensajes publicados en ese tópico.

Este modelo desacopla completamente a los nodos entre sí, ya que un publicador no necesita conocer qué nodos están suscritos, ni un suscriptor necesita conocer el origen de los datos. Esto aumenta la flexibilidad del sistema y permite modificar o agregar nodos sin afectar al resto de la arquitectura [5] [6].

1.5.1.1.3 Relación entre nodos y tópicos

En una aplicación robótica típica, cada nodo puede publicar información en uno o varios tópicos y, a su vez, suscribirse a otros. Por ejemplo, un nodo de control puede suscribirse a un

tópico que contiene estados articulares (es decir, la posición de cada articulación del robot a donde quiero que se mueva) y publicar comandos de movimiento en otro.

Esta interconexión de nodos a través de tópicos conforma el grafo de comunicación del sistema ROS, el cual puede observarse y depurarse mediante herramientas gráficas como rqt_graph (Figura Nº2) [5] [6].

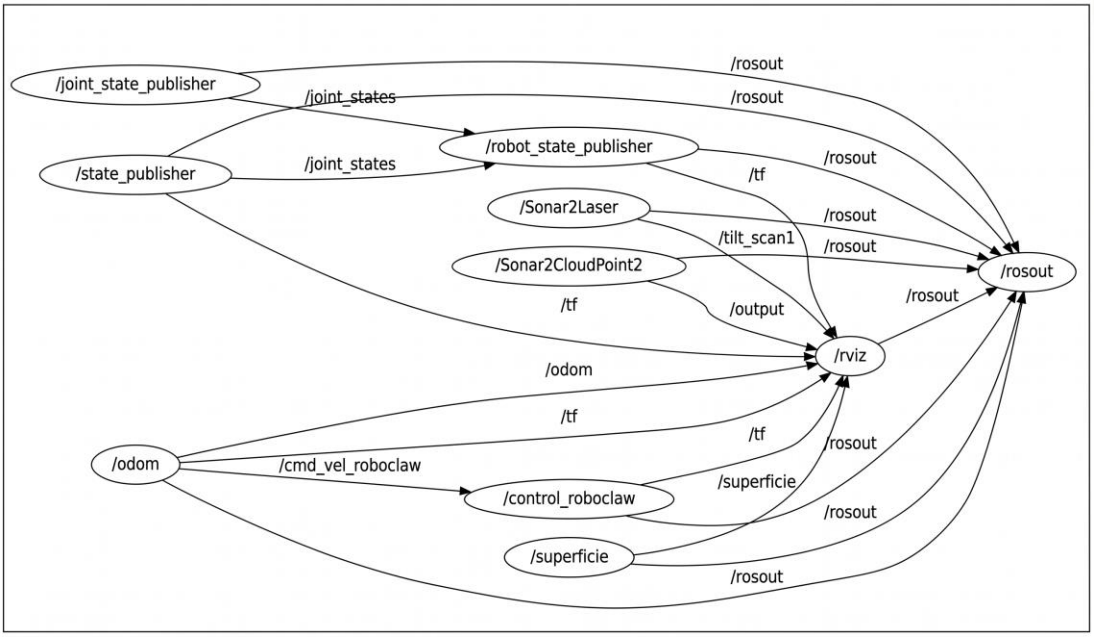

Figura Nº2 Grafico rxgraph de nodos y tópicos.

Fuente: tomado de [6]

1.5.1.2. Mensajes y tipos de mensajes

1.5.1.2.1 Mensajes

La información intercambiada a través de los tópicos se encapsula en estructuras de datos denominadas mensajes. Un mensaje define el formato de los datos transmitidos y garantiza que tanto publicadores como suscriptores interpreten la información de forma consistente [5] [6].

1.5.1.2.2 Tipos de mensajes

ROS provee una gran cantidad de tipos de mensajes estándar, organizados en paquetes, tales como std_msgs, sensor_msgs, geometry_msgs, trajectory_msgs y control_msgs, ampliamente utilizados en aplicaciones de control y planificación de movimiento.

Además de los mensajes estándar, el desarrollador puede definir mensajes personalizados cuando los tipos existentes no se ajustan a las necesidades del sistema. Esta extensibilidad permite adaptar ROS a una amplia variedad de aplicaciones específicas [5] [6].

1.5.1.3 MoveIt: planificación y control de movimiento

MoveIt es un framework dentro del ecosistema ROS para la planificación, ejecución y visualización de movimientos de robots, especialmente manipuladores industriales y colaborativos [10].

MoveIt integra múltiples componentes que permiten el cálculo de cinemática directa e inversa, la planificación de trayectorias considerando colisiones y restricciones, la gestión de un planning scene y la ejecución de movimientos en simulación o en hardware real [10].

El framework se apoya en bibliotecas externas como OMPL y ofrece una interfaz unificada que simplifica el desarrollo de aplicaciones de movimiento complejas [10].

1.5.1.3.1 Move Group y MoveIt Commander

El nodo move_group actúa como el núcleo de planificación y ejecución de MoveIt. Interfaces de alto nivel [11][12], como moveit_commander en Python, permiten definir objetivos de movimiento sin necesidad de implementar directamente los algoritmos de planificación subyacentes [13].

1.5.1.4 Planificador Pilz Industrial Motion Planner

El Pilz Industrial Motion Planner es un planificador integrado a MoveIt, orientado a aplicaciones industriales. Este planificador permite la generación de trayectorias PTP, LIN y CIRC, con perfiles de velocidad y aceleración bien definidos, lo que lo hace adecuado para entornos de automatización industrial y robótica colaborativa [14].

1.5.1.5 URDF y XACRO

El Unified Robot Description Format (URDF) es el estándar utilizado en ROS para describir la estructura física de un robot, incluyendo su geometría, articulaciones, jerarquía de enlaces y propiedades dinámicas básicas. Esta representación es fundamental para herramientas de simulación, visualización y planificación de movimiento, como RViz y MoveIt, las cuales requieren un modelo estructurado del robot para operar correctamente. El URDF se

basa en XML y permite especificar mallas, límites articulares y parámetros cinemáticos esenciales para la interacción del robot con su entorno [15][16].

En aplicaciones robóticas reales, los modelos definidos en URDF suelen volverse extensos y difíciles de mantener, por lo que ROS incorpora XACRO, un lenguaje de macros que permite generar URDF de manera modular y reutilizable. Con XACRO es posible emplear parámetros, bloques reutilizables y expresiones condicionales para simplificar la definición de robots complejos, reduciendo la duplicación de código y facilitando la adaptación del modelo ante cambios en el diseño mecánico. Finalmente, los archivos .xacro se procesan para producir un URDF expandido, compatible con todas las herramientas del ecosistema ROS (Figura Nº3) [17][18].

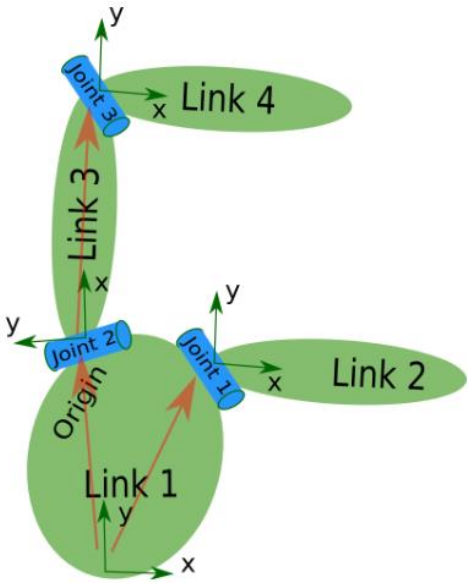

Figura Nº3 Representación de archivo .xacro.

Fuente: tomado de [19]

1.5.1.6 Rosserial y arduino

Para integrar la placa de desarrollo Arduino dentro del sistema basado en ROS se utiliza la librería rosserial_arduino. Esta herramienta permite establecer la comunicación entre ROS y Arduino a través de un enlace serie, haciendo posible que la placa Arduino sea tratado, desde el punto de vista del sistema, como un nodo más de ROS. De esta manera, Arduino puede enviar y recibir información utilizando mensajes de ROS, facilitando la interacción con el resto de la arquitectura de control [20].

1.5.2 Interfaces de usuario basadas en web

Si bien las interfaces tradicionales en robótica suelen basarse en aplicaciones de escritorio específicas, el uso de tecnologías web permite mejorar la accesibilidad, portabilidad y desacoplamiento del sistema, especialmente en entornos educativos y de prototipado.

1.5.2.1 React

React.js es una biblioteca de JavaScript de código abierto desarrollada por Facebook para la construcción de interfaces de usuario dinámicas y altamente interactivas. A diferencia del enfoque tradicional de desarrollo web, en el que HTML, CSS y JavaScript están separados y el navegador realiza recargas completas de páginas, React propone una renderización en el lado del cliente que actualiza el contenido de forma reactiva sin recargar toda la página (Single Page Application, SPA) . Esto se logra mediante el uso de un Virtual DOM, un modelo ligero del Document Object Model que permite minimizar las actualizaciones directas al DOM real, reduciendo costos de rendimiento asociados a manipulaciones frecuentes de interfaz [21].

La arquitectura de React se basa en un desarrollo orientado a componentes, donde cada componente encapsula su propio estado y lógica de presentación, favoreciendo la reutilización y modularidad del código. Esta estructura declarativa facilita la lectura y mantenimiento del código, ya que el desarrollador describe lo que la interfaz debe renderizar según el estado de la aplicación, y React se encarga de actualizar eficientemente la vista ante cambios de datos. Estas características hacen de React una alternativa moderna frente al desarrollo convencional con HTML, CSS y JavaScript puros, especialmente cuando se busca construir aplicaciones web dinámicas y responsivas con una experiencia de usuario fluida y consistente [21].

1.5.3 Dynamixels

Los actuadores Dynamixel utilizados en este proyecto son motores inteligentes de la serie XL de Robotis (Figura Nº4), integrados como módulos completos que combinan un motor de corriente continua, reducción mecánica, controlador interno y sistema de comunicación en un solo componente compacto. Estos actuadores están diseñados específicamente para aplicaciones robóticas donde se requiere control de posición, velocidad y torque con retroalimentación de alta resolución, integrándose fácilmente en arquitecturas de control basadas en ROS u otros controladores embebidos [23].

Figura Nº4 Motor Dynamixel XL430-W250.

Fuente: tomado de [24]

Para controlar los actuadores DYNAMIXEL, el controlador principal necesita convertir sus señales UART al tipo semidúplex. El diagrama de circuito recomendado para esto se muestra a continuación (Figura Nº5) [24].

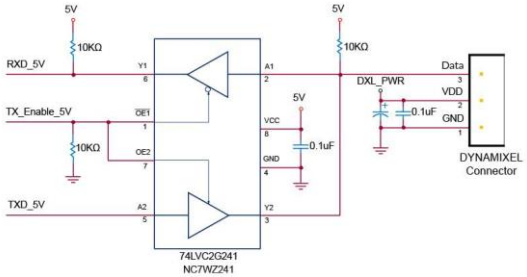

Figura Nº5 Circuito para convertir señales UART al tipo semidúplex.

Fuente: tomado de [24]

Tabla Nº1. Características relevantes al proyecto de motores Dynamixels modelos XL430-W250 y XL320

Fuente: tomado de [24] [25]

Características	**XL430-W250**	**XL320**
Sensor de posición	Codificador absoluto sin contacto (12 bits, 360 [°])	10 bits ,300º
Resolución	4096 [pulso/rev]- 0,088 [grados/pulso]	0.29 [grados/pulso]
Peso	57,2 [g]	16.7 [g]
Torque máximo	-1,0 [Nm] (a 9,0 [V], 1,0 [A], 1.000 [Nm/A]) -1,4 [Nm] (a 11,1 [V], 1,3 [A], 1,077 [Nm/A]) -1,5 [Nm] (a 12,0 [V], 1,4 [A], 1,071 [Nm/A])	0.39 [N.m] (at 7.4 [V], 1.1 [A])
Voltaje de entrada	6,5 ~ 12,0 [V] (**Recomendado: 11.1 [V]**)	6 ~ 8.4 [V] (Recommended : 7.4 [V])
Conexión física	Bus multidrop TTL (lógica de 5 V) Comunicación en serie asincrónica semidúplex TTL	
Velocidad sin carga	-47 [rev/min] (a 9,0 [V]) **-57 [rev/min] (a 11,1 [V])** -61 [rev/min] (a 12,0 [V])	114 [rev/min] (at 7.4 [V], 0.18 [A])
Velocidad		0 ~ 1.023 – (0,111 rpm [rev/min])
Tiempo de retardo de retorno	0 ~ 254 - 2 [µseg]/unidad – valor inicial 250 (500[µseg])	

1.5.3.1 Dynamixel2Arduino

La librería Dynamixel2Arduino es una biblioteca de software de código abierto diseñada para facilitar la comunicación y el control de actuadores DYNAMIXEL desde placas compatibles con Arduino. Esta librería implementa el protocolo de comunicación DYNAMIXEL, que permite enviar y recibir comandos y datos con servomotores inteligentes fabricados por ROBOTIS. Su propósito principal es abstraer la complejidad de la comunicación serial bidireccional con los actuadores, proporcionando funciones que permiten inicializar los servomotores, configurar parámetros, solicitar estados y ejecutar movimientos con relativa simplicidad desde un microcontrolador.

Esta librería está disponible en GitHub y es compatible con la mayoría de las arquitecturas y placas soportadas por el entorno de desarrollo de Arduino. Se utiliza ampliamente cuando se desea integrar servomotores DYNAMIXEL con controladores basados en Arduino, tanto en aplicaciones educativas como en proyectos de robótica experimental y de investigación [26].

1.5.4 Almacenamiento de datos: TinyDB

Una de las soluciones no relacionales más apropiadas para aplicaciones de baja escala es TinyDB, una base de datos documental escrita en Python que almacena datos en formato JSON y no requiere un servidor separado ni dependencias externas. TinyDB es un ejemplo de base de datos NoSQL embebida, que resulta adecuada para proyectos pequeños, prototipos o aplicaciones sin demandas de alto rendimiento o concurrencia. Su uso está enfocado a situaciones donde la complejidad de un servidor de base de datos completo no está justificada, y se prioriza una interfaz simple de inserción, consulta y modificación de registros en colecciones de documentos [27][28].

A partir de la revisión realizada, se observa que existen diversas herramientas y entornos ampliamente utilizados para el control y la planificación de movimientos en robótica, como ROS y MoveIt.

En este contexto, el presente trabajo no busca desarrollar nuevas herramientas, sino integrar tecnologías existentes en un sistema accesible orientado al ámbito educativo. La propuesta se centra en la implementación de un controlador y una interfaz que permitan a los

estudiantes interactuar con un robot colaborativo de bajo costo, facilitando la comprensión de conceptos fundamentales de robótica mediante una plataforma práctica.

De esta manera, se busca generar una herramienta didáctica que permita aproximar a los estudiantes a entornos reales de aplicación, mediante una solución accesible y adaptable.

1.6 Justificación

El presente proyecto se enmarca dentro de una iniciativa mayor del Laboratorio de Mecatrónica destinada a incorporar plataformas robóticas didácticas que representen, a escala, los sistemas utilizados en la industria. En este contexto, el desarrollo de un controlador para un robot colaborativo de seis grados de libertad contribuye a ampliar las herramientas disponibles para la enseñanza y la experimentación.

Si bien el eje principal del trabajo es la implementación del software de control, se integra también un prototipo físico del robot a fin de disponer de una plataforma funcional que permita validar el sistema y que, a futuro, pueda ser mejorada o utilizada en otras líneas de trabajo. De esta manera, el proyecto aporta tanto a la formación académica como al fortalecimiento tecnológico del laboratorio.

Asimismo, el desarrollo de este tipo de plataformas resulta relevante desde el punto de vista de la ingeniería, ya que permite integrar en un mismo sistema conceptos de control, planificación de movimiento, comunicación distribuida y desarrollo de interfaces, representando un caso de estudio completo de sistemas mecatrónicos modernos.

CAPITULO 2: Análisis y Desarrollo

En este capítulo se presenta el análisis y el diseño de los componentes que conforman el sistema —el robot, el controlador y la interfaz— así como la forma en que se organizan e interactúan entre sí. Cada uno de estos elementos cumple una función específica dentro del conjunto. A continuación, se describen en detalle sus características principales, su estructura y su comportamiento dentro del sistema.

Antes de avanzar con el análisis particular de cada parte, es necesario destacar algunos criterios considerados durante el proceso de diseño. Desde el inicio, el proyecto se planteó con un enfoque modular y escalable, de manera que cada componente pudiera modificarse, reemplazarse o analizarse de forma independiente sin afectar el resto del sistema. Asimismo, se priorizó que la solución desarrollada pudiera migrarse fácilmente a una Raspberry Pi, permitiendo prescindir de una computadora externa y facilitando su uso en entornos educativos o demostrativos.

2.1 Presentación general del sistema

El sistema completo esta compuesto por el robot(cuerpo), la base de datos(memoria), la arquitectura de ROS(cerebro) y la interfaz gráfica (interacción con el usuario) , cada parte representa un modulo que puede ser modificado o reemplazado permitiendo realizar mejoras de forma sencilla, escalar el sistema y efectuar pruebas puntuales.

2.1.1 Robot

El robot está compuesto por una estructura impresa en 3D, cuatro actuadores Dynamixel modelo XL-430, tres actuadores Dynamixel modelo XL-320, una placa controladora Arduino ATmega2560, dos módulos de botoneras, una placa controladora dedicada a los motores y una fuente de alimentación principal de 11.1 V junto con un variador de tensión LM2596 Step-down. En la Figura Nº 6 se muestra una vista general del robot con la identificación de sus principales elementos.

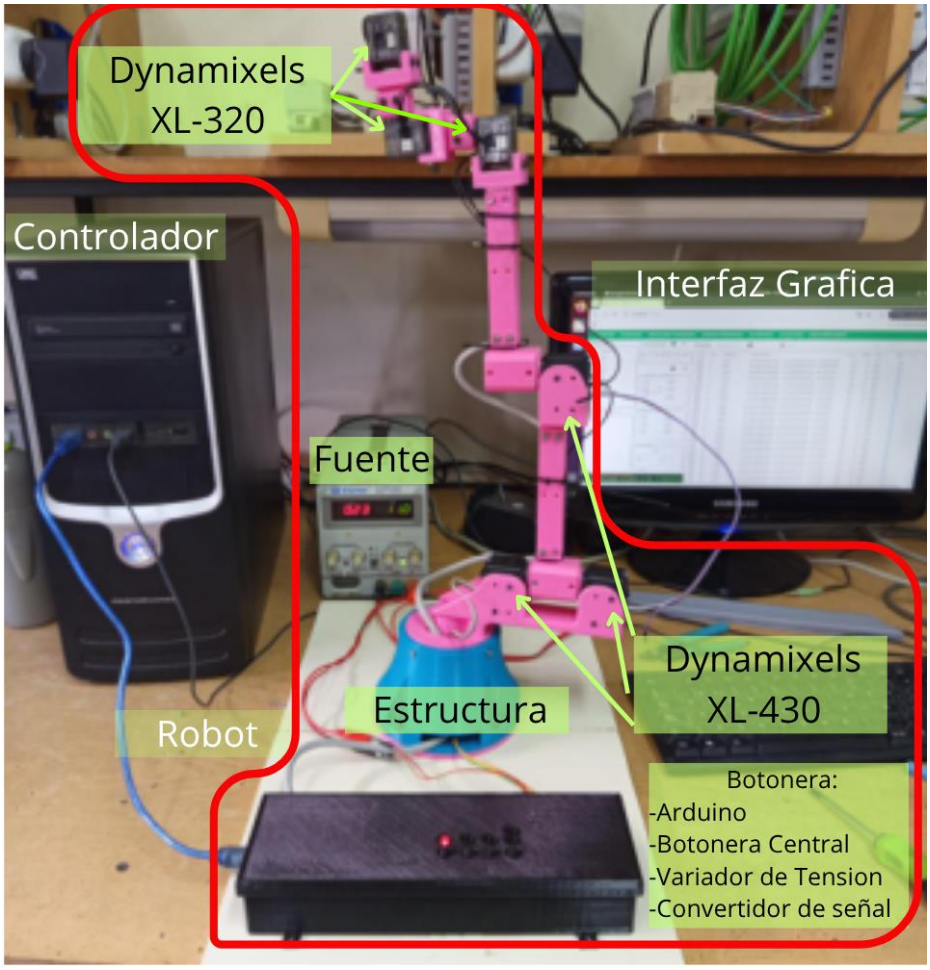

Figura Nº6 Cobot completo.

2.1.1.1 Estructura

El diseño estructural del robot fue desarrollado por **Gabriel Iglesias**, y se basa en una configuración típica de robots colaborativos, similar a la empleada por fabricantes industriales, por su estabilidad mecánica y facilidad para resolver la cinemática (Figura Nº7) [2]. Inicialmente, los eslabones correspondientes al brazo desde el hombro hasta la muñeca tenían una longitud mayor (Figura Nº8) (que había sido calculada aproximadamente por medio de simulación en base al torque máximo de los motores), pero se optó por reducirla debido a que, durante movimientos amplios, se generaban cargas elevadas sobre los actuadores del hombro. Esto producía corrientes pico que activaban los mecanismos de protección interna de los motores, ocasionando paradas o desconexiones durante el funcionamiento.

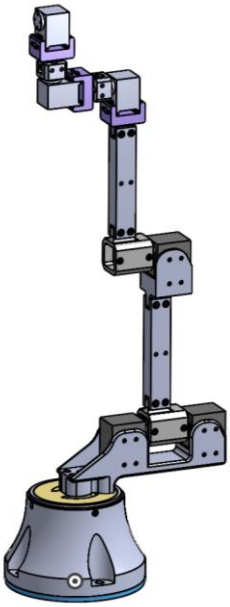

Figura Nº7 Diseño de Cobot actual.

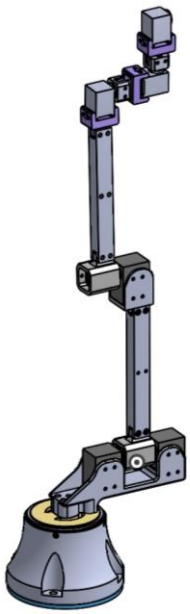

Figura Nº8 Diseño de Cobot con eslabones largos.

2.1.1.2 Actuadores

En cuanto a los actuadores, se emplearon motores Dynamixel XL-430 (Figura Nº9) y XL-320 (Figura Nº10). Estos componentes habían sido previamente adquiridos por el LabMe con fines de desarrollo de un robot colaborativo, por lo que resultaron adecuados para el presente proyecto. En un inicio, el sistema utilizaría únicamente Dynamixel XL-430 en todas las articulaciones; sin embargo, se decidió reemplazar los tres últimos Dynamixel por XL-320 debido a que estas articulaciones no requieren un gran torque y son mucho más livianos. Este cambio permitió reducir la carga total soportada por los primeros actuadores (cadera, hombro y codo), posibilitando un brazo de mayor alcance sin comprometer la seguridad ni la precisión.

Este reemplazo requirió modificaciones adicionales: los actuadores XL-430 operan óptimamente a 11.1 V, mientras que los XL-320 requieren 7.4 V y utilizan conectores diferentes. Para resolver esta incompatibilidad se incorporó Modulo Lm2596 Step-down (Figura Nº11), encargado de convertir la tensión de alimentación de 11.1 V a 7.4 V y se realizaron empalme del cable de datos entre los conectores diferentes (el de Dynamixel XL-430 y el Dynamixel XL-430) a parte de la conexión separa de la alimentación.

Aun así, el actuador del hombro continuaba presentando limitaciones en movimientos de gran amplitud debido al peso acumulado de los motores distales. Para mitigar este problema, se incorporó un segundo motor sincronizado en esta articulación, aumentando la capacidad de

torque útil y reduciendo los errores durante el movimiento. Posteriormente, esto se complementó con el acortamiento de los eslabones previamente mencionado.

Figura Nº9 Dynamixel XL-430 W250.

Fuente: tomado de [24]

Figura Nº10 Dynamixel XL-320

Fuente: tomado de [25]

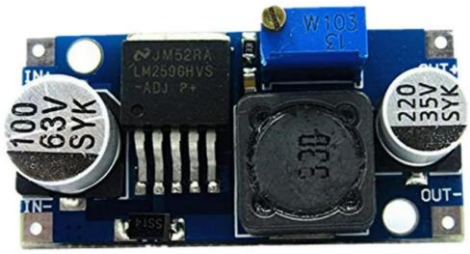

Figura Nº11 Modulo Lm2596 Step-down.

Fuente: tomado de [29]

2.1.1.3 Placas Electrónica

Considerando la futura migración del sistema ROS hacia una rasppberry pi se incorporó una botonera destinada a permitir el control básico del robot sin depender de la interfaz gráfica. Esto es especialmente útil para demostraciones rápidas, pruebas de laboratorio o exposiciones. La botonera se compone de dos módulos: uno ubicado en la base del robot, que permite seleccionar modos de operación y ejecutar la última rutina almacenada (Figura Nº12), y otro ubicado en el eslabón entre el codo y la muñeca, utilizado para desenclavar los motores y registrar puntos (Figura Nº13). Su funcionamiento detallado se describe más adelante.

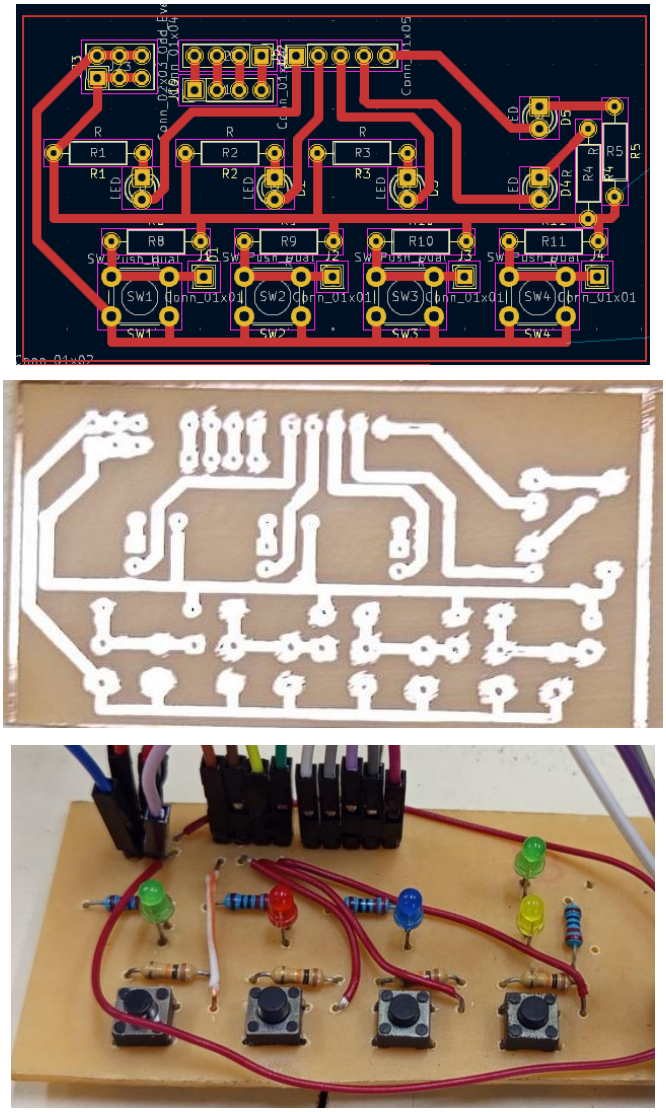

Figura Nº12 Placa de Botonera central.

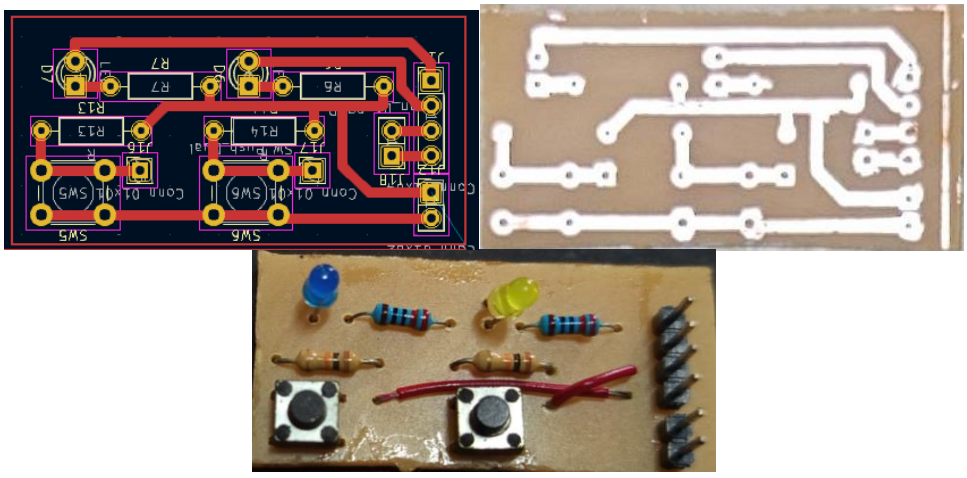

Figura Nº13 Placa de Botonera de lectura.

Estas placas corresponden a circuitos simples de interfaz, compuestos por pulsadores y LEDs. Cada pulsador está implementado como una entrada digital al Arduino mediante una resistencia de pull-down, mientras que los LEDs se controlan desde salidas digitales del mismo. En esencia, se trata de circuitos básicos e independientes entre sí, que únicamente comparten las líneas de alimentación (5 V y GND).

La placa no establece una conexión directa con el Arduino, sino que actúa como un elemento de ordenamiento del cableado. Para ello, se disponen hileras de pines que agrupan, por un lado, las entradas provenientes de los pulsadores y, por otro, las salidas hacia los LEDs. Esta disposición respeta el orden físico de los elementos en la botonera (de derecha a izquierda), facilitando la identificación y posterior conexión de cada señal al pin correspondiente del Arduino.

Adicionalmente, en la placa central se incorporó una bornera implementada mediante un conjunto de pines macho (dos hileras de tres pines interconectados), destinada a la distribución común de la alimentación. A través de esta, se vinculan las líneas de 5 V y GND, permitiendo que tanto el Arduino (como fuente de alimentación) como las distintas placas de la botonera compartan una referencia eléctrica común.

Además, se diseñó una tercera placa destinada al control de los actuadores Dynamixel por medio de arduino donde convierte señales UART al tipo semidúplex (Figura Nº14), esta se hizo a parte para que se pueda utilizar en cualquier otro proyecto que utilicen estos actuadores [24].

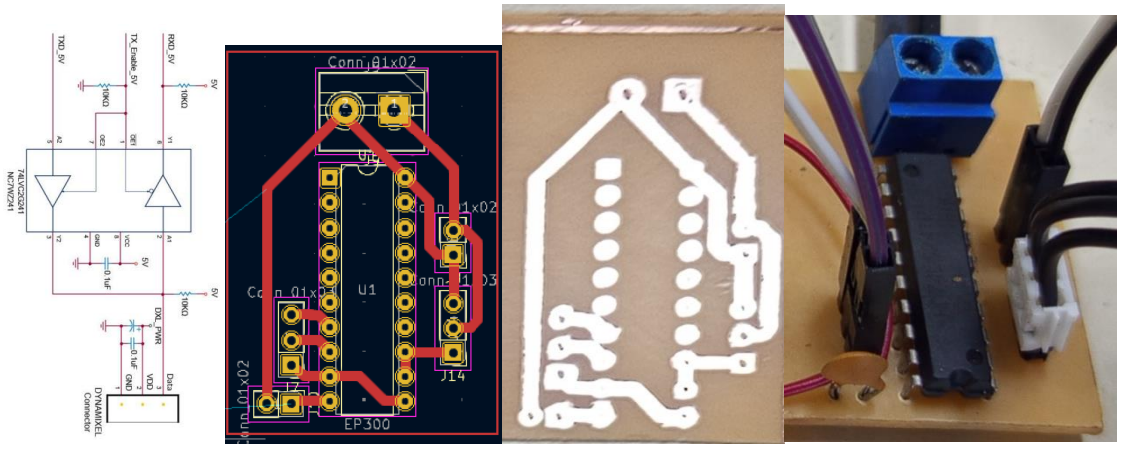

Figura Nº14 Placa para convertir señales UART al tipo semidúplex.

En cuanto a las conexiones, la placa dispone de pines destinados a vincular el puerto serie del Arduino (RX1 y TX1), junto con una línea de control digital (pin 2), utilizada para gestionar la dirección de la comunicación semidúplex. Una vez adaptada la señal, se dispone de un conector de tres pines para la conexión del bus de datos de los Dynamixel.

Por otra parte, se incorporan bornes para la alimentación de los actuadores, previendo el uso de una fuente de 11,1 V. A partir de esta misma línea, se dispone una salida adicional en paralelo destinada a alimentar un variador de tensión, el cual permite obtener un nivel adecuado (7,4 V) para los modelos XL-320. Finalmente, la placa incluye pines de conexión a la bornera de la botonera central, desde donde se obtiene la alimentación de 5 V y GND proveniente del Arduino.

Cabe destacar que todo el sistema comparte una referencia común de masa (GND), incluyendo tanto la alimentación del Arduino como la fuente de potencia de 11,1 V, garantizando así un correcto funcionamiento de las señales de control y comunicación.

A continuación se presentan los esquemas eléctricos correspondientes a las tres placas desarrolladas, donde se detallan las interconexiones internas y las señales disponibles para su vinculación con el resto del sistema (Figura Nº15).

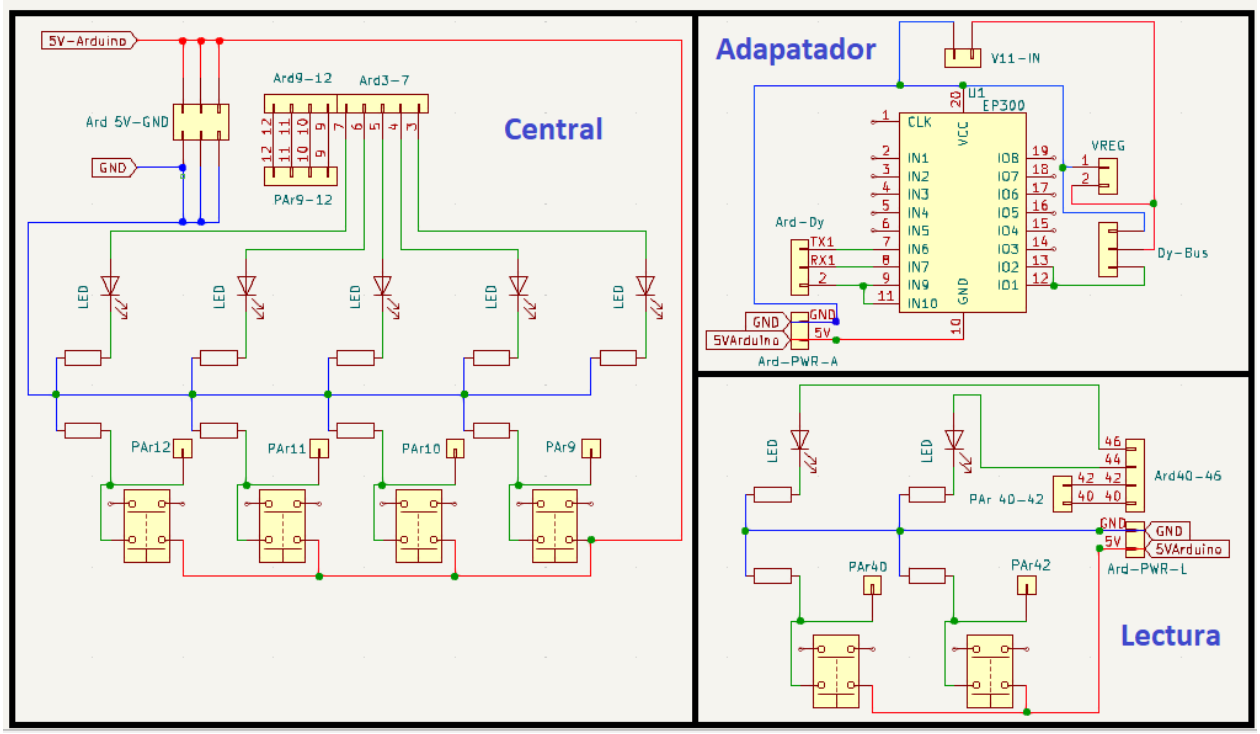

Figura Nº15: Circuitos de placas, Central, Lectura y Adaptador

En base a los esquemas presentados, en la siguiente tabla se resumen las señales disponibles en cada placa, indicando su función, tipo y destino de conexión dentro del sistema.

Tabla Nº2. Elementos de Sección de Coordenadas

Nombre	Placa	Tipo de conector	Tipo de señal	Destino / Conexión	Observaciones
Ard 5V-GND	Central	Pin	5V / GND	Arduino	Alimentación común
Ard9-12	Central	Pin	Digital INPUT	Pines 9–12 Arduino	Ordenado físicamente
Ard3-7	Central	Pin	Digital OUTUT	Pines D3–D7 Arduino	Ordenado físicamente
PAr9-12	Central	Cable	Digital	PAr9, PAr10, PAr11, PAr12	Distribución de señales
PAr12	Central	Cable	Digital	PAr9-12 (pin 12)	Cable externo por limitación PCB
PAr11	Central	Cable	Digital	PAr9-12 (pin 11)	Igual criterio
PAr10	Central	Cable	Digital	PAr9-12 (pin 10)	Igual criterio
PAr9	Central	Cable	Digital	PAr9-12 (pin 9)	Igual criterio
Ard-Dy	Adaptador	Pin	UART (half-duplex)	Arduino (TX1, RX1, 2)	Control semidúplex
Dy-Bus	Adaptador	Pin	semidúplex	Actuadores Dynamixel	Bus de datos Dynamixel
V11-IN	Adaptador	Bornera	Alimentación	Fuente 11.1 V	Alimentación principal
VREG	Adaptador	Pin	Alimentación	Variador de tensión	Salida al regulador
Ard-PWR-A	Adaptador	Pin	5V / GND	Ard 5V-GND	Alimentación común
Ard-PWR-L	Lectura	Pin	5V / GND	Ard 5V-GND	Alimentación común
Ard40-46	Lectura	Pin	Digital IN/OUT	Pines 40, 42, 44 y 46 Arduino	Ordenado físicamente
Par40-42	Lectura	Cable	Digital	Par40, Par42	Distribución de señales
Par40	Central	Cable	Digital	Par40-42 (pin 40)	Cable externo por limitación PCB
Par42	Central	Cable	Digital	Par40-42 (pin 42)	Igual criterio

2.1.1.4 Arduino

Por último, en la base del robot se integraron la placa controladora de Dynamixel, el módulo principal de la botonera, el variador de tensión Lm2596 Step-down y el controlador Arduino ATmega2560, todos conectados a la fuente de 11.1 V y con punto de neutro común.

En el sistema desarrollado, el Arduino cumple un rol central como interfaz entre el hardware del robot y la arquitectura de control basada en ROS. Sus funciones principales pueden resumirse en:

- **Comunicación con los actuadores Dynamixel:** mediante la Librería **Dynamixel2Arduino** [26], el Arduino obtiene la posición real de los motores cada 0.5 s y les envía las posiciones objetivo a donde deben moverse.

- **Comunicación con ROS e interfaz web:** utilizando la librería Rosserial, el Arduino actúa como un nodo de ROS, recibiendo los puntos generados desde el sistema y enviando la telemetría (posiciones reales y estados de la botonera) tanto a ROS como a la plataforma web.

- **Gestión de la botonera:** procesa los eventos y comandos provenientes de los módulos físicos de control manual.

En una primera etapa, el Arduino fue programado para enviar información de realimentación a ROS con una frecuencia al menos dos veces mayor que la frecuencia con la que recibía las coordenadas objetivo. A frecuencias bajas (5 Hz y 10 Hz), este esquema funcionaba de manera estable; sin embargo, el movimiento del robot resultaba notablemente entrecortado. Al aumentar la frecuencia de comunicación, comenzaron a manifestarse distintos problemas: el movimiento se volvía aún más irregular, la respuesta de la botonera se deterioraba y la visualización de datos presentaba inconsistencias.

Inicialmente, y en base a experiencias previas, se consideró que estas limitaciones estaban asociadas a la capacidad de procesamiento del Arduino. En consecuencia, se reestructuró el sistema para reducir al mínimo la carga computacional del microcontrolador, delegando cálculos y procesamiento a los nodos de ROS. Entre otras modificaciones, se eliminaron cálculos innecesarios en el Arduino y se configuró el sistema para que ROS envíe directamente los valores que debían transmitirse a los actuadores Dynamixel. No obstante, estas medidas no resolvieron el problema.

Posteriormente, se evaluó la influencia de la comunicación mediante Rosserial, optándose por reducir el tamaño de los mensajes transmitidos a través de los tópicos. Para ello, se reemplazaron tipos de mensajes complejos —como joint_states— por arreglos de tipo int16, lo que implicó modificaciones tanto en los nodos de ROS como en el código del Arduino. A pesar de esta optimización, tampoco se obtuvieron mejoras significativas en el desempeño del sistema.

Tras un análisis más exhaustivo, se identificó que el principal cuello de botella del sistema se encontraba en el protocolo de comunicación implementado por la Librería **Dynamixel2Arduino** [26]. Las pruebas realizadas mostraron que cada comunicación con un motor presenta una latencia de entre 2 y 3 ms, en condiciones óptimas de operación, y contempla un tiempo máximo de espera de hasta 100 ms antes de declarar la ausencia de un

motor. Como consecuencia, cada interacción con el conjunto de siete motores consume aproximadamente entre 14 ms y 21 ms, sin considerar el tiempo de procesamiento del resto del sistema, sino únicamente el envío o consulta de datos a los actuadores.

Paralelamente, se determinó experimentalmente que, para que el robot ejecute trayectorias de manera fluida y sin movimientos entrecortados, es necesario trabajar con una frecuencia mínima de 20 puntos por segundo. Ante estas restricciones, se adoptó como solución enviar las posiciones objetivo a una frecuencia de 20 Hz, mientras que la lectura de las posiciones reales se redujo a 2 Hz, utilizándola únicamente con fines de visualización y monitoreo por parte del usuario.

Esto generó la necesidad de programar un nodo que controle el envío de puntos a esa frecuencia (nodo "**controlador_cobot"** que se explicara más adelante)

2.1.2 Base de Datos

Para este proyecto se consideró desde el inicio la posibilidad de migrar el controlador a una Raspberry Pi. Por este motivo, se buscó una base de datos ligera, simple y con buen rendimiento, capaz de funcionar correctamente en entornos de recursos limitados y compatibles con el sistema operativo utilizado. Además, se evaluaron aspectos como la facilidad de integración con Python y la forma en que el sistema produce y consume los datos.

Para la elección final se consultó a un colega con amplia experiencia profesional en bases de datos y en sistemas Linux. Luego de presentarle los criterios de diseño y el uso previsto dentro del controlador, recomendó emplear TinyDB una base de datos NoSQL basada en archivos JSON, de bajo consumo de recursos y muy adecuada para aplicaciones de tamaño medio que requieren simplicidad y rapidez de implementación [27][28].

2.1.2.1 db_puntos.py

Con el fin de facilitar el acceso a la base de datos y mantener el software organizado, se desarrolló un archivo independiente que funciona como intermediario entre el sistema y TinyDB. Este módulo centraliza todas las funciones necesarias para leer, almacenar, modificar y eliminar información, proporcionando una interfaz clara y uniforme para el resto del proyecto. Gracias a esta separación, los demás componentes no necesitan interactuar directamente con TinyDB, lo que simplifica el desarrollo, evita duplicación de código y mejora la mantenibilidad y escalabilidad del sistema.

2.1.3 Arquitectura ROS:

En este proyecto, ROS se utiliza como la plataforma principal para organizar, coordinar y comunicar los distintos componentes del sistema. Los nodos desarrollados en Python interactúan entre sí mediante tópicos, servicios y acciones, lo que permite estructurar el controlador del robot de manera modular y distribuida.

Si bien algunos nodos realizan funciones directamente relacionadas con ROS —como el cálculo de trayectorias, la ejecución de movimientos a través de MoveIt/Pilz o la obtención de la cinemática— otros cumplen tareas adicionales, como gestionar la comunicación con Arduino, interactuar con el módulo de la base de datos o procesar la información proveniente de la plataforma web. Estos procesos no forman parte del núcleo de ROS, pero se integran dentro de nodos para mantener una arquitectura unificada y facilitar la comunicación entre todos los módulos del sistema.

A continuación, se describen los nodos que conforman el controlador y las funciones principales que realiza cada uno.

2.1.3.1 modo_lectura (modo_lectura.py)

El nodo modo_lectura actúa como el administrador central de toda la información del sistema. Se encarga de recibir los datos reales enviados por Arduino —que corresponden a las posiciones del robot y órdenes de la botonera—, procesarlos, convertirlos a formatos utilizables y enviarlos a la interfaz web para su visualización. Al mismo tiempo, cuando la interfaz web realiza alguna acción (por ejemplo agregar, modificar o eliminar puntos o rutinas), este nodo interpreta esos comandos y se comunica con el módulo encargado de la base de datos para guardar, actualizar o recuperar la información solicitada y posteriormente darle una devolución a la interfaz para corroborar los cambios o informar fallos.

2.1.3.2 modo_control (trayectoria.py)

Este nodo es el encargado de coordinar el movimiento del robot. Recibe las órdenes para ejecutar una rutina completa o una trayectoria simple, ya sea desde la plataforma web o desde la botonera. A partir de estas órdenes, consulta los puntos almacenados en la base de datos o utiliza la información enviada por la plataforma web para trayectorias simples. Con estos datos arma la secuencia correspondiente y se la envía al planificador de MoveIt (Pilz), que

genera los planes de movimiento del brazo robótico de acuerdo con las indicaciones establecidas, ya sea en términos de velocidad, precisión o tipo de trayectoria. Además, administra las pausas programadas. De esta manera, el nodo funciona como el “centro de control” que organiza, gestiona y ejecuta todos los movimientos del robot

2.1.3.3 controlador_cobot (controlador_rutina_prueba.py)

El nodo controlador_cobot cumple el rol de coordinador de la ejecución de trayectorias generadas por MoveIt. Su función principal es interceptar los planes de movimiento enviados por el nodo de planificación, validar su estado mediante el feedback de MoveIt/Pilz y, una vez verificados, transformar y enviar los puntos articulares al Arduino para que el robot los ejecute físicamente. Además, interpreta si debe ejecutarse una trayectoria calculada por MoveIt o una trayectoria bruta guardada por el usuario (sub-modo trayectoria), y administra los tiempos de espera definidos en la rutina.

Este nodo trabaja en conjunto directo con el nodo modo_trayectoria, que es el encargado de solicitar a MoveIt la generación de los planes. Una vez que modo_trayectoria inicia una planificación, controlador_cobot recibe la señal de control correspondiente, espera la llegada de los fragmentos del plan junto con el feedback de estado (“PLANNING”, “MONITOR”, “IDLE”), y determina si la planificación fue válida. En caso afirmativo, ejecuta el plan completo enviando los puntos articulados ya convertidos al formato requerido por los servomotores Dynamixel. Durante la ejecución informa a modo_trayectoria el avance mediante mensajes específicos, indicando si debe enviar el siguiente movimiento, si la ejecución fue correcta o si ocurrió un error.

De esta manera, controlador_cobot actúa como el módulo responsable de supervisar, validar y ejecutar los movimientos del robot físico, garantizando que solo se ejecuten trayectorias correctas y administrando eventos adicionales como pausas programadas y trayectorias capturadas manualmente. Es el nexo entre el planificador de alto nivel (MoveIt) y el actuador final (Arduino), asegurando que la ejecución siga el orden y la estructura definidos en las rutinas programadas.

2.1.3.4 publicador_posicion_real (publicacion_posicion_real.py)

Este nodo recibe las posiciones de los motores enviadas por el Arduino, organiza esos datos en un mensaje adecuado y los publica para que el sistema de control (MoveIt) conozca

en todo momento la posición real del robot. Esto permite que los cálculos de movimiento se realicen siempre en base al estado actual del brazo. Aunque esta tarea podría ejecutarse directamente en el Arduino, se decidió trasladarla a ROS para reducir la carga de procesamiento sobre el microcontrolador y evitar demoras, ya que constituye el principal cuello de botella del sistema.

2.1.4 Interfaz Gráfica

Para la interfaz gráfica se decidió desarrollar una plataforma web accesible mediante red LAN, considerando que en el futuro el sistema será migrado a una Raspberry Pi. De esta forma, la interfaz puede visualizarse desde cualquier dispositivo conectado a la misma red, ya sea una PC, notebook, tablet o teléfono.

En una primera etapa, la interfaz fue programada de forma tradicional utilizando HTML, CSS y JavaScript, gestionando sus propios estados internos y almacenando temporalmente la información del usuario. Por ejemplo, la “Rutina Actual” existía únicamente en la interfaz gráfica, si el usuario actualizaba la vista o cerraba la pestaña, todos los datos se perdían, y solo se comunicaba con la Arquitectura ROS al guardar o ejecutar una rutina. Este enfoque dio lugar a la primera versión de la interfaz (Figura Nº16).

Figura Nº16 Primera Versión de Pagina Web.

Sin embargo, durante la implementación del control de errores surgieron diversas limitaciones. En primer lugar, se evidenció que la interfaz no podía verificar si una instrucción era válida hasta el momento de ejecutarla, lo que podía generar rutinas con puntos incorrectos o imposibles de realizar. Esto, además, dificultaba que el usuario mantuviera el hilo de lo que

estaba programando, especialmente en rutinas extensas, o lo obligaba a ejecutar cada punto individualmente para comprobar su validez. En segundo lugar, se buscó que la “Rutina Actual” fuera más robusta y quedara almacenada en la base de datos, en lugar de depender únicamente del estado temporal de la interfaz, aunque siguiera siendo un dato efímero que se reinicia junto con el controlador. También se identificaron funciones que mejorarían significativamente la comodidad del usuario, como la eliminación de múltiples puntos. Finalmente, se reconoció la necesidad de manejar información altamente dinámica —como la posición en tiempo real del robot—, lo cual requería una plataforma y un enfoque más adecuados.

Para resolver estas limitaciones se replanteó por completo el funcionamiento de la interfaz. La plataforma web pasó a actuar como un visualizador del estado real del sistema. Es decir, la interfaz envía una orden, el sistema la procesa y, una vez validada y ejecutada, devuelve a la interfaz los datos que deben mostrarse, aunque coincidan con los que el usuario haya ingresado. De esta manera se garantiza coherencia, seguridad y sincronización entre la interfaz y el controlador.

Como parte de esta reestructuración se decidió reescribir toda la interfaz en React, lo que permitió mejorar la dinámica visual, simplificar la implementación de funciones complejas y estructurar el código de forma modular, facilitando futuras expansiones del sistema. El resultado de esta mejora se observa en la segunda versión de la interfaz (Figura Nº17).

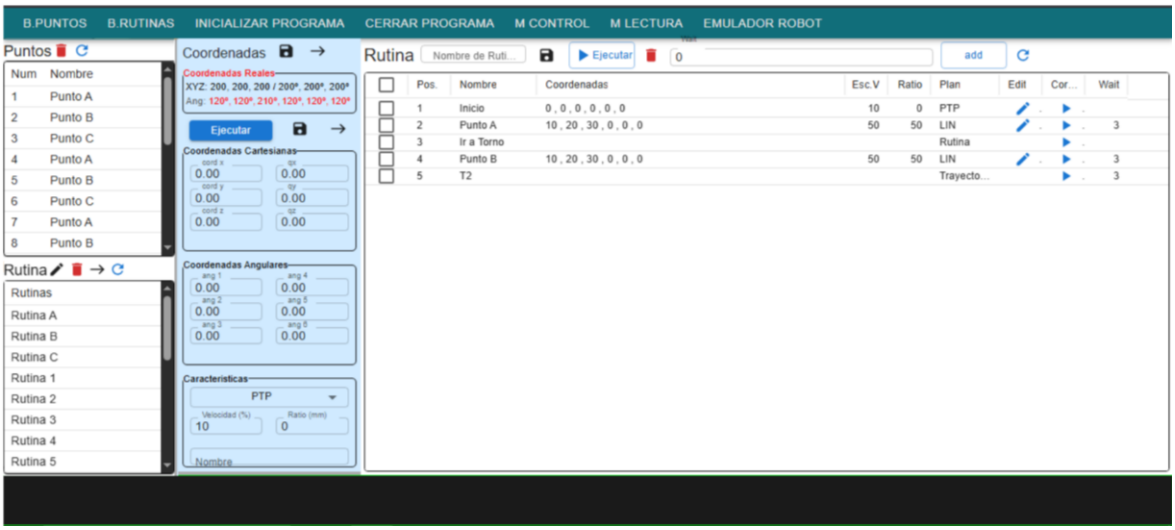

Figura Nº17 Pagina Web actual.

2.2 Conexiones

En este proyecto, el framework **ROS** actúa como la plataforma central para organizar, coordinar y comunicar los distintos componentes del sistema: el robot, los nodos de control, y la interfaz web. Cada nodo cumple una función específica, pero su comportamiento depende de la información que intercambian entre sí mediante **tópicos**, **servicios** y **acciones**. Adicionalmente, la **base de datos** se integra al sistema mediante un módulo que interactúa directamente con los nodos de ROS, permitiendo almacenar y recuperar información relevante para la operación del robot.

Si bien no se profundizara en la implementación interna de cada conexión, se presenta una visión general del flujo de comunicación que permite comprender cómo se integra cada parte del sistema. En la Figura Nº18 se muestra de manera esquemática la interacción entre los nodos de ROS, el microcontrolador Arduino y la interfaz web, representando el recorrido de la información dentro del sistema.

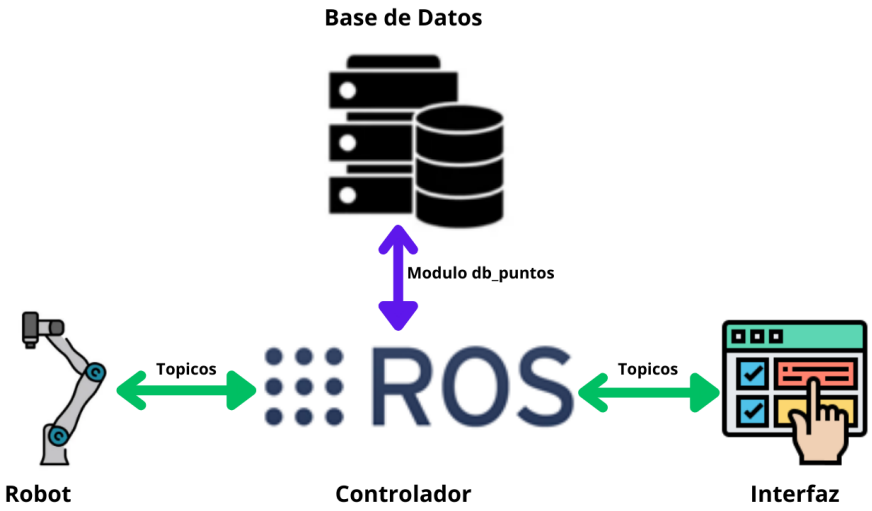

Figura Nº18 Interacción entre elementos del sistema.

Para la comunicación del sistema se adoptó un enfoque orientado a la eficiencia, procurando utilizar mensajes de tamaño reducido siempre que la aplicación lo permitiera, sin incorporar una cantidad excesiva de tópicos que pudiera afectar el desempeño general. Se buscó así un equilibrio entre el volumen de información transmitida y la cantidad de canales de comunicación, garantizando que cada mensaje contenga únicamente los datos necesarios para su función. En este marco, se definieron y desarrollaron distintos tipos de mensajes específicos,

adaptados a las necesidades de cada tópico, los cuales se detallan en la (Tabla Nº16 del anexo 1).

Con el fin de ofrecer una referencia precisa para futuros desarrolladores y estudiantes —especialmente quienes deseen modificar, extender o reemplazar algún módulo—, en la (Tabla Nº17 del anexo 2) se detalla cada tópico utilizado. Para cada uno se especifica el tipo de mensaje, su función dentro del sistema y los nodos que actúan como publicadores o suscriptores. Esta información permite comprender de forma directa qué influencia tiene cada tópico, dónde se origina cada dato y qué componentes dependen de él.

Complementariamente, la misma información se presenta desde la perspectiva de los nodos, organizada por programa o módulo. Esta estructura resulta útil para quienes necesiten modificar o analizar un nodo específico, ya que permite identificar de manera rápida todos los tópicos que utiliza, especificando si cumple el rol de publicador, suscriptor o ambos. (Tabla Nº18 del anexo 3).

Finalmente, para facilitar aún más la comprensión de la arquitectura del sistema, en la Figura Nº19 se incluye un diagrama detallado que representa todas las conexiones entre nodos y tópicos. Este diagrama constituye la vista más completa y visual del intercambio de información dentro del sistema, integrando tanto la parte del robot como la interfaz y los módulos internos de ROS.

En el gráfico, los rectángulos representan los nodos, las elipses los tópicos y las flechas indican el rol que cumple cada nodo en la comunicación. Una flecha orientada desde un nodo hacia un tópico indica que dicho nodo actúa como publicador, mientras que una flecha orientada desde un tópico hacia un nodo indica que este se comporta como subscriptor. En los casos en los que existen flechas en ambos sentidos, el nodo cumple simultáneamente ambas funciones. Además, las flechas se diferencian mediante colores para facilitar la identificación del nodo al que corresponden.

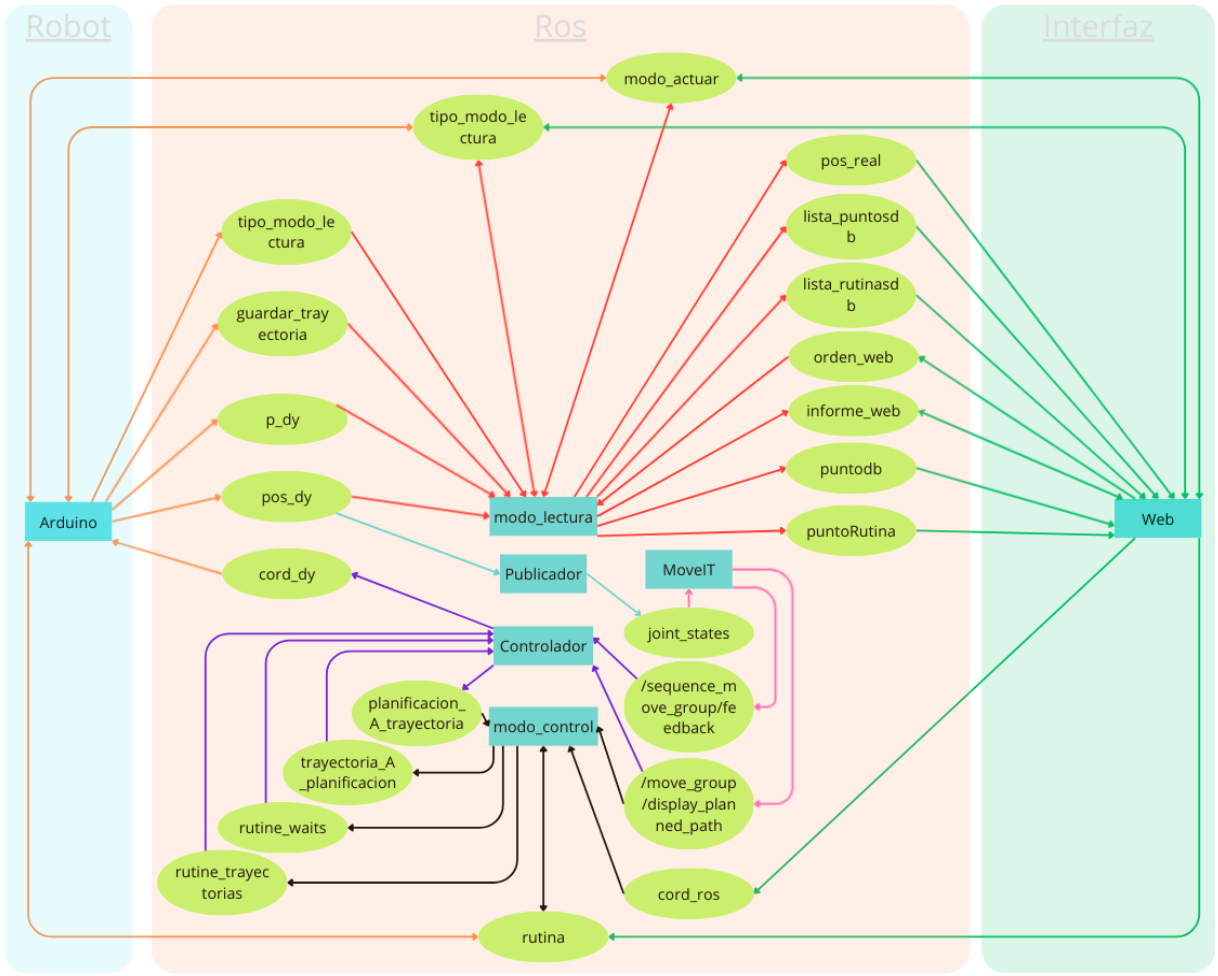

Figura Nº19 Diagrama de Nodos y Tópicos del proyecto.

2.3 Funcionamiento

En este capítulo se describe el funcionamiento del controlador desarrollado y la forma de interacción con el robot utilizado en las pruebas. El sistema se compone de distintos modos de operación, una interfaz gráfica y una botonera física, los cuales permiten al usuario programar y ejecutar movimientos del robot.

A continuación, se presentan de manera general estos elementos, para luego profundizar en el funcionamiento de cada uno de ellos. En primer lugar, se describen los modos de operación del controlador; posteriormente, se detallan los elementos de la interfaz gráfica y la botonera; y finalmente, se explica el uso del sistema y su comportamiento durante la ejecución.

2.3.1 Modos de uso

El controlador consta de 2 modos, con 2 sub-modos:

Modo control; en este modo el usuario puede hacer que el robot ejecute las rutinas programadas o puntos esecíficos, a su vez puede ir creando rutinas y guardando puntos (ingresados o las posiciones reales del robot), pero solo a través de la interfaz gráfica.

Modo Lectura; en este modo se bloquea la posibilidad de que el robot ejecute movimientos para protección del usuario, evitando que el robot pueda moverse mientras el usuario está manipulándolo. Solo en este modo el usuario puede guardar las posiciones reales del robot por medio de la botonera secundaria, y desenclavarlo para poder moverlo manualmente. Aparte de esto el usuario puede seguir programando rutinas y guardándolas junto con puntos a través de la plataforma web. Este modo cuenta con 2 sub-modos: Modo punto y Modo trayectoria.

Sub-Modo Punto: En este sub-modo cuando el usuario guarda la posición con la botonera, se guarda un único puto en la rutina con cada pulsación del botón de guardado. Aquí el controlador transforma los valores entregados por el arduino (0 a 4095 y 0 a 1023) a grados y pose, valores que el usuario y ROS respectivamente pueden entender.

Sub-Modo Trayectoria: En este sub-modo cuando el usuario guarda la posición con la botonera se guarda 20 puntos por segundos mientras el usuario mantenga pulsado el botón de desenclavar, y luego para guardar todos esos puntos en un único archivo el usuario debe presionar el botón de guardado. En este caso se guardan los datos tal cual entran y luego al correrlo se van a entregan a la misma frecuencia sin pasar por el planificador, lo que genera que copie tal cual la trayectoria hasta la velocidad con la que el usuario movió el robot al momento de guardarla.

2.3.2 Elementos

Botonera de control (1.X)

Tabla Nº3. Elementos de Botonera de control

1.x	Botonera de control		
1.1	Botón Modo Control B	1.6	Led Modo Lectura
1.2	Botón Modo Lectura B	1.7	Led Ejecutar Rutina
1.3	Botón Ejecutar Rutina	1.8	Led Sub-Modo Punto
1.4	Botón Sub-Modos	1.9	Led Sub-Modo Trayectoria
1.5	Led Modo Control		

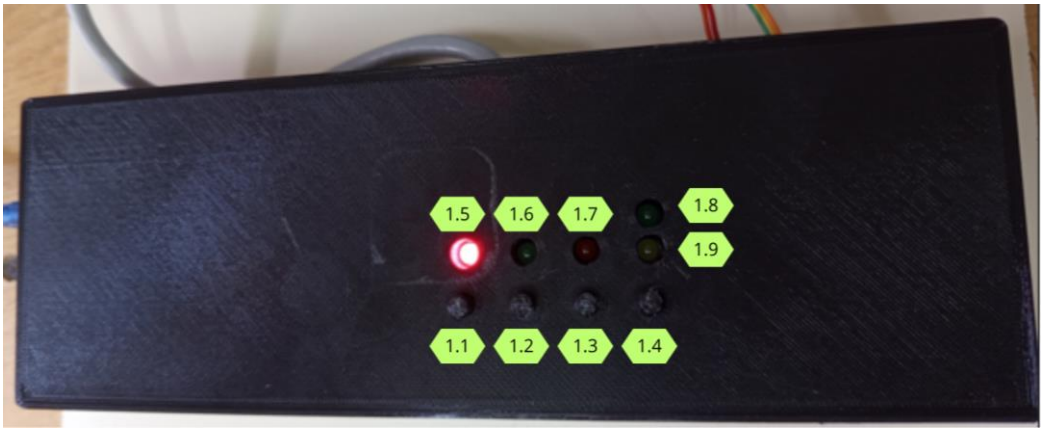

Figura Nº20 Botonera de control.

Botonera de Lectura (2.x)

Tabla Nº4. Elementos de Botonera de lectura

2.x	Botonera de Lectura
2.1	Botón de Desenclavar
2.2	Botón de Guardado
2.3	Led de Desenclavar
2.4	Led de Guardado

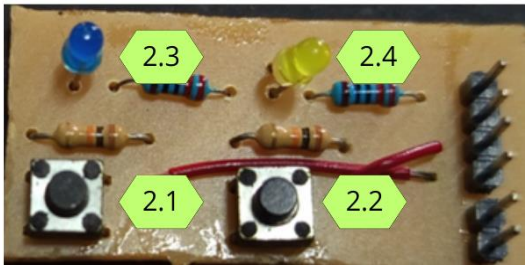

Figura Nº21 Botonera de lectura.

Plataforma Web (3.x)

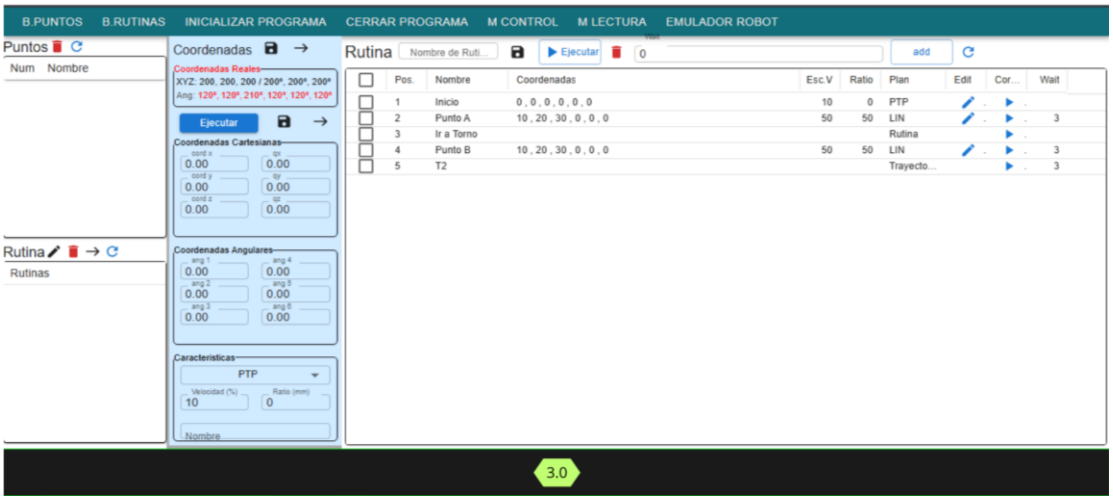

Figura Nº22 Pagina web con indicación Pantalla de mensajes (3.0).

Pantalla de mensajes (3.0):

Barra de herramientas (3.1.x):

Tabla Nº5. Elementos de Barra de herramientas

3.1.x	Barra de herramientas		
3.1.1	Botón de puntos guardados	3.1.4	Botón de Cierre de programa
3.1.2	Botón de Rutinas guardadas	3.1.5	Botón de Modo Control PW
3.1.3	Botón de Inicialización de programa	3.1.6	Botón de Modo Lectura PW

Figura Nº23 Barra de herramientas con enumeración de elementos.

Sección de Puntos Guardados (3.2.x):

Tabla Nº6. Elementos de Sección de Puntos guardados

3.2.x	Sección de Puntos Guardados
3.2.1	Botón Modo Lectura B
3.2.2	Botón Ejecutar Rutina
3.2.3	Lista de puntos guardados

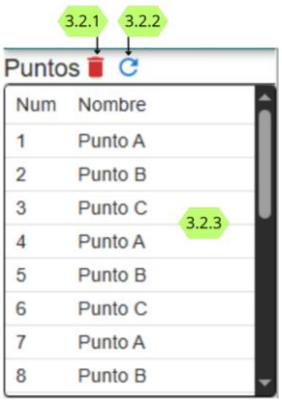

Figura Nº24 Sección de Puntos Guardados con enumeración de elementos.

Sección de Rutinas Guardadas (3.3.x):

Tabla Nº7. Elementos de Sección de Rutinas Guardadas

3.3.x	Sección de Rutinas Guardadas
3.3.1	Botón Editar Rutina
3.3.2	Botón eliminar rutina guardada
3.3.3	Botón agregar rutina guardada a la rutina
3.3.4	Botón refrescar lista de rutinas guardadas
3.3.5	Lista de rutina guardada

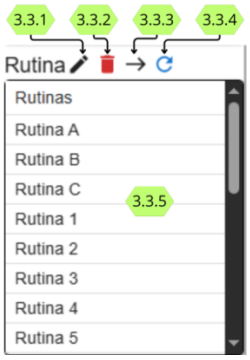

Figura Nº25 Sección de Rutinas Guardadas con enumeración de elementos.

Sección de Coordenadas (3.4.x):

Tabla Nº8. Elementos de Sección de Coordenadas

3.4.x	Sección de Coordenadas		
3.4.1	Botón guardar punto real	3.4.7	Inputs coordenadas cartesianas
3.4.2	Botón agregar punto real a la rutina	3.4.8	Inputs coordenadas angulares
3.4.3	Coordenadas reales	3.4.9	Desplegable tipo de trayectoria
3.4.4	Botón ejecutar punto	3.4.10	Velocidad del punto
3.4.5	Botón guardar punto	3.4.11	Precisión del punto
3.4.6	Botón agregar punto a la rutina	3.4.12	Nombre del punto

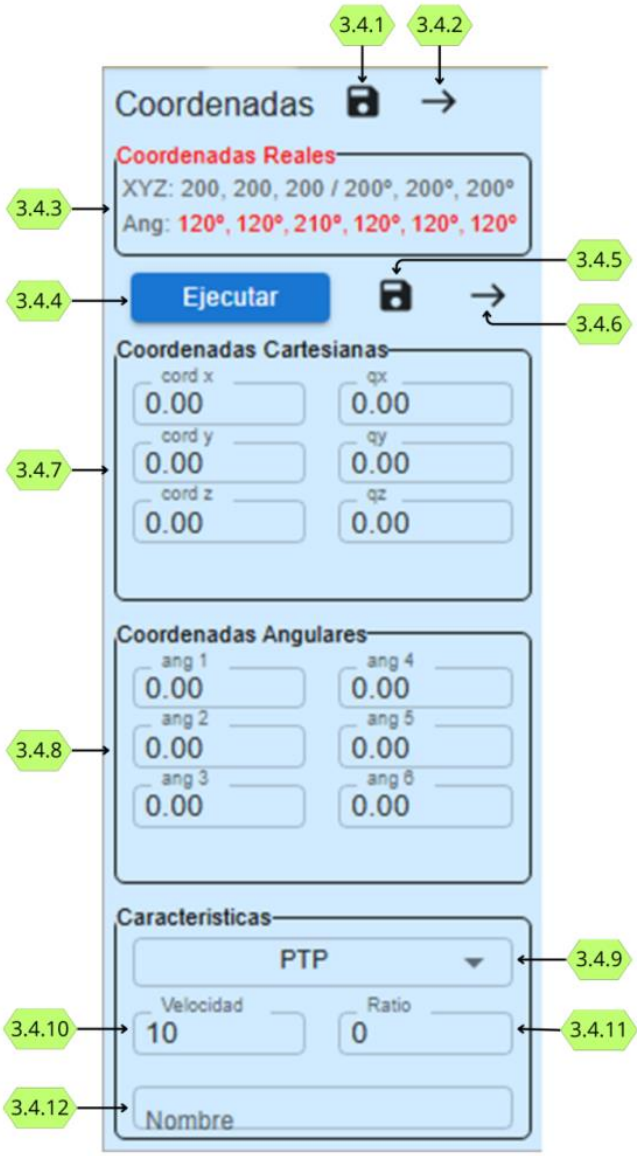

Figura Nº26 Sección de Coordenadas con enumeración de elementos.

Sección de Rutina (3.5.x):

Tabla Nº9. Elementos de Sección de Rutina

3.5.x	Sección de Rutina		
3.5.1	Nombre de la rutina	3.5.10	Posiciones de las instrucciones
3.5.2	Botón guardar rutina	3.5.11	Nombres de las instrucciones
3.5.3	Botón ejecutar rutina	3.5.12	Coordenadas de las instrucciones
3.5.4	Botón eliminar puntos seleccionados	3.5.13	Velocidad de la instrucción
3.5.5	Input tiempo de espera	3.5.14	Precisión de la instrucción
3.5.6	Botón agregar tiempo de espera	3.5.15	Tipo de instrucción
3.5.7	Botón refrescar	3.5.16	Botón de habilitar/confirmar edición de instrucción
3.5.8	Cuadro de rutina actual	3.5.17	Botón para correr instrucción
3.5.9	check de instrucciones de rutina	3.5.18	Tiempo de espera de la instrucción

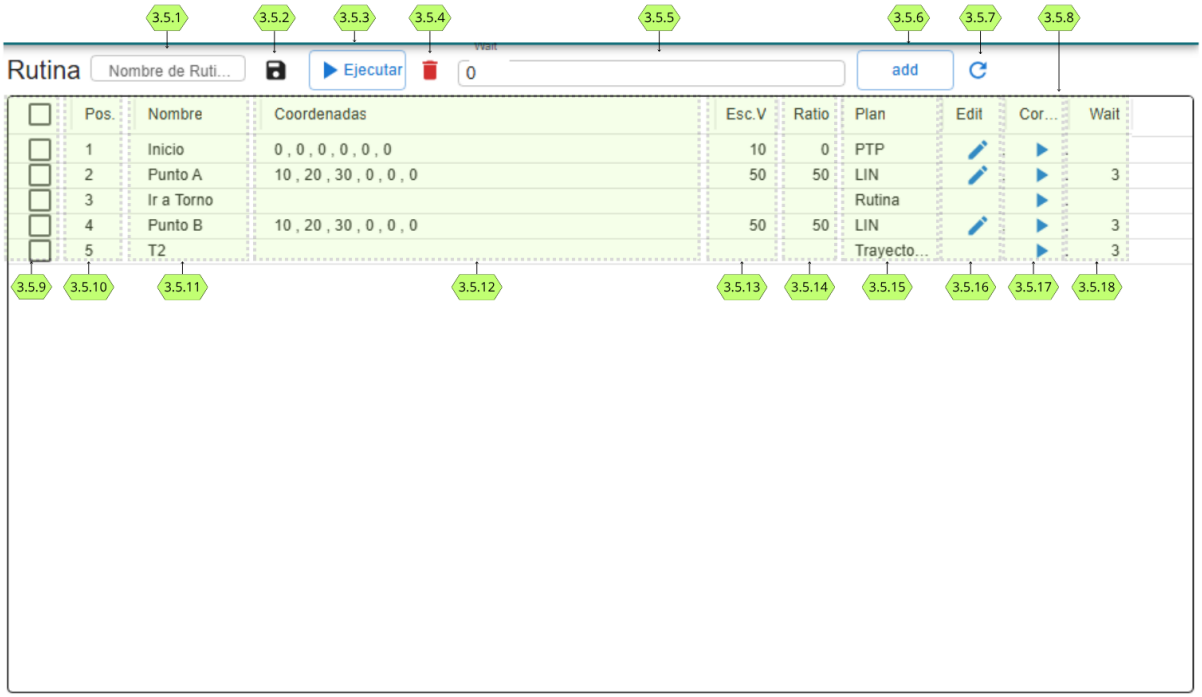

Figura Nº27 Sección de Rutina con enumeración de elementos.

2.3.3 Descripción

A continuación, se definen los conceptos principales utilizados:

Rutina actual (RA):

Es el conjunto de instrucciones guardada en una tabla volátil de la base de datos, esta es la que se ejecutará cuando se ordene correr la rutina, y donde el usuario programará. Es como un borrador donde se puede probar sin miedo a alterar los datos importantes guardados y sin la necesidad de guardar datos innecesarios al hacer pruebas, esta se borrará automáticamente al iniciar el controlador, en esta se copiarán las rutinas guardadas y solo ingresarán de vuelta a esta parte de la base de datos cuando el usuario realice el proceso de guardado, cuando la rutina creada o editada sea de su agrado. Se visualiza en el **Cuadro de rutina actual (3.5.8)**

Diagrama de secuencia (DS):

Con el objetivo de describir de manera clara y ordenada el funcionamiento del sistema desarrollado, se emplearon Diagramas de Secuencia (DS). Estos diagramas permiten representar, de forma simplificada, la interacción temporal entre los distintos componentes del

sistema, mostrando el intercambio de información y las acciones que se producen ante una determinada orden del usuario.

En los diagramas presentados se incluyen únicamente aquellos elementos que participan activamente en la acción analizada, tales como los actuadores (Dynamixel), la botonera, los microcontroladores Arduino, los nodos de control implementados en ROS, la base de datos, el planificador de movimientos (MoveIt) y la interfaz gráfica web. El objetivo de estos diagramas no es detallar la implementación interna de cada componente, sino brindar una visión general del flujo de información y del rol que cumple cada uno dentro del proceso.

A lo largo del informe, los Diagramas de Secuencia se utilizarán como apoyo para explicar de forma progresiva las distintas acciones que puede ejecutar el sistema, facilitando la comprensión del funcionamiento global del robot.

Tipos de instrucción:

En este controlador se tiene 3 tipos de instrucciones que puede hacer que corra el robot:

Punto: El robot va de la posición actual del a las coordenadas indicadas, si se ordena desde la **sección de Coordenadas (3.4.x)** cada motor va al ángulo indicado, en cambio sí se ordena desde **sección de Rutina (3.5.x)** el TCP del robot es el que va a la coordenada cartesiana indicada.

En esta instrucción se corre con un plan del tipo PTP (punto a punto) donde el robot va a las coordenadas indicadas con el menor movimiento de los motores.

Rutina: El robot realiza de forma ordenada una serie de instrucciones que tiene guardada, si bien por dentro es un conjunto de instrucciones tipo punto, se coloco asi para que el usuario y el controlador pueda diferenciarlas permitiendo que la programación de las rutinas sea modular y no haya que ingresar cada punto que desea ejecutar teniendo ya un conjunto prediseñados de estos.

Trayectoria: Este tipo de instrucción se genera solo en sub-modo trayectoria desde la botonera del robot y representa el conjunto de coordenadas tal cual entrega el arduino, explicado anteriormente en el “modo trayectoria” del “modo lectura”

Sección de Rutina (3.5.x):

Comenzaremos explicando cómo funciona esta sección, que es la parte principal del proyecto, en esta se puede visualizar, editar y guardar la (RA) (Figura Nº28).

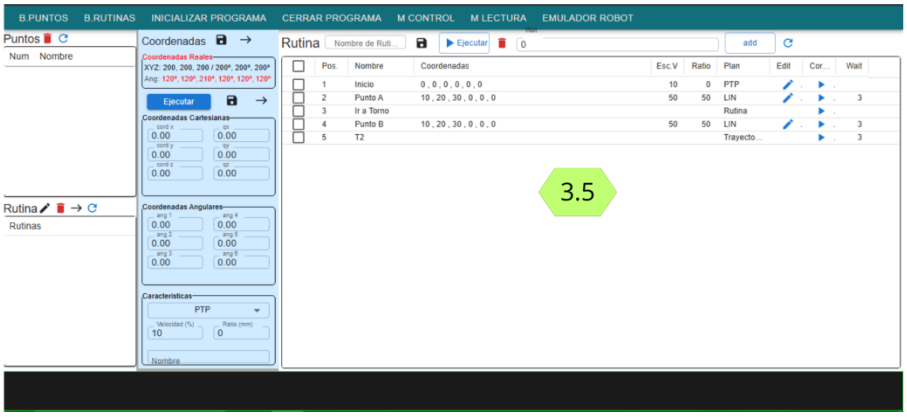

Figura Nº28 Sección de rutina (3.5) en Pagina Web.

Cuadro de rutina actual (3.5.8) se visualiza la (RA), donde cada fila representa una instrucción que, al momento de su ejecución, define una acción del sistema, mientras que las columnas contienen sus parámetros principales, los cuales se pueden modificar en su mayoría en la misma tabla (Figura Nº29).

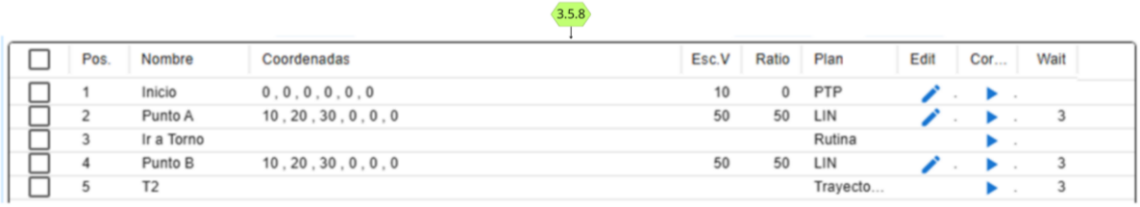

Figura Nº29 Cuadro de rutina actual (3.5.8).

check de instrucciones de rutina (3.5.9) sirve para seleccionar varias instrucciones y luego realizar acciones a estas, por el momento solo se pueden eliminar y agregar tiempos de espera (Figura Nº30).

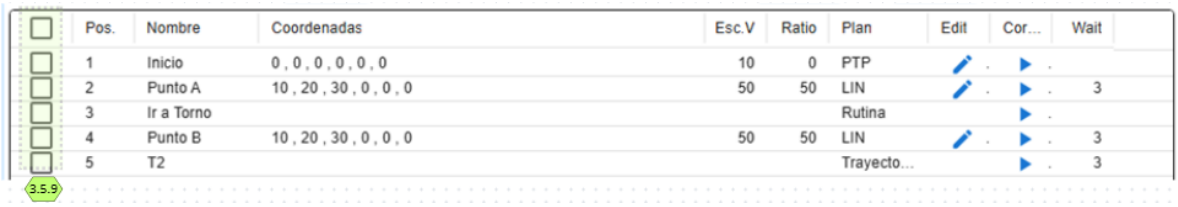

Figura Nº30 check de instrucciones de rutina (3.5.9).

Posiciones de las instrucciones (3.5.10) sirve para indicar el orden con el que se ejecutaran las instrucciones y por dentro como un identificador para algunos procesos (Figura Nº31).

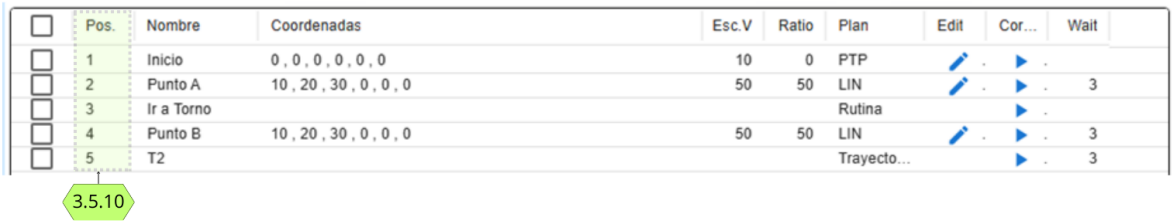

Figura Nº31 Posiciones de las instrucciones (3.5.10).

Coordenadas de las instrucciones (3.5.12): muestra las coordenadas Cartesianas de la instrucción, solo cuando es del tipo punto, para rutina y trayectoria esta casilla está vacía (Figura Nº32).

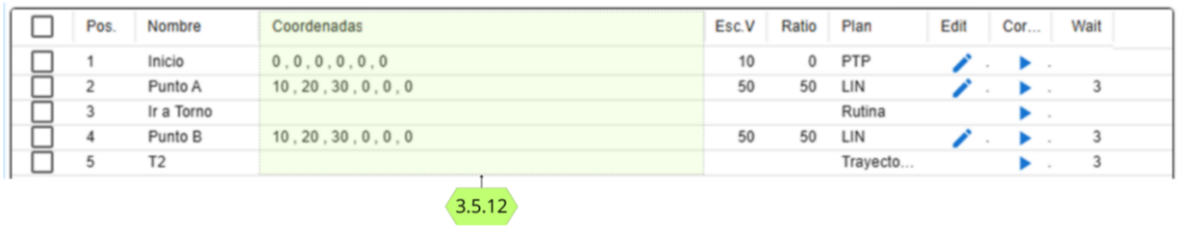

Figura Nº32 Coordenadas de las instrucciones (3.5.12).

Velocidad de la instrucción (3.5.13): indica la velocidad porcentual a la que se va a mover en esa instrucción siendo la velocidad máxima(100) de 2rad/seg, solo para puntos, para rutina y trayectoria esta casilla está vacía (Figura Nº33).

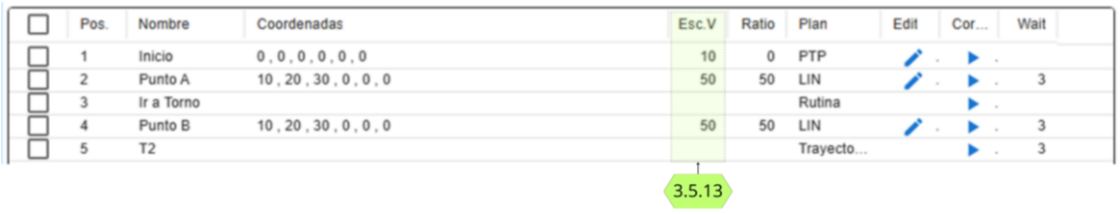

Figura Nº33 Velocidad de la instrucción (3.5.13).

Exactitud de la instrucción (3.5.14): (Figura Nº34) indica que tan cerca del punto el TCP va a llegar en esta instrucción en milímetros, esto ayuda a que la trayectoria de la rutina sea más suave, ejemplo: cuando es 0 el TCP va a ir a la posición especificada, cuando sea 10 va a llegar a 1cm de la posición especificada y luego seguirá a la siguiente instrucción, solo

para puntos, para rutina y trayectoria esta casilla está vacía. La variable en ROS se llama “blend radium” (Figura Nº35).

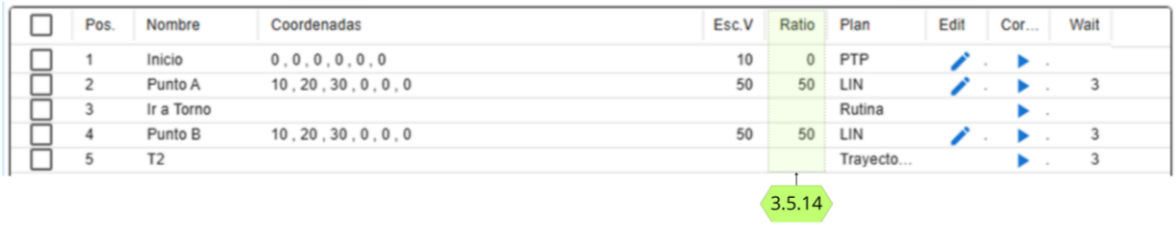

Figura Nº34 Exactitud de la instrucción (3.5.14).

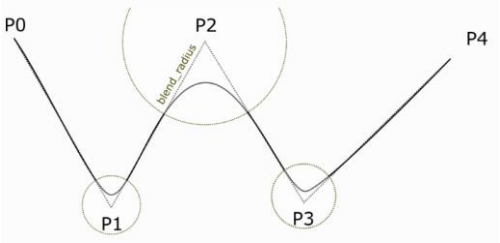

Figura Nº35 Representación del blend radius en Pilz MoveIt

Fuente: tomado de [14].

Tipo de instrucción (3.5.15): Indica el tipo de instrucción, esta se puede modificar mediante un menú desplegable solo cuando son del tipo punto y se muestran como PTP o LIN; para rutina y trayectoria no se permite (Figura Nº36).

Figura Nº36 Tipo de instrucción (3.5.15).

Botón de habilitar/confirmar edición de instrucción (3.5.16): (Figura Nº37) Normalmente, las casillas de la tabla no son editables; para habilitar su modificación, en esta columna se presenta un ícono de lápiz azul que, al ser presionado, cambia a un símbolo ✓ y permite editar los valores de la fila correspondiente, solo en caso de que sea una instrucción tipo punto, sino solo la columna de tiempo de espera(wait) puede ser modificada esta, para cualquier tipo de instrucción la columna posición no puede ser modificada; esta es la única forma de modificar las instrucciones, sino se deben borrar y volver a ingresarla. En este estado

se borra el **Botón para correr instrucción (3.5.17)** (Figura Nº37) (Figura Nº38) bloqueando la posibilidad de correr esa instrucción ya que no se puede asegurar que lo mostrado concuerde con lo guardado en la base de datos.

Al volver a apretar sobre el símbolo de esta columna, que ahora es ✓, indica al sistema que quiere guardar la modificación y recién ahí lo envía al controlador, este verifica si es una instrucción valida e informa a la plataforma web, si no es aceptable se muestra el error en la Pantalla de mensajes (3.0) y el símbolo ✓ no cambia, si es aceptable el símbolo de la columna vuelve al del lápiz azul original el controlador guarda la instrucción en la tabla de RA; esto ayuda a que ocurra menos errores al correr rutinas por instrucciones invalidas.

Figura Nº37 Botón de habilitar/confirmar edición de instrucción (3.5.16) y Botón para correr instrucción (3.5.17).

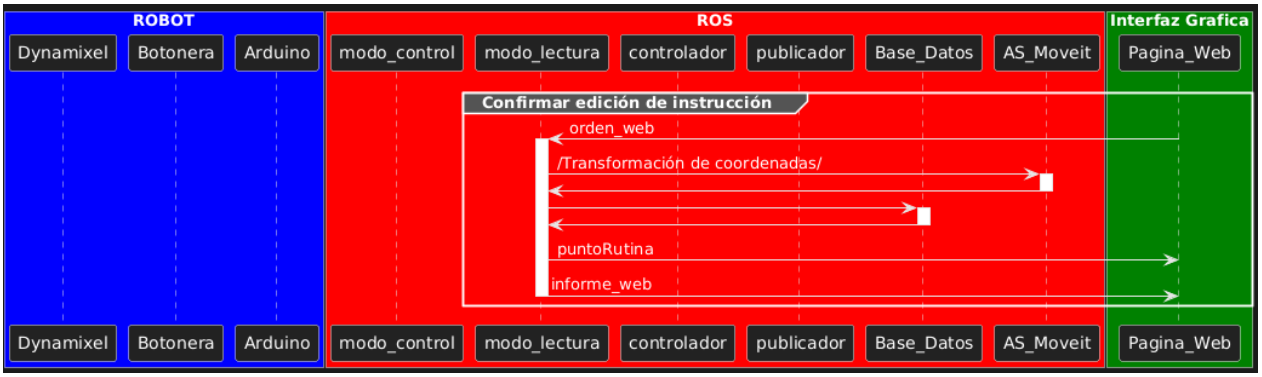

Figura Nº38 DS- Confirmar la edición de instrucción.

Botón para correr instrucción (3.5.17): sirve para que el robot corra esa instrucción, este icono desaparece en “modo lectura” y cuando se está modificando una instrucción (Figura Nº37) (Figura Nº39).

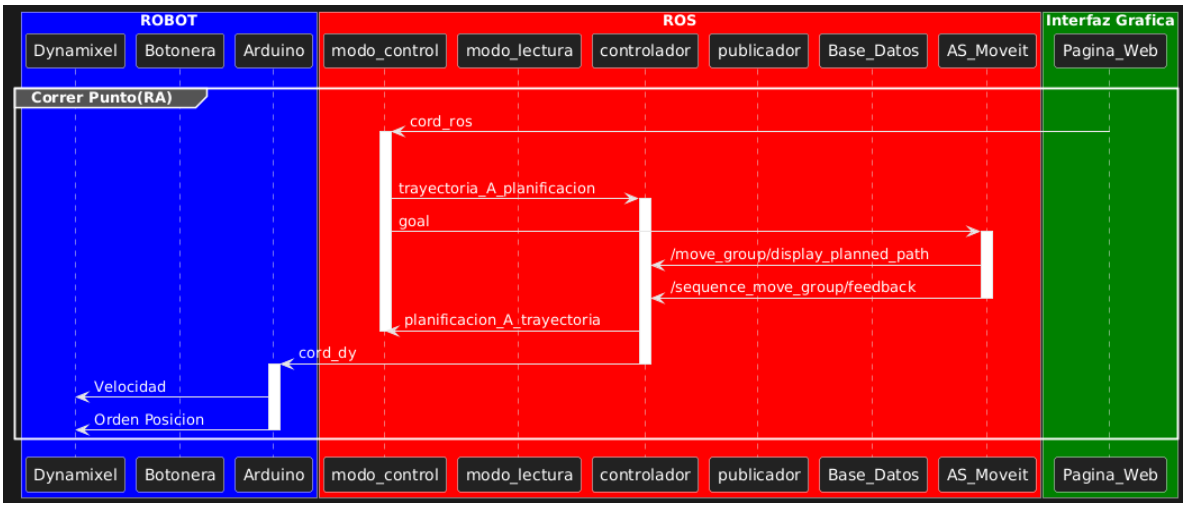

Figura Nº39 DS-Correr instrucción a través del cuadro de rutina.

Tiempo de espera de la instrucción (3.5.18): representado como “wait” en la tabla, este es el tiempo en segundos que el robot queda quieto luego de ejecutar la instrucción en donde se está, antes de correr la siguiente instrucción (Figura Nº40).

Figura Nº40 Tiempo de espera de la instrucción (3.5.18).

Botón guardar rutina (3.5.2): se utiliza para guardar la rutina que se encuentra en la RA en la base de datos, lo que pasa internamente es que la base de datos crea una tabla nueva copiando la RA con el nombre ingresado en **Nombre de la rutina (3.5.1)**, no es necesario rellenar este input (Figura Nº41) (Figura Nº42).

Si no se ingresó ningún nombre el sistema le crea uno genérico que nos se repita con los existentes.

Si el nombre ya existe no se guarda la rutina indicando que esta ya existe

Figura Nº41 Botón guardar rutina (3.5.2) y Nombre de la rutina (3.5.1).

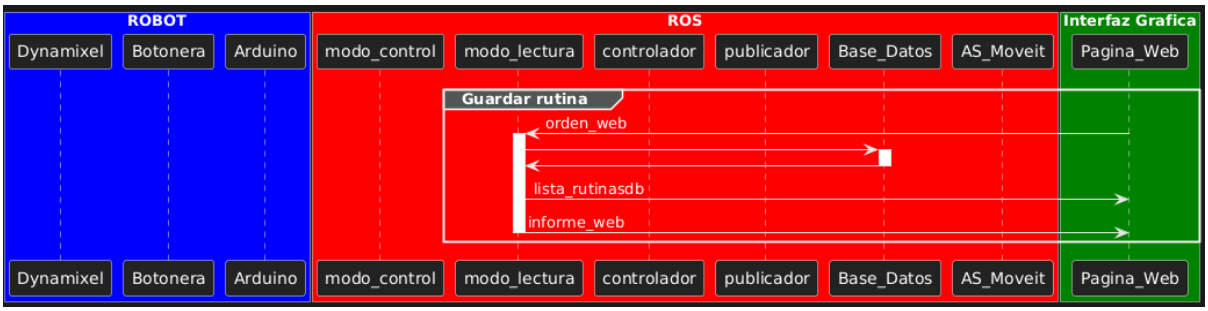

Figura Nº42 DS-Guardar Rutina.

Botón ejecutar rutina (3.5.3): Al apretar este botón se enviara la orden al controlador que ejecute la RA, si todo está en orden se le enviara la información al robot para que corra la rutina, sino, si hay cualquier error lo comunica en Pantalla de mensajes (3.0), por ejemplo que no exista alguna de las rutinas que se usen en esta (Figura Nº43) (Figura Nº44).

Figura Nº43 Botón ejecutar rutina (3.5.3).

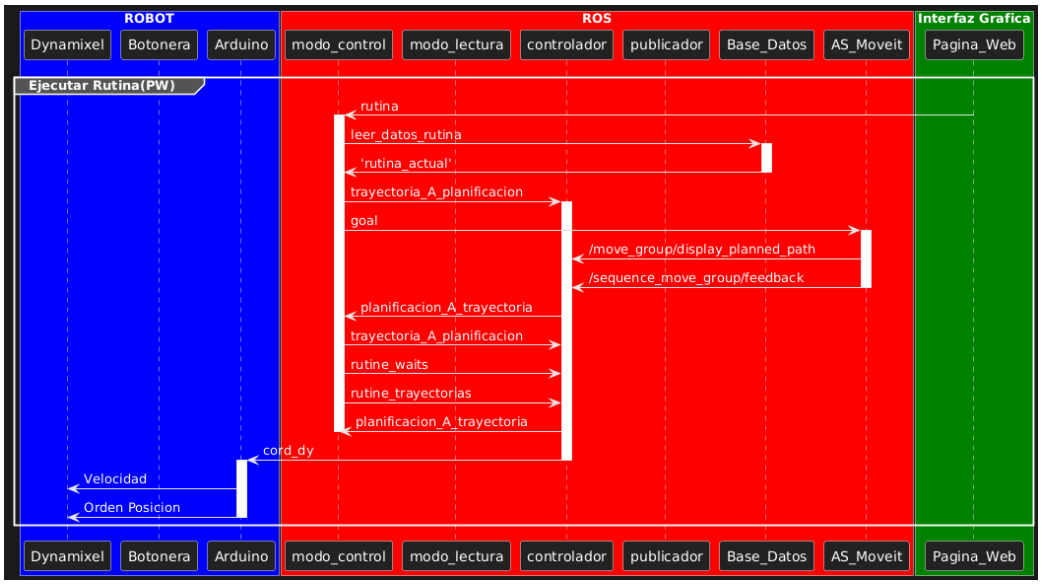

Figura Nº44 DS-Ejecutar rutina a través de la plataforma web.

Botón eliminar puntos seleccionados (3.5.4): al cliquear este botón se eliminaran las instrucciones previamente seleccionadas (**check de instrucciones de rutina (3.5.9)**) (Figura Nº45) (Figura Nº47), si alguna instrucción no existe se indicara al usuario por medio de una ventana emergente (Figura Nº46).

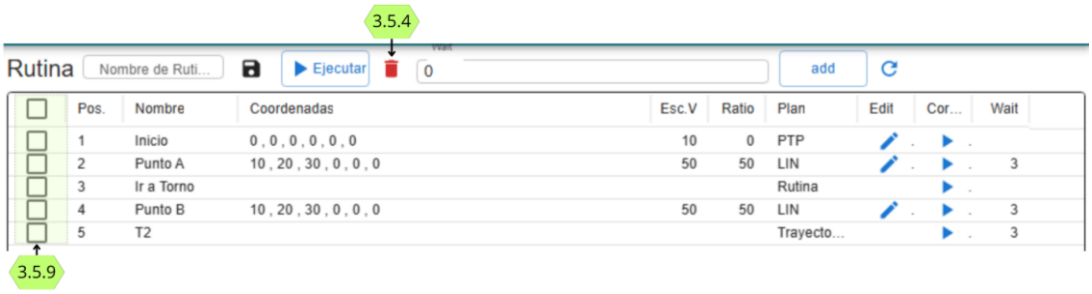

Figura Nº45 Botón eliminar puntos seleccionados (3.5.4) y check de instrucciones de rutina (3.5.9).

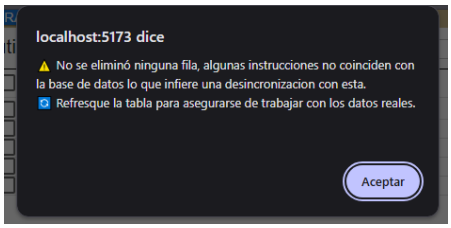

Figura Nº46 Ventana emergente de error al intentar eliminar.

Figura Nº47 DS-Eliminar puntos seleccionados de Rutina actual.

Botón agregar tiempo de espera (3.5.6): al cliquear este botón se agregara a las instrucciones previamente seleccionadas (**check de instrucciones de rutina (3.5.9)**) el tiempo indicado ingresado en **Input tiempo de espera (3.5.5)** (Figura Nº48) (Figura Nº49).

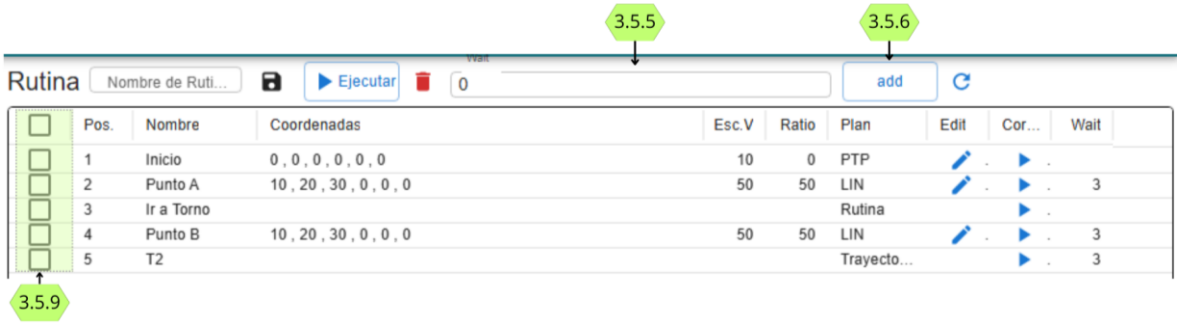

Figura Nº48 Botón agregar tiempo de espera (3.5.6), check de instrucciones de rutina (3.5.9) e Input tiempo de espera (3.5.5).

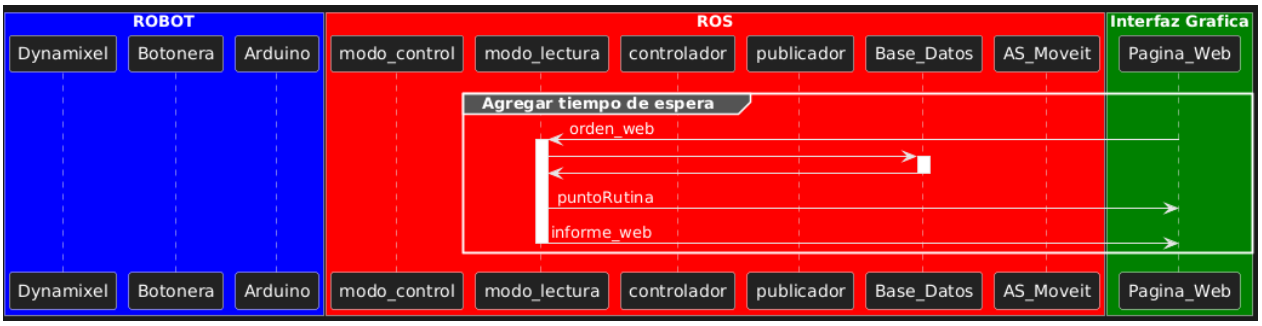

Figura Nº49 DS-Agregar tiempo de espera.

Botón refrescar (3.5.7): al cliquear este botón refresca el **Cuadro de rutina actual (3.5.8)** con la información de la RA, se usa por cualquier error que se desincronice la información mostrada con respecto a la que está en la base de datos, el sistema tiene incorporado verificaciones en diversas acciones para detectar alguna desincronización para informarle al usuario asi refresca la tabla (Figura Nº50) (Figura Nº51).

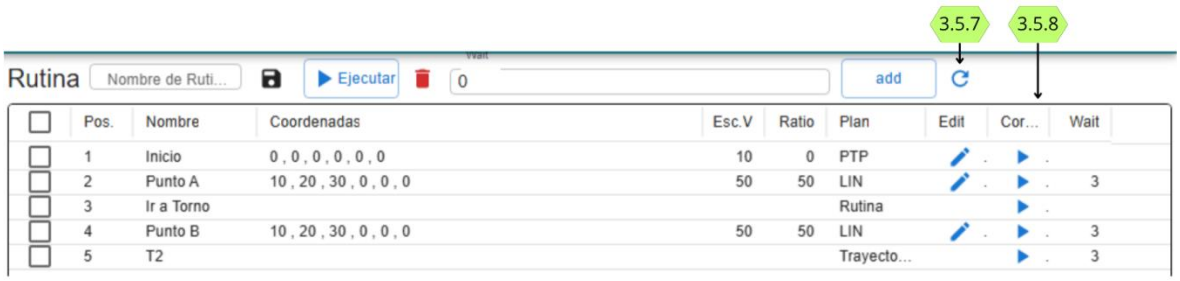

Figura Nº50 Botón refrescar (3.5.7) y Cuadro de rutina actual (3.5.8).

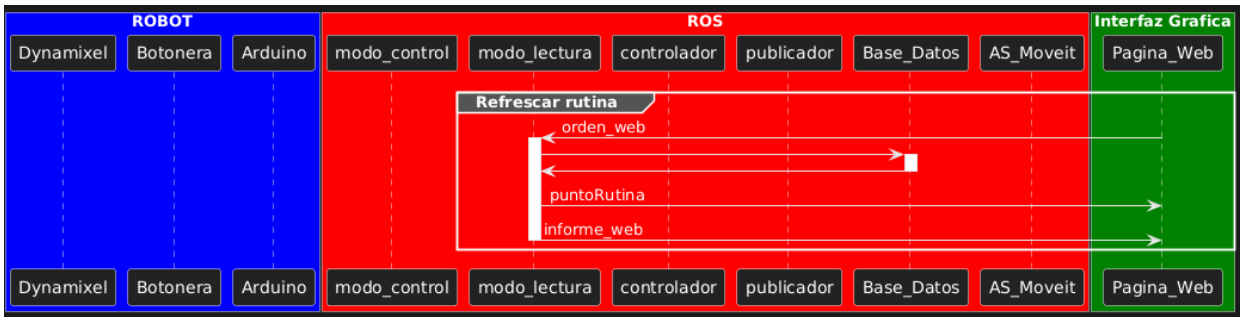

Figura Nº51 DS-Refrescar Rutina actual.

Sección de Coordenadas (3.4.x)

En esta sección se puede usar para probar puntos e ingresarlos a la rutina, por aquí se puede comenzar a programar la rutina desde la interfaz web (Figura Nº52).

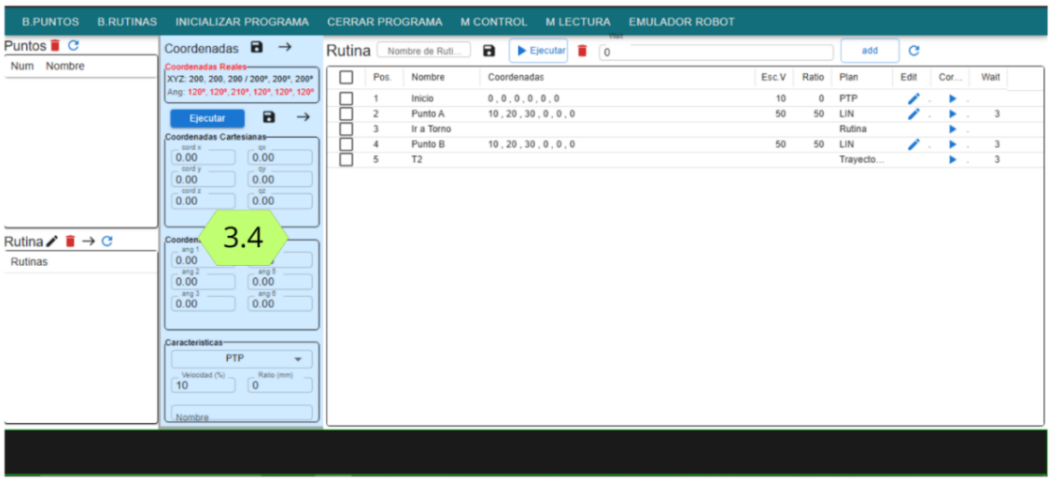

Figura Nº52 Sección de Coordenadas (3.4) en plataforma web.

Coordenadas reales (3.4.3): aquí se visualiza las coordenadas angulares y cartesianas reales que se obtiene de los encoder de los motores (cada 0.5 seg), aquí también se puede visualizar cuando hay algún problema con los motores ya que en las coordenadas angulares mostradas se colocaran en rojo cada elemento del vector respecto al motor que representa si este presenta algún error en el dato enviado indicando que el motor pudo haberse desconectado (Figura Nº53) (Figura Nº54).

Se podrá roja la coordenada angular del vector que este desconectado.

Figura Nº53 Coordenadas reales (3.4.3) con error (izquierda) y sin error (derecha).

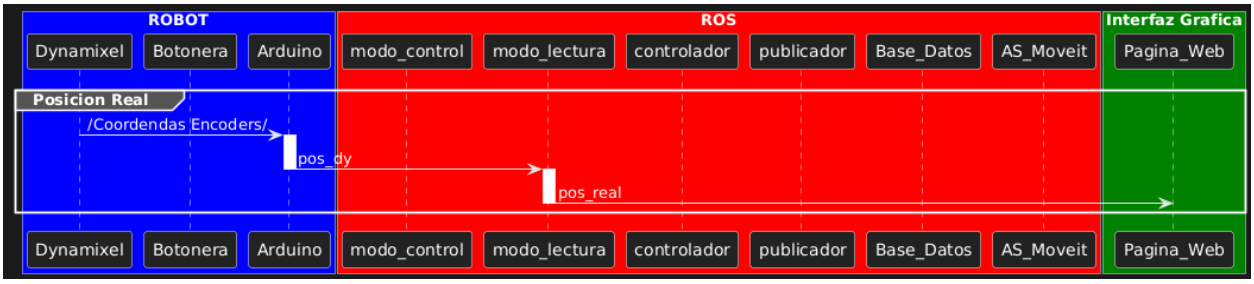

Figura Nº54 DS-Posición real.

Inputs coordenadas cartesianas (3.4.7) y Inputs coordenadas angulares (3.4.8): (Figura Nº55) sirven para ingresar las coordenadas que quiero utilizar, ya sean angulares en grados (cada una representa el ángulo que se moverán los motores, siendo 1 el de la base y 6

el del extremo del robot) o cartesianas (que representa las coordenadas del TCP) en milímetros (x,y,z) y grados (qx, qy, qz).

Cada vez que se sale de la casilla de alguno de los 2 conjuntos se envían los valores de este al controlador para que recalcule los valores del otro conjunto y los devuelva a la página web para que siempre estén sincronizados ambos conjuntos y el usuario tenga una mejor idea de cómo se va a mover el robot en base a sus instrucciones, así si ingresa unas coordenadas angulares, puede visualizar la posición que tendrá el TCP, y viceversa (Figura Nº56).

En caso de ingresar coordenadas angulares, el controlador realiza la transformación mediante cinemática directa, la cual permite obtener de forma determinista la posición cartesiana correspondiente. No obstante, en la implementación práctica pueden presentarse errores asociados a datos inválidos o inconsistencias en los valores ingresados, los cuales son detectados y reportados al usuario.

En cambio, al realizar la conversión de coordenadas cartesianas a articulares, el controlador emplea cinemática inversa, proceso que no siempre encuentra solución, ya sea por limitaciones geométricas, singularidades o restricciones articulares. En estos casos, el sistema informa al usuario que el punto ingresado no es válido para su ejecución.

Cuando se detecta un error, tanto en la conversión de coordenadas angulares a cartesianas como en el proceso inverso, se notifica mediante la **Pantalla de mensajes (3.0)** y no se actualiza el conjunto de coordenadas correspondiente.

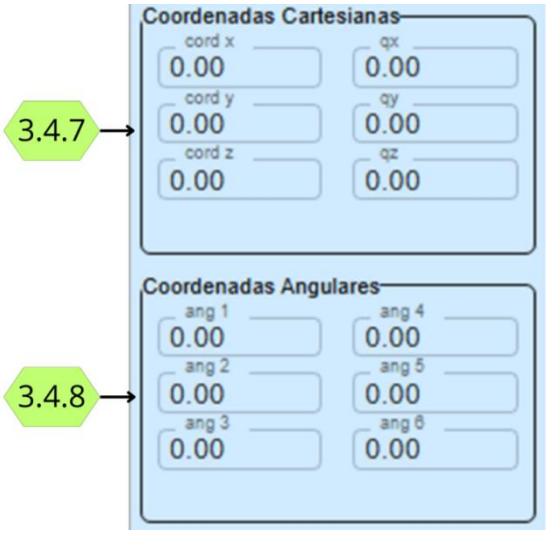

Figura Nº55 Inputs coordenadas cartesianas (3.4.7) e Inputs coordenadas angulares (3.4.8)

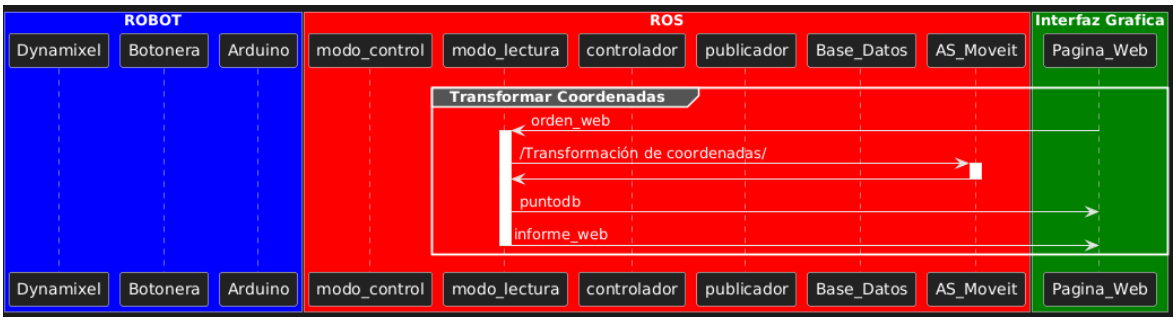

Figura Nº56 DS-Transformar Coordenadas.

Botón guardar punto real (3.4.1): Guarda el punto donde se encuentra el robot real en la base de datos para luego ser usado de forma rápida, es muy útil para puntos que piense repetir en una rutina o puntos de referencia muy utilizados (como un home). Para esto utiliza las coordenadas angulares actuales del robot y si se ha ingresado el **Nombre del punto (3.4.12),** si se ha dejado la casilla vacía el controlador antes de guardarlo le asigna un nombre que no se repite, si el nombre ingresado en la casilla ya existe no se guarda y se informa al usuario. Una vez guardada se podrá visualizar en **Lista de rutina guardada (3.3.5)** (Figura Nº57) (Figura Nº58).

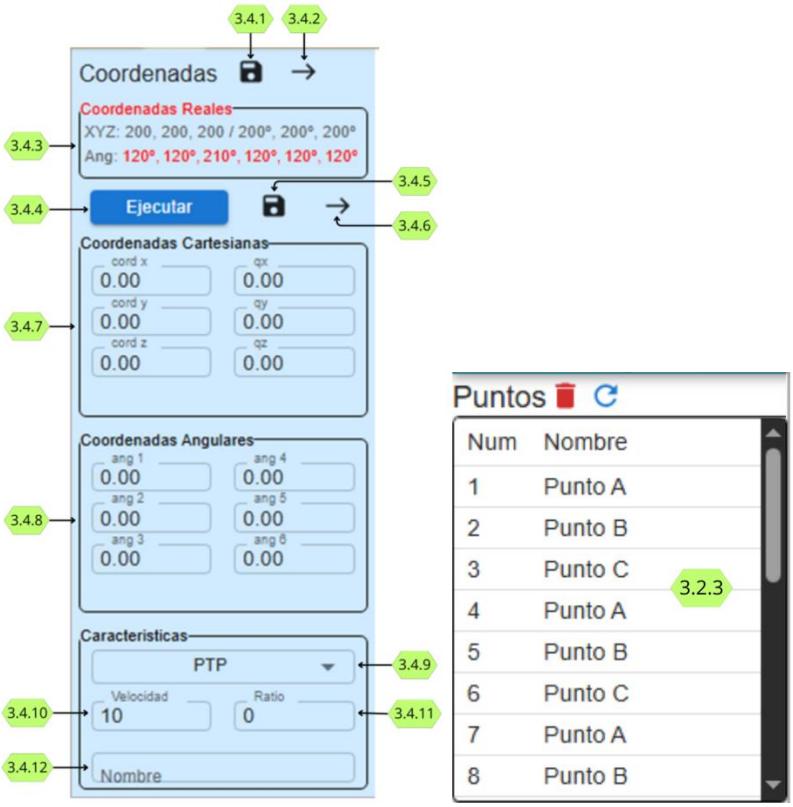

Figura Nº57 Sección de coordenadas (3.4) y Lista de puntos guardados (3.2.3).

Figura Nº58 DS-Guardar punto real.

Botón agregar punto real a la rutina (3.4.2): Agrega a la RA una instrucción del tipo punto al final de esta utilizando todos los datos de **Desplegable tipo de trayectoria (3.4.9)**, **Velocidad del punto (3.4.10) porcentual (max:2rad/seg)**, **Precisión del punto (3.4.11)** en milímetro y **Nombre del punto (3.4.12)**, si se ha dejado la casilla vacía el controlador antes de guardarlo le asigna un nombre. Aquí también utiliza las coordenadas angulares reales del robot, pero para transformarla y luego guardarla en la RA, realiza la transformación a pose, que es la más exacta para que ROS la comprenda y pueda correr cualquier tipo de plan, y luego realiza la transformación a cartesiana que es la que el usuario más comprende (Figura Nº57) (Figura Nº59).

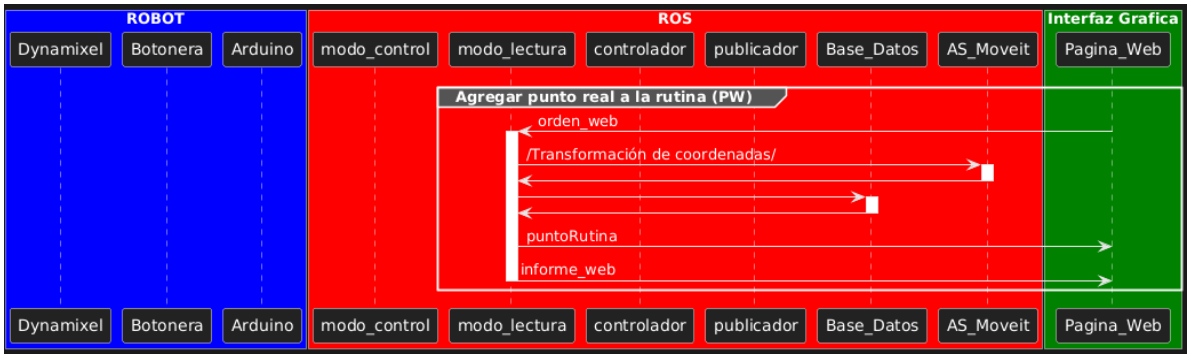

Figura Nº59 DS-Agregar punto real a la rutina desde plataforma web.

Botón guardar punto (3.4.5): funciona igual que **Botón guardar punto real (3.4.1)** salvo que utiliza las coordenadas de **Inputs coordenadas angulares (3.4.8)** (Figura Nº57) (Figura Nº60).

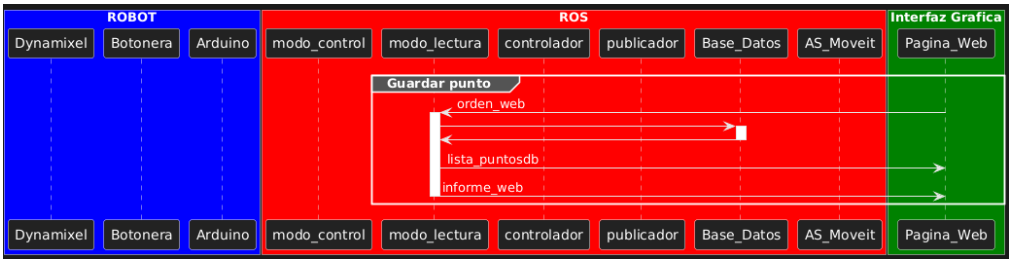

Figura Nº60 DS-Guardar punto.

Botón agregar punto a la rutina (3.4.6): funciona igual que **Botón agregar punto real a la rutina (3.4.2)** salvo que utiliza las coordenadas de **Inputs coordenadas angulares (3.4.8)** (Figura Nº57) (Figura Nº)61.

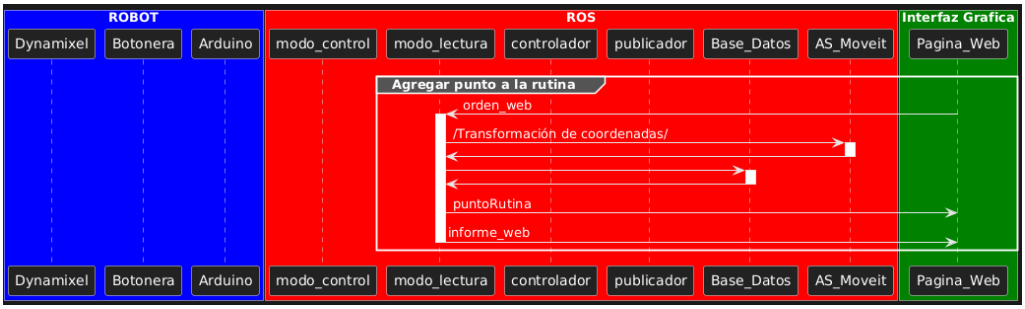

Figura Nº61 Agregar punto a la rutina actual.

Botón ejecutar punto (3.4.4): Envía la orden al controlador de mover el robot al punto ingresado utilizando los datos **Inputs coordenadas angulares (3.4.8)**, **Desplegable tipo de trayectoria (3.4.9)**, **Velocidad del punto (3.4.10)** porcentual(max:2rad/seg), **Precisión del punto (3.4.11).** Si todo esta correcto el robot se mueve adecuadamente, sino se informa al usuario en Pantalla de mensajes (3.0). Este botón no aparece cuando se está en “modo lectura” (Figura Nº57) (Figura Nº62).

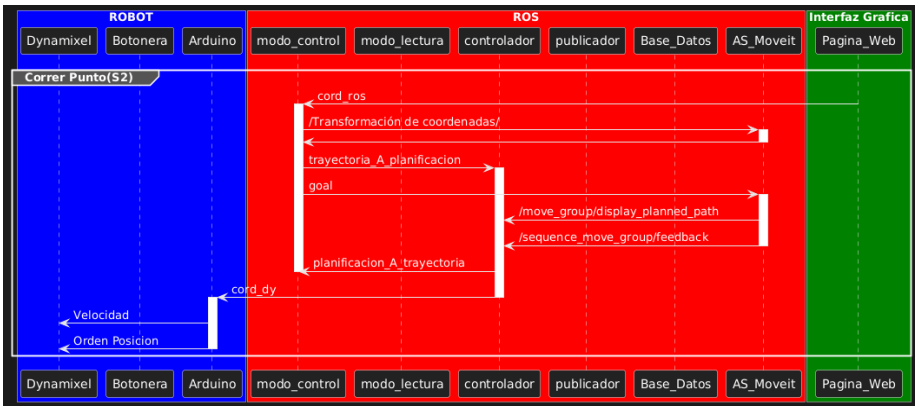

Figura Nº62 DS-Correr punto de Sección de coordenadas (3.4).

Sección de Puntos Guardados (3.2.x):

Aquí se muestran y manipulan los puntos guardados en la base de datos, que internamente se guardan en una tabla de esta (Figura Nº63).

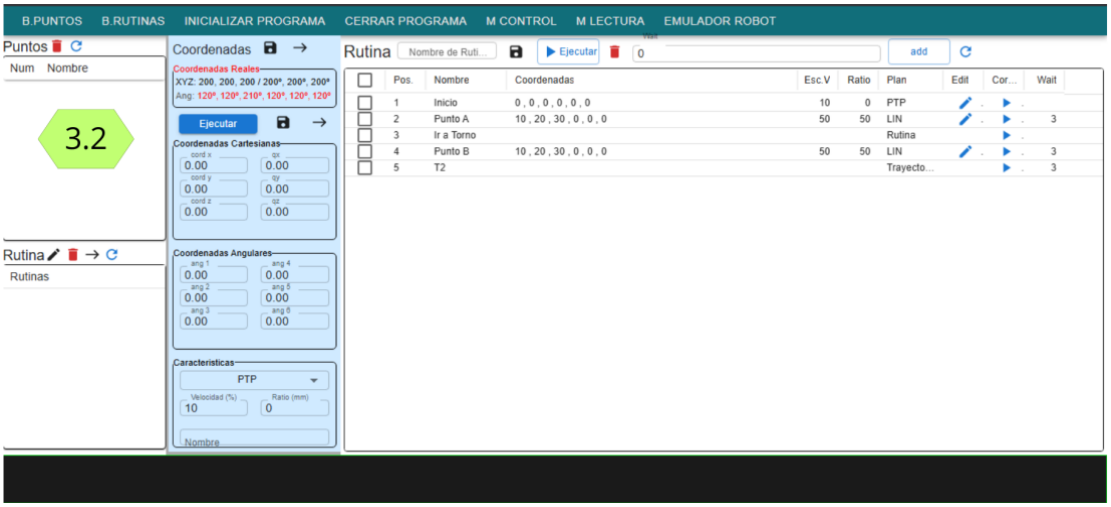

Figura Nº63 Sección de Puntos Guardados (3.2) en plataforma web.

Lista de puntos guardados (3.2.3): aquí se muestran los puntos guardados en la base de datos. En esta el usuario puede cliquear cualquiera y los datos de este se mostraran en los **Inputs coordenadas cartesianas (3.4.7) y Inputs coordenadas angulares (3.4.8)** y **Nombre del punto (3.4.12)** (Figura Nº64) (Figura Nº65).

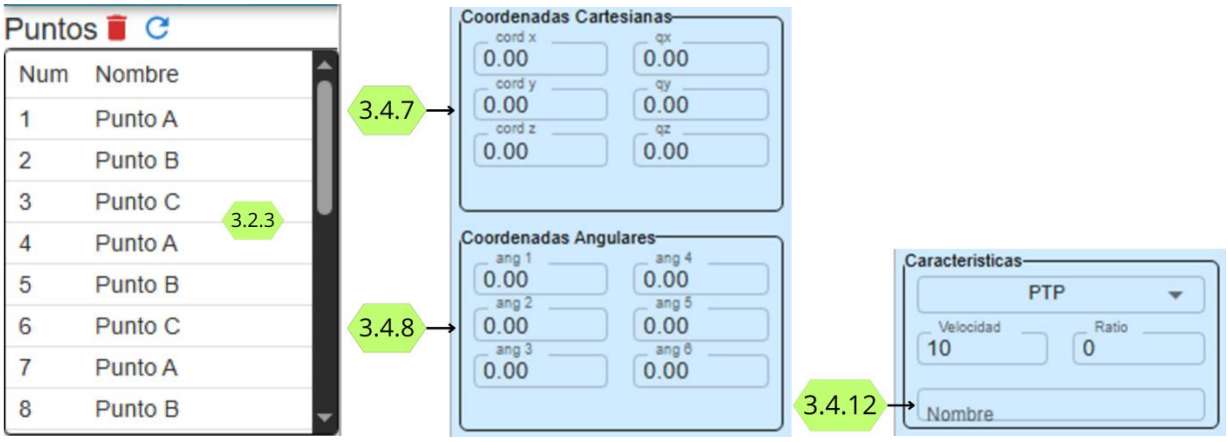

Figura Nº64. Lista de puntos guardados (3.2.3), Inputs coordenadas cartesianas (3.4.7), Inputs coordenadas angulares (3.4.8) y Nombre del punto (3.4.12)

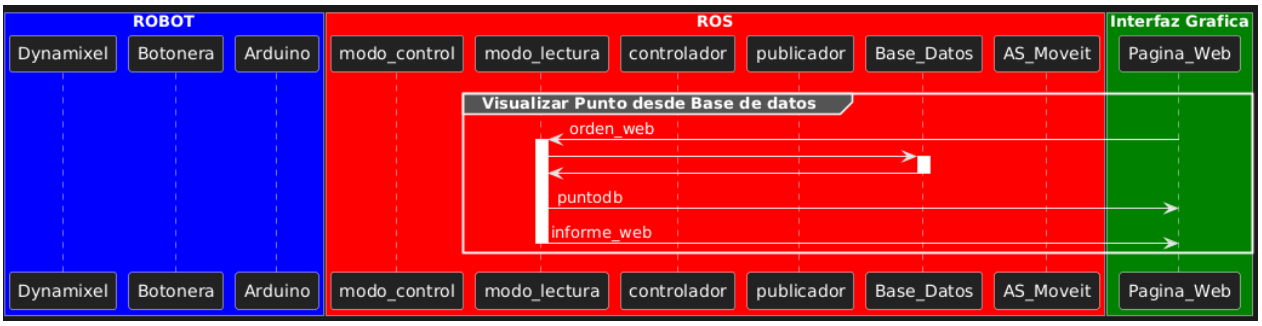

Figura Nº65 DS-Visualizar puntos guardados.

Botón eliminar punto guardado (3.2.1): manda la orden al controlador de borrar de la base de datos el punto seleccionado previamente (Figura Nº66) (Figura Nº68), si este no existe o genera un error se informará al usuario por medio de una ventana emergente.(Figura Nº67).

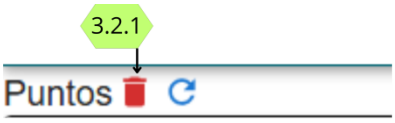

Figura Nº66 Botón eliminar punto guardado (3.2.1).

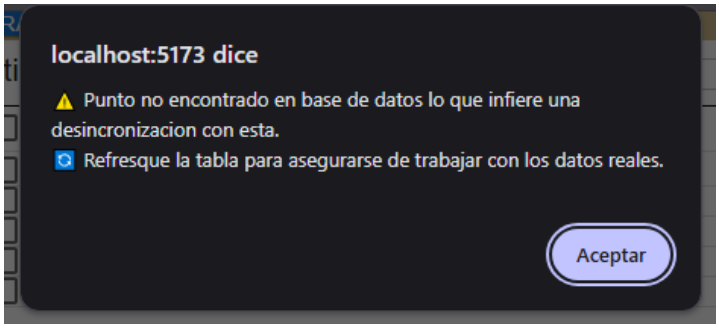

Figura Nº67 Ventana emergente por error al eliminar punto guardado.

Figura Nº68 DS- Eliminar punto guardado de base de datos.

Botón refrescar lista de puntos guardados (3.2.2): sincroniza los datos con los de la base de datos por si ha habido un error y la plataforma web no está mostrando los datos reales (Figura Nº69) (Figura Nº70).

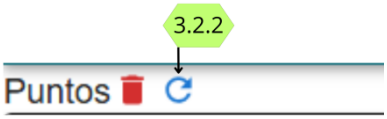

Figura Nº69 Botón refrescar lista de puntos guardados (3.2.2).

Figura Nº70 DS-Refrescar lista de puntos guardados.

Sección de Rutinas Guardadas (3.3.x):

Aquí se muestran y manipulan las rutinas guardados en la base de datos, que internamente se guardan cada una en una tabla diferente (Figura Nº71).

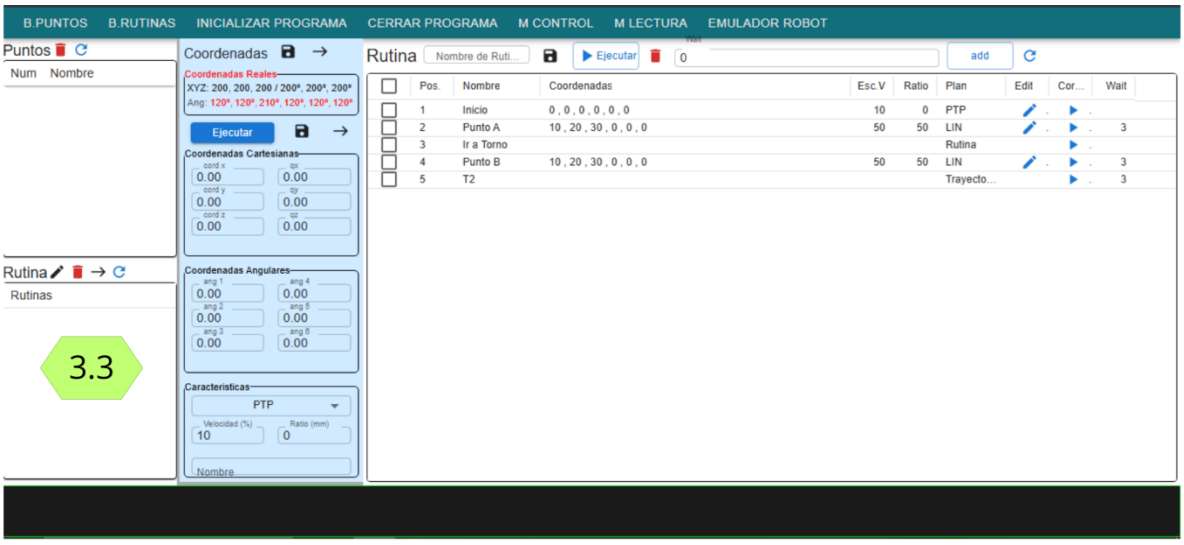

Figura Nº71 Sección de Rutinas Guardadas (3.3) en plataforma web.

Lista de rutina guardadas (3.3.5): aquí se muestra las rutinas guardadas en la base de datos, al cliquear una se selecciona para luego poder hacer diferentes acciones, en cada acción que se explica a continuación se verifica si la rutina seleccionada existe en base de datos y si no es así informa en Pantalla de mensajes (3.0) así el usuario no está trabajando con información adecuada (Figura Nº72).

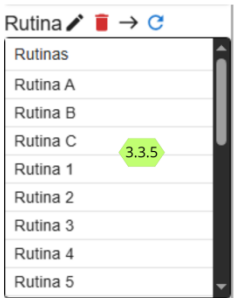

Figura Nº72 Lista de rutina guardadas (3.3.5).

Botón Editar Rutina (3.3.1): sirve para copiar la rutina seleccionada previamente en la RA para poder correrla, modificarla o programar con esta como base, al copiarla en RA no se esta trabajando sobre la rutina seleccionada puntualmente lo que permite probar y modificarla sin riesgo a cambiar la rutina guardada hasta que no se guarde. Lo que se encontraba en RA se elimina y rellena con la anterior por lo que se pierde si no fue guardado (Figura Nº73) (Figura Nº78).

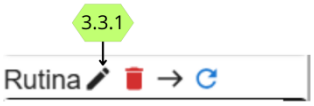

Figura Nº73 Botón Editar Rutina (3.3.1).

Botón eliminar rutina guardada (3.3.2): elimina la rutina seleccionada previamente (Figura Nº74) (Figura Nº78), antes de esto la interfaz gráfica pregunta por medio de una ventana emergente (Figura Nº75) para evitar borrar elementos por error.

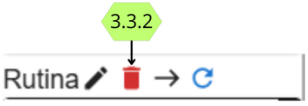

Figura Nº74 Botón eliminar rutina guardada (3.3.2).

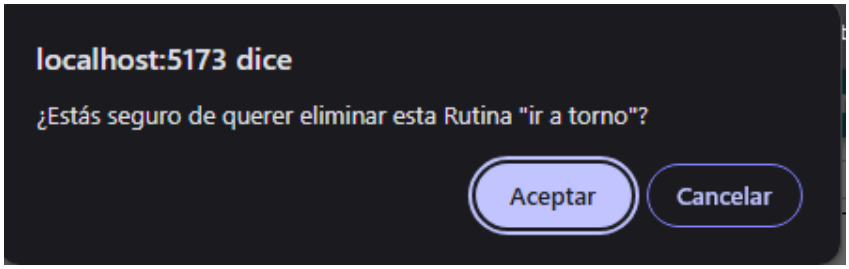

Figura Nº75 Ventana emergente para confirmar eliminación de rutina.

Botón agregar rutina guardada a la rutina (3.3.3): agrega a la RA la rutina como una instrucción (Figura Nº76) (Figura Nº78).

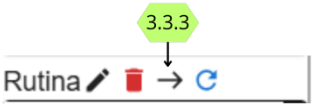

Figura Nº76 Botón agregar rutina guardada a la rutina (3.3.3).

Botón refrescar lista de rutinas guardadas (3.3.4): Refresca la lista de rutina con las rutinas de la base de datos por si ha habido alguna error y los datos mostrados no son los correctos (Figura Nº77) (Figura Nº78).

Figura Nº77 Botón refrescar lista de rutinas guardadas (3.3.4).

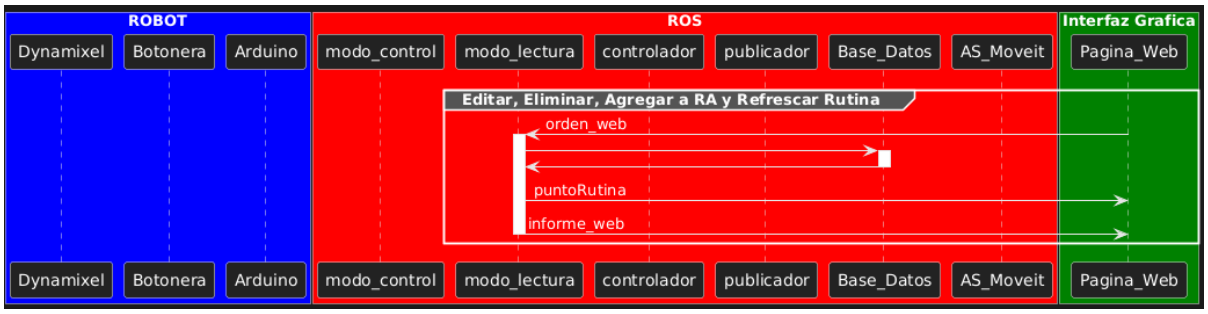

Figura Nº78 DS-Editar, Agregar a rutina actual y Refrescar Rutinas guardadas.

Barra de herramientas (3.1.x):

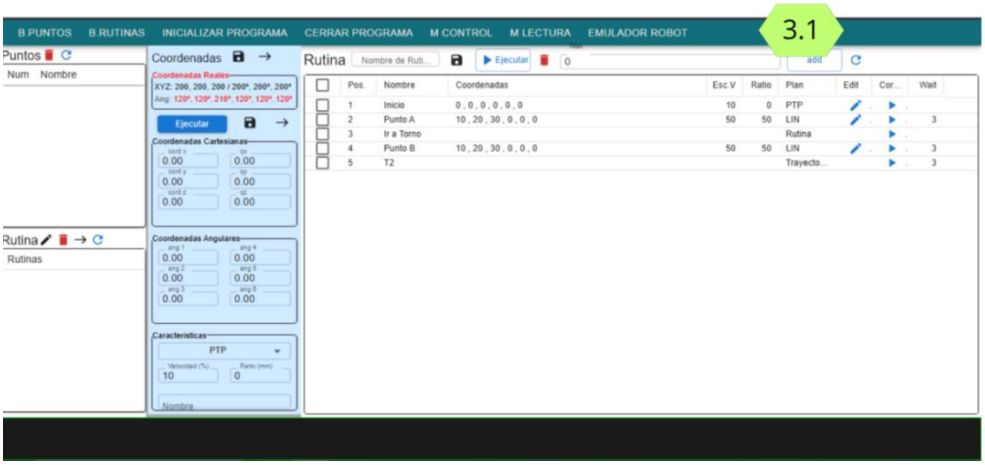

Figura Nº79 Barra de herramientas (3.1) en plataforma web.

Botón de puntos guardados (3.1.1): permite mostrar y ocultar la **Sección de Puntos Guardados (3.2.x)** (Figura Nº80) (Figura Nº81).

Figura Nº80 Botón de puntos guardados (3.1.1)

Figura Nº81 DS-Desplegar u ocultar barra de Puntos guardados

Botón de Rutinas guardadas (3.1.2): permite mostrar y ocultar la **Sección de Rutinas Guardadas (3.3.x)** (Figura Nº82) (Figura Nº83).

Figura Nº82 Botón de Rutinas guardadas (3.1.2)

Figura Nº83 DS-Desplegar u ocultar barra de Rutinas guardadas.

Botón de Inicialización de programa (3.1.3): inicializa el controlador en ROS. Esto se pensó principalmente para cuando se migre el controlador a la Raspberry Pi, para no tener que conectar una computadora a esta e iniciarlo por comando, se configuro el sistema operativo para que pueda recibir órdenes de la plataforma web y se realizó un programa de python que al recibir la orden al cliquear este botón se inicialice el controlador (Figura Nº83).

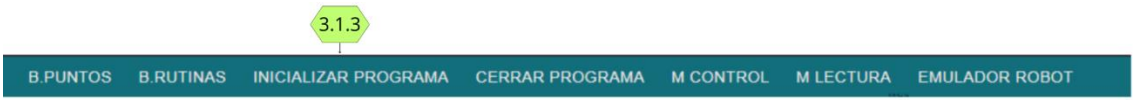

Figura Nº84 Botón de Inicialización de programa (3.1.3)

Botón de Cierre de programa (3.1.4): cierra el controlador, por si hay que reiniciarlo o simplemente cerrarlo, esto de la misma forma y el mismo criterio que los explicador en el elemento anterior (Figura Nº85).

Figura Nº85 Botón de Cierre de programa (3.1.4)

Botón de Modo Control PW (3.1.5): pone el robot en modo control se puede visualizar por el cambio de color (azul) en la barra de tarea plataforma web (Figura Nº86) (Figura Nº87), y confirmar al encenderse el **Led Modo Control (1.5)**.

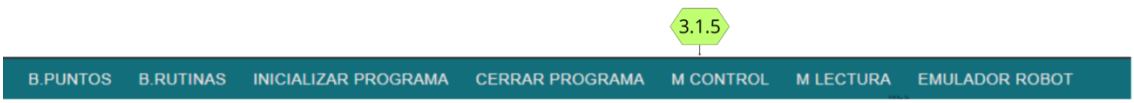

Figura Nº86 Botón de Modo Control PW (3.1.5)

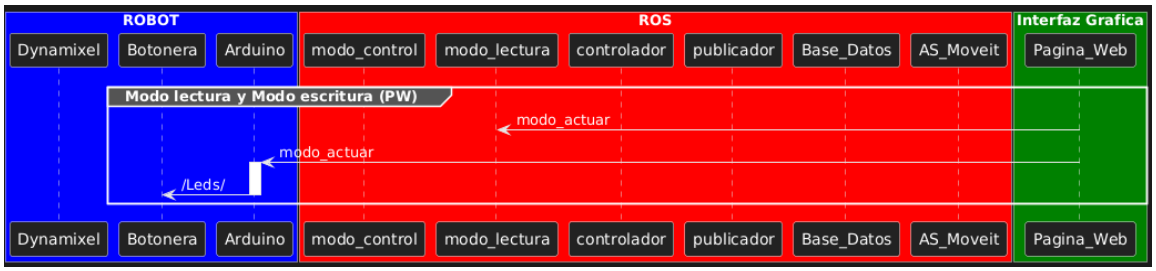

Figura Nº87 DS-Activar modo lectura y modo escritura desde plataforma web

Botón de Modo Lectura PW (3.1.6): pone el robot en modo Lectura se puede visualizar por el cambio de color (verde) en la barra de tarea plataforma web (Figura Nº88) (Figura Nº87), y confirmar al encenderse el **Led Modo Lectura (1.6)**.

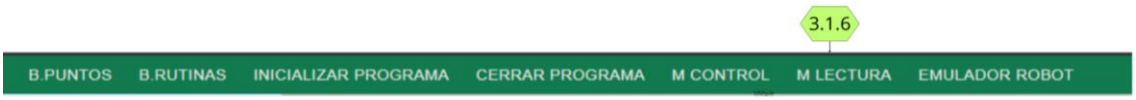

Figura Nº88 Botón de Modo Lectura PW (3.1.6)

Botonera de control (1.x)

Botón Modo Control B (1.1): pone el robot en modo control se puede encendiendo **Led Modo Control (1.5)** (Figura Nº89) (Figura Nº90)**.** El cambio también se visualiza en la página web.

Figura Nº89 Botón Modo Control B (1.1) y Led Modo Control (1.5)

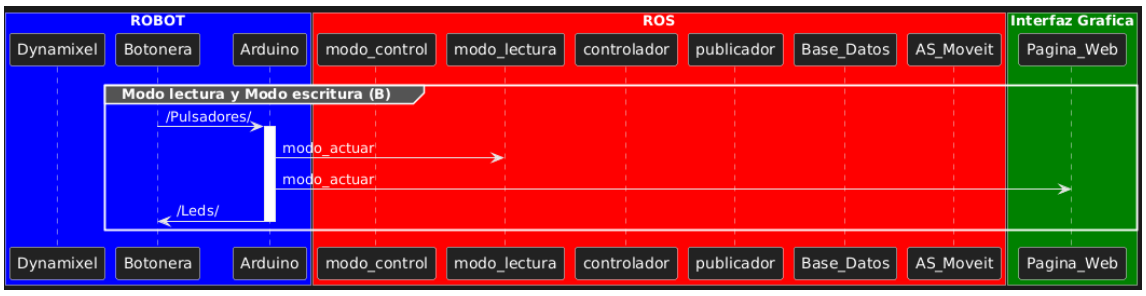

Figura Nº90 DS- Activar modo lectura y modo escritura desde Botonera

Botón Modo Lectura B (1.2): pone el robot en modo Lectura se puede encendiendo **Led Modo Lectura (1.6)** (Figura Nº91) (Figura Nº90)**.** El cambio también se visualiza en la página web.

Figura Nº91 Botón Modo Lectura B (1.2) y Led Modo Lectura (1.6)

Botón Ejecutar Rutina (1.3): manda la orden al controlador para que ejecute la RA, se visualiza que la orden fue enviada al encenderse el **Led Ejecutar Rutina (1.7)** temporalmente (Figura Nº92) (Figura Nº93).

Figura Nº92 Botón Ejecutar Rutina (1.3) y Led Ejecutar Rutina (1.7)

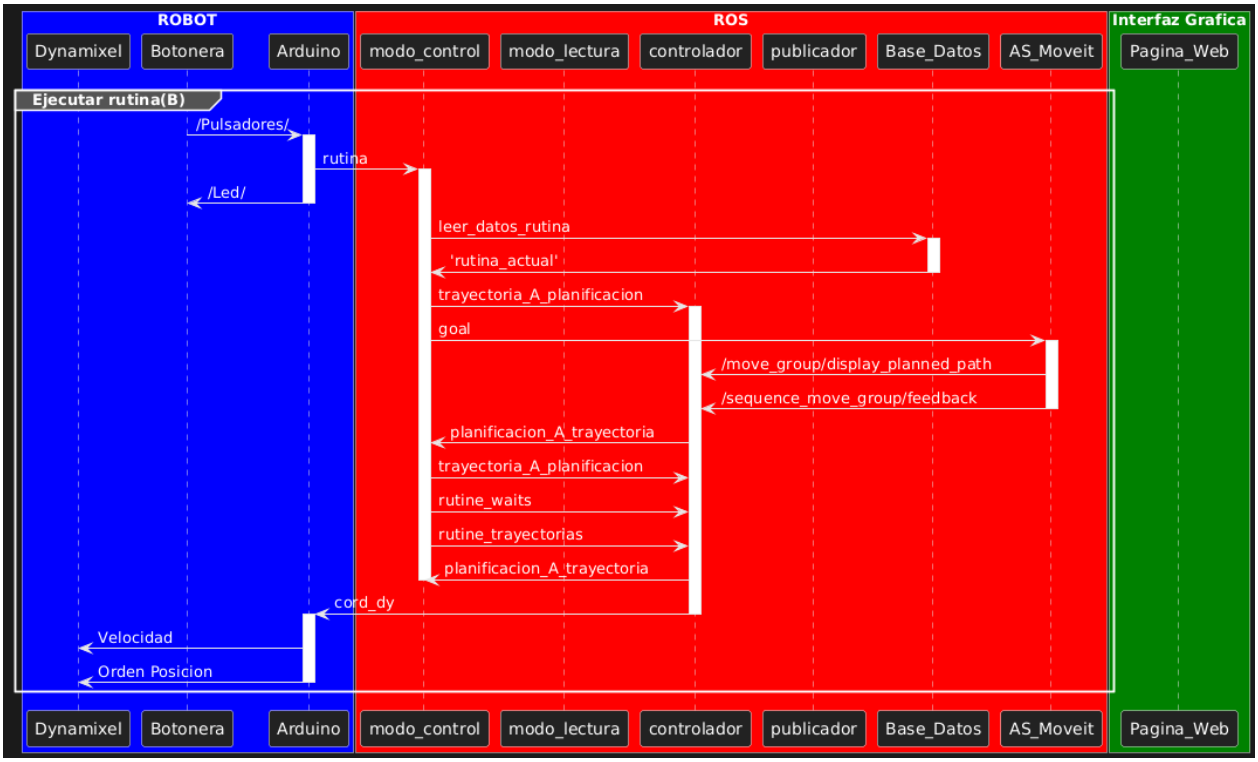

Figura Nº93 DS-Ejecutar rutina actual activada desde Botonera

Botón Sub-Modos (1.4): al apretarlo cambia de sub-modo, de punto a trayectoria y viceversa encendiendo y apagando los **Led Sub-Modo Punto (1.8)** (Figura Nº94) **y Led Sub-Modo Trayectoria (1.9)** (Figura Nº95) según corresponda (Figura Nº96).

Este botón también sirve para borrar la trayectoria que se estaba almacenando en controlador antes de que se guarde en la base de datos.

Figura Nº94 Botón Sub-Modos (1.4) y Led Sub-Modo Punto (1.8)

Figura Nº95 Botón Sub-Modos (1.4) y Led Sub-Modo Trayectoria (1.9)

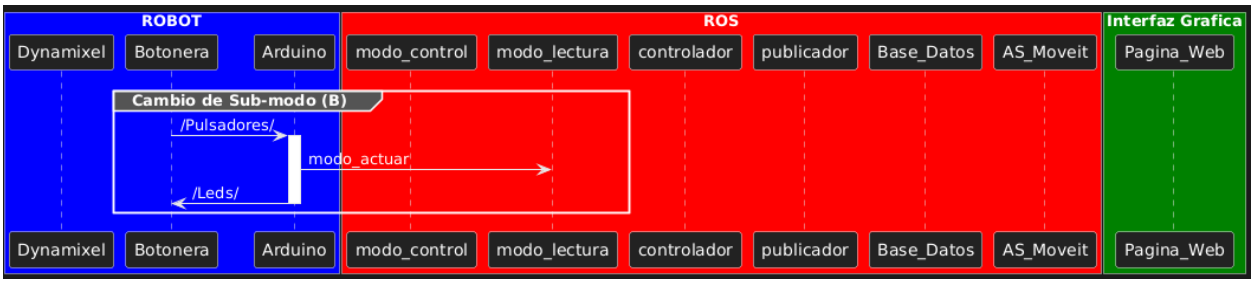

Figura Nº96 DS-Cambio de sub-modos

Botonera de Lectura (2.x)

Todos los elementos de esta placa solo reaccionan en "modo lectura"

Botón de Desenclavar (2.1): mientras se mantenga apretado desenclava los motores permitiendo que el usuario pueda mover el robot con la mano, si se está en el sub-modo trayectoria también envía al controlador las posiciones de los motores cada 0.05 segundos para que los almacene hasta que sean guardados en la base de datos al pulsar el **Botón de Guardado (2.2)** o eliminados al presionar **Botón Sub-Modos (1.4)** (de esta forma puedo eliminar la trayectoria que no ha sido del agrado antes de ser guardada al cambiar de sub modo y luego apretando de vuelta para volver al sub-modo trayectoria) (Figura Nº97) (Figura Nº98).

Mientras se mantenga apretado enciende el **Led de Desenclavar (2.3)**

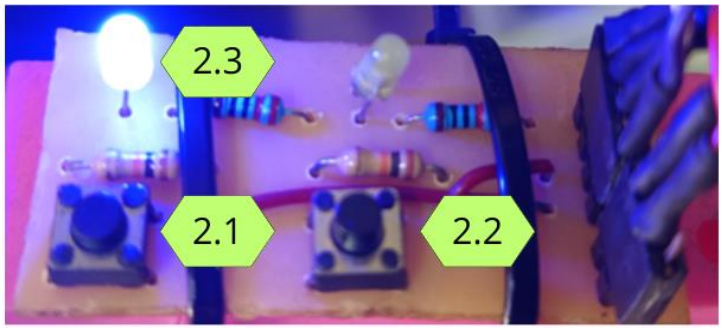

Figura Nº97 Botón de Desenclavar (2.1), Botón de Guardado (2.2) y Led de Desenclavar (2.3).

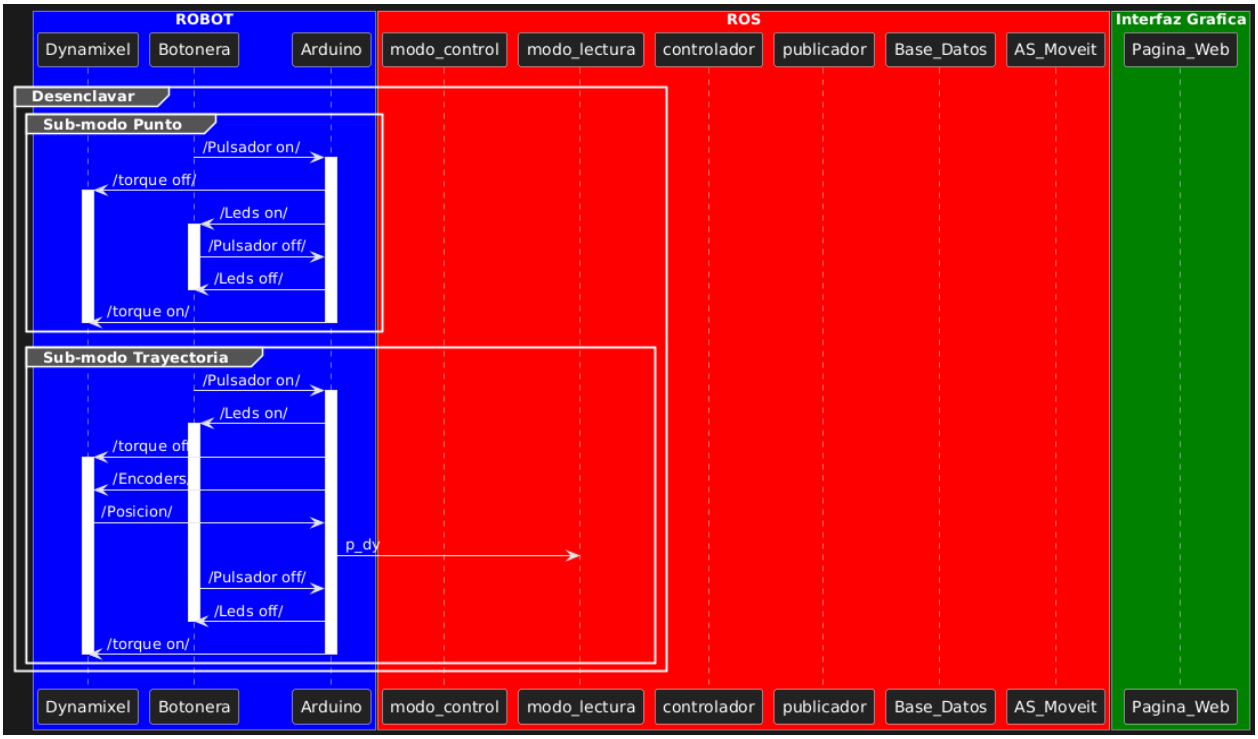

Figura Nº98 DS-Desenclavar motores de acuerdo al sub-modo.

Botón de Guardado (2.2): sirve para guardar el dato adecuado en la base de datos al ser apretado, si se esta en el sub-modo punto se guarda la posición actual del robot, si se esta en el sub-modo trayectoria ordena guardar la trayectoria que se estaba almacenando en el controlador (Figura Nº99) (Figura Nº100).

Al apretarlo enciende el **Led de Guardado (2.4)**

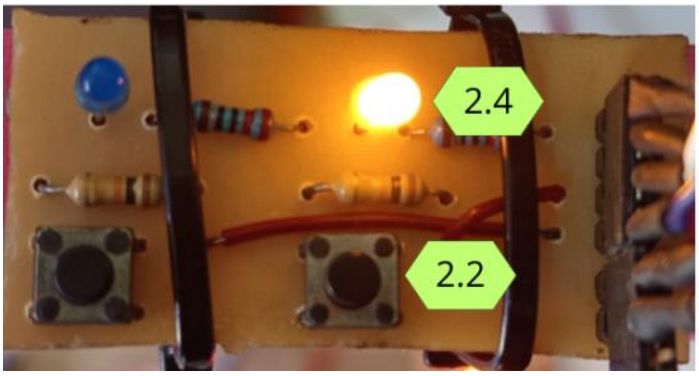

Figura Nº99 Botón de Guardado (2.2) y Led de Guardado (2.4).

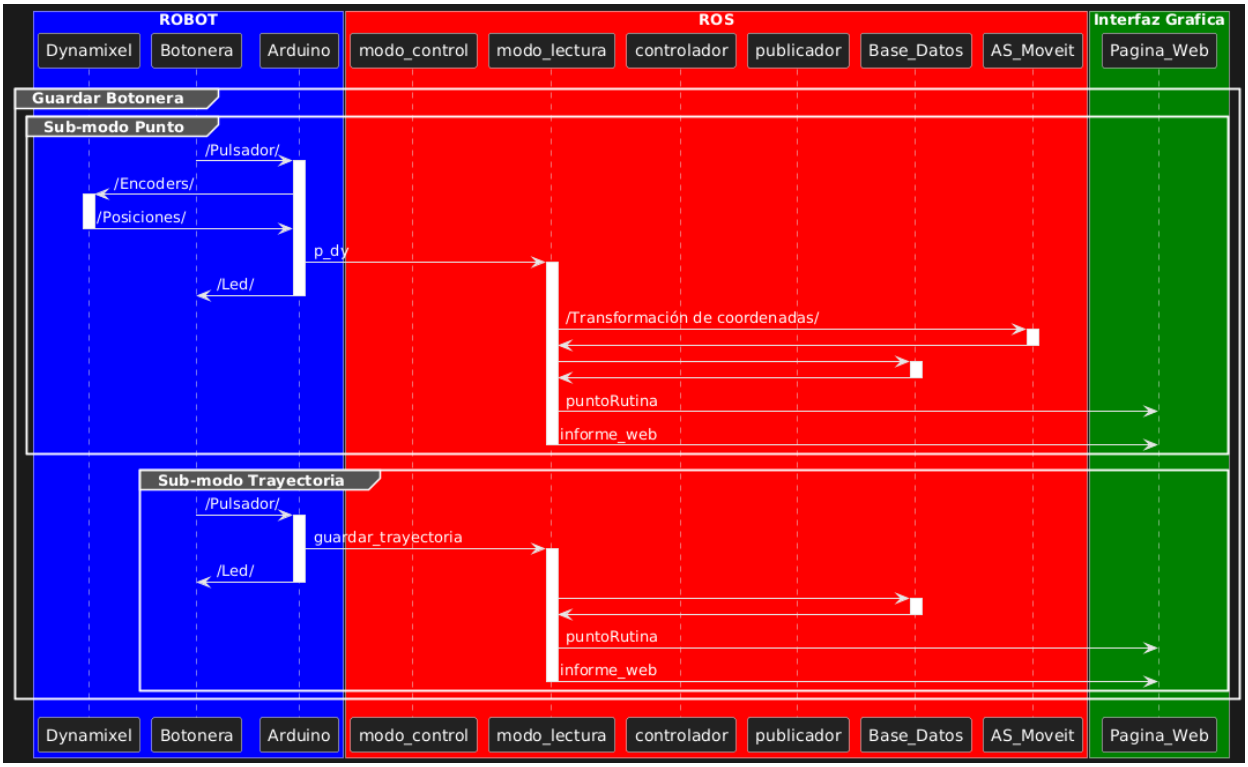

Figura Nº100 Guardado de punto o trayectoria en base al sub-modo.

2.4 Herramientas complementarias para modularidad y pruebas funcionales

En esta sección se presentan una serie de herramientas complementarias que fueron implementadas con el objetivo de facilitar futuras modificaciones, mejoras o pruebas del sistema. Estos elementos aportan mayor modularidad, flexibilidad y facilidad de mantenimiento, resultando especialmente útiles en etapas de desarrollo, validación o adaptación del sistema a otros robots.

2.4.1 Conversión de unidades (Conversion_Robot.py):

Se desarrolló un módulo específico que funciona como un servicio para el controlador, encargado de realizar la conversión de unidades entre los valores provenientes del robot y aquellos utilizados por ROS y la interfaz de usuario. Este módulo actúa como un traductor de coordenadas entre el hardware del robot y el sistema de control, permitiendo convertir los datos a unidades normalizadas como grados y radianes, y viceversa.

La implementación de este módulo se realizó con el objetivo de centralizar las ecuaciones de conversión en un único archivo, de modo que ante un eventual cambio de robot, o la modificación de sensores y actuadores del sistema existente, no sea necesario alterar múltiples secciones del código. En estos casos, basta con actualizar el archivo Conversion_Robot.py con las nuevas relaciones de conversión, preservando el resto de la arquitectura del controlador.

2.4.2 Emulador de robot

Durante el desarrollo del proyecto se presentaron situaciones en las que no fue posible ir físicamente al laboratorio, así como también la necesidad de realizar pruebas en un entorno ideal para aislar posibles fallas del sistema. Por este motivo, se implementó un emulador de robot integrado en un enlace adicional de la interfaz web, dentro del mismo proyecto (Figura Nº101).

Este emulador reproduce el comportamiento del robot desde la perspectiva de ROS, el cual solo interpreta tópicos y nodos La aplicación se suscribe y publica en los mismos tópicos que el robot real, permitiendo al usuario simular la información que entregarían los encoders y las posiciones asociadas al guardado de puntos. Cabe aclarar que este emulador no implementa el sub-modo trayectoria.

Adicionalmente, la interfaz permite visualizar los datos que el controlador envía al robot, facilitando la verificación del correcto funcionamiento de la comunicación. Para mejorar la comprensión por parte del usuario, cada campo de entrada muestra también su valor equivalente en grados.

No se considera necesario detallar el funcionamiento interno del emulador, ya que la interfaz indica claramente la función de cada columna y botón, así como el tópico asociado a

cada acción. Para una mejor comprensión de la comunicación utilizada, puede consultarse la Tabla Nº 4: Tópicos, incluida en el Anexo 2

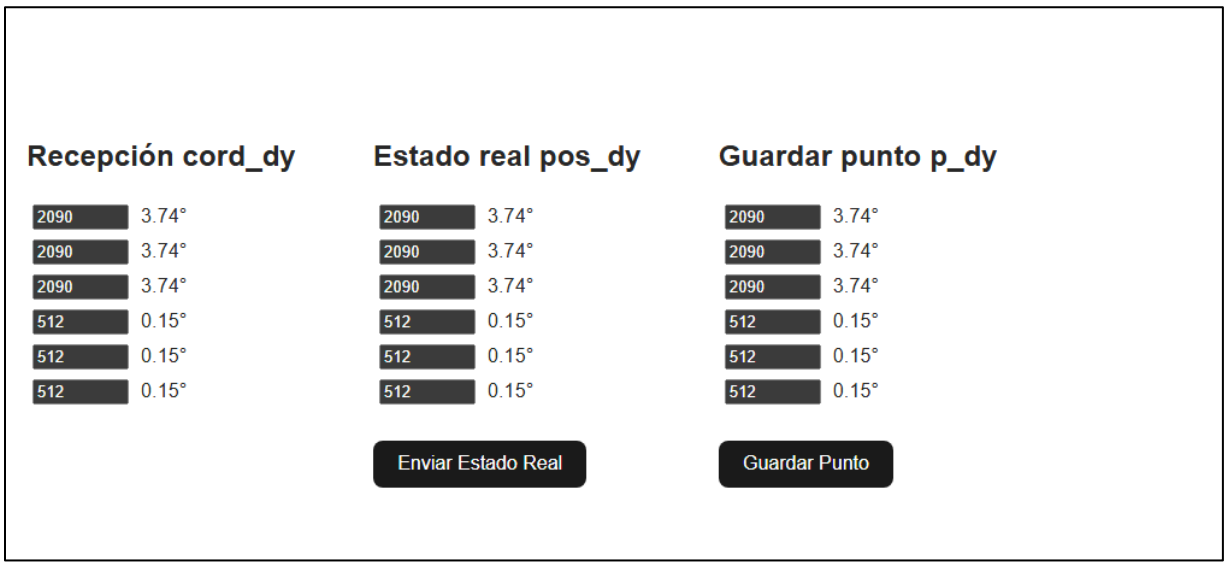

Figura Nº101 Paina web emulador del cobot.

Cuando el controlador se ejecuta desde una computadora, se inicia automáticamente la herramienta RViz, la cual permite visualizar un modelo virtual del robot que está siendo controlado, mostrando en tiempo real los movimientos ejecutados (Figura Nº102).

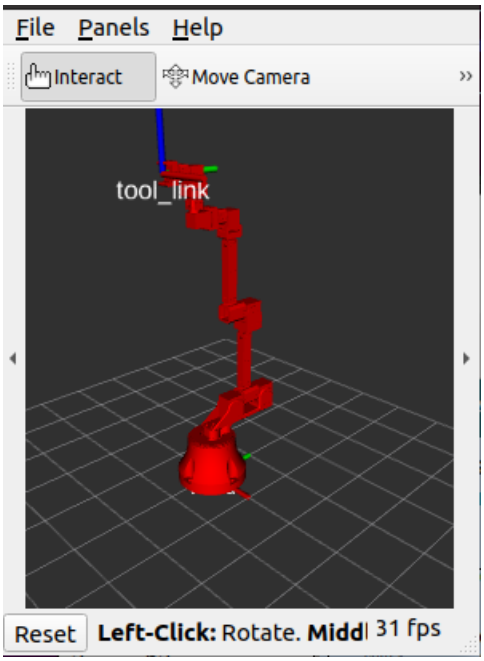

Figura Nº102 RViz.

Si bien el emulador fue desarrollado inicialmente como una herramienta de prueba durante el proceso de implementación, se considera que resulta especialmente útil para los usuarios del sistema, en particular para estudiantes. Esta herramienta permite diseñar y probar robots de seis grados de libertad sin necesidad de contar con el hardware físico, siempre que se genere correctamente el archivo .xacro correspondiente e integrarlo mediante el asistente de MoveIt. Esta tarea puede realizarse con los conocimientos adquiridos en la asignatura Robótica I y siguiendo las instrucciones provistas en la documentación oficial de MoveIt [19].

Esta herramienta se encuentra integrada dentro de la interfaz web del sistema y es accesible mediante la extensión ***/arduino*** de la URL utilizada o cliqueando en el menú de herramientas el botón “EMULADOR ROBOT” (Figura Nº103).

Figura Nº103 Botón de Emulador Robot PW (3.1.7).

2.5 Evaluación Cuantitativa del Controlador

Con el objetivo de caracterizar el desempeño del controlador desarrollado, se realizó un análisis cuantitativo centrado en tres métricas principales: exactitud, precisión y tiempo promedio de respuesta.

El estudio se efectuó exclusivamente sobre el comportamiento computacional del sistema, sin considerar la ejecución sobre el robot físico, ya que el alcance del presente trabajo se limita al desarrollo del controlador e interfaz.

Durante el análisis se identificaron tres fuentes principales de error dentro del procesamiento:

- **Transformación cinemática (Angular → Pose → Angular)**

En la ejecución de rutinas, las coordenadas angulares son convertidas a coordenadas cartesianas (posición + cuaterniones) mediante cinemática directa, y posteriormente resueltas nuevamente a coordenadas angulares mediante cinemática inversa. Este proceso introduce un error asociado a tolerancias numéricas y aproximaciones del solver.

- **Planificación en MoveIt**
    - El planificador puede introducir pequeñas variaciones adicionales debido a discretización y tolerancias internas del algoritmo.

- **Cuantización a unidades Dynamixel**

La conversión de radianes a unidades discretas (0–4095 para los modelo XL430 y 0–1023 para los modelo XL320) introduce un error teórico de resolución (0.088° y 0.293° respectivamente).

Este tercer factor no se incluye en la presente evaluación, ya que corresponde a la etapa posterior al controlador.

2.5.1 Metodología de evaluación

2.5.1.1 Métricas de evaluación

Para cada ejecución se registró el valor angular objetivo θ_{ref} (tomado de la plataforma web) y el valor angular resultante θ_{out} al finalizar el procesamiento del controlador (el valor enviado al Arduino).

A partir de estos datos se definieron las siguientes métricas**:**

Exactitud (Error promedio absoluto)

La exactitud se definió como el valor medio del error absoluto:

$$E_{ang} = \frac{1}{n}\sum_{i=1}^{n}\left|\theta_{ref,i} - \theta_{out,i}\right| \quad (1)$$

Dónde:

- n es el número total de muestras.
- $\theta_{ref,i}$ es el valor angular de referencia.
- $\theta_{out,i}$ valores angular obtenido tras el procesamiento del controlador.

Precisión (Desviación estándar)

La precisión se evaluó mediante la desviación estándar de los errores:

$$\sigma = \frac{1}{n-1}\sum_{i=1}^{n}(\theta_i - \bar{\theta})^2 \quad (2)$$

Dónde:

- θ_i valores angular obtenido.
- $\bar{\theta}$ Media aritmética

Esta métrica permite cuantificar la dispersión de los resultados y, por lo tanto, la repetibilidad del sistema computacional.

Tiempo promedio de respuesta

El tiempo de respuesta se definió como el intervalo entre el envío del comando y la obtención del valor final procesado por el controlador:

$$t_{ang} = \frac{1}{n}\sum_{i=1}^{n} t_i \quad (3)$$

Donde t_i corresponde al tiempo registrado en cada ejecución

2.5.1.2 Metodología de obtención de datos

Para la obtención de los datos para sacar exactitud y precisión, se modificó el nodo "controlador_cobo" con el fin de registrar en la terminal los valores angulares finales generados por el planificador de MoveIt, expresados en grados. Estos valores fueron capturados antes de su conversión a unidades discretas (RAW), evitando así la introducción de errores de cuantización.

Los ensayos se realizaron ingresando configuraciones articulares mediante los Inputs de coordenadas angulares (3.4.8), ejecutando cada punto mediante el botón correspondiente (3.4.4). Para cada ejecución, se registró el valor angular de entrada y el valor resultante obtenido tras el procesamiento cinemático y la planificación de trayectoria, permitiendo así evaluar la precisión y exactitud del sistema (Figura Nº104).

Figura Nº104 Identificación de puntos de entrada y salida del sistema en DS

Para la medición del tiempo de procesamiento, se modificó el nodo "controlador_cobo" con el fin de enviar una señal al nodo "modo_control" inmediatamente antes de comenzar el envío de los primeros puntos al robot en cada orden de ejecución. Asimismo, el nodo "modo_control" fue adaptado para iniciar un temporizador al recibir una orden de ejecución (punto o rutina) y detenerlo al recibir dicha señal desde el nodo "controlador_cobot", registrando el tiempo transcurrido en la terminal.

De este modo, se obtiene una estimación del tiempo requerido desde que se emite la orden de ejecución hasta que el robot comienza a moverse, lo que permite evaluar el tiempo de procesamiento del controlador, incluyendo el cálculo de trayectorias y el envío de comandos al robot (Figura Nº105) (Figura Nº106).

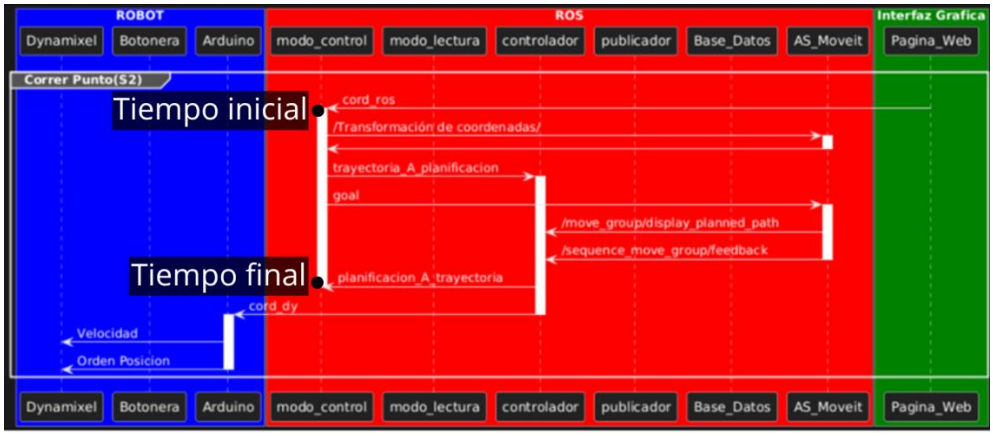

Figura Nº105 Identificación de tiempo inicial y final en DS al ejecutar punto

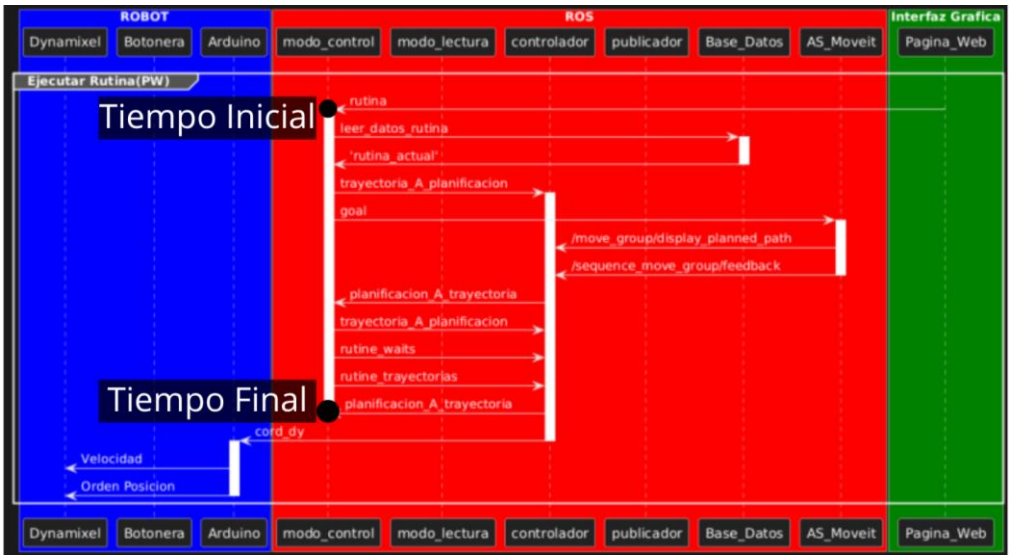

Figura Nº106 Identificación de tiempo inicial y final en DS al ejecutar rutina

2.5.2 Condiciones de ensayo

Siendo los puntos:

- A: (0, 0, 0, 0, 0, 0) (Figura Nº107)
- B: (77, 55, -17, 90, 4, 100) (Figura Nº108)

- C: (163, 36, 72, -10, -132, 50) (Figura Nº109)

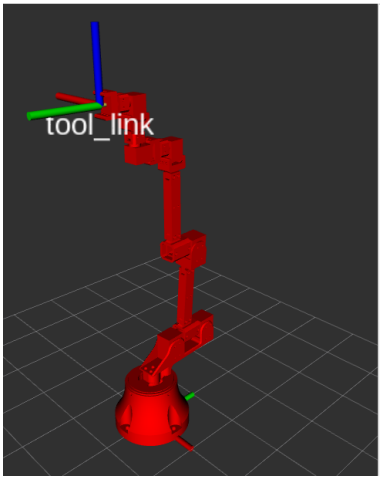

Figura Nº107 Punto A

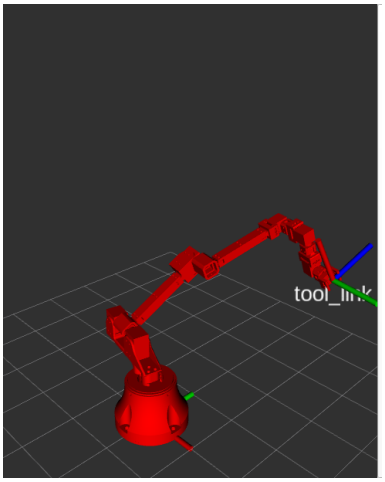

Figura Nº108 Punto B

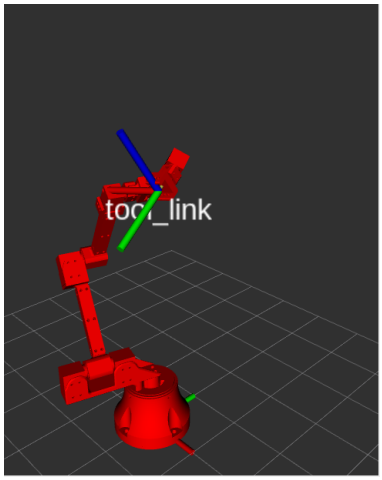

Figura Nº109 Punto C

Los puntos fueron seleccionados con el objetivo de representar distintas configuraciones articulares del robot. El punto A corresponde a la posición de referencia (home), mientras que los puntos B y C fueron definidos con combinaciones de valores positivos y negativos, así como diferentes magnitudes, buscando generar configuraciones variadas dentro del espacio articular. Esta selección permite evaluar el comportamiento del sistema frente a distintos pares de estados iniciales y finales, contemplando tanto desplazamientos cercanos como más alejados.

Las mediciones se realizaron bajo tres escenarios principales:

- Diez ejecuciones desde un punto A hacia un punto B.
- Diez ejecuciones desde el punto B hacia el punto A.
- Diez ejecuciones desde distintos puntos iniciales hacia un mismo punto C.

Cada ejecución generó seis valores angulares correspondientes a las seis articulaciones, obteniéndose un total de 60 muestras por condición.

En las tablas se presenta:

- θ_{refi}: ángulo ingresado.
- θ_{refIK}: ángulo tras la conversión FK+IK.

- E_{angT}: error total del proceso completo.
- E_{angP}: error asociado únicamente a la planificación.
- σ: precisión.

2.5.3 Resultados de exactitud y precisión

a) Condición A → B

Tabla Nº10. Precisión y Exactitud de A hacia B

θ_{refi}	θ_{refIK}	E_{angT}	E_{angP}	σ
77,0000	77,0000	1,1850E-05	1,1850E-05	1,2205E-08
55,0000	51,5000	3,4800E+00	2,0022E-02	7,4472E-06
-17,0000	-23,8900	6,8475E+00	4,2495E-02	1,4658E-05
90,0000	83,7500	6,2086E+00	4,1388E-02	1,4282E-05
4,0000	4,0000	1,3063E-03	1,3063E-03	6,8663E-09
100,0000	97,1400	2,8407E+00	1,9312E-02	7,0707E-06

b) Condición B → A

Tabla Nº11. Precisión y Exactitud de B hacia A

θ_{refi}	θ_{refIK}	E_{angT}	E_{angP}	σ
0,0000	2,3400	2,3409	0,0009	0,0001
0,0000	-0,1900	0,1775	0,0125	0,0000
0,0000	-0,0300	0,0114	0,0414	0,0000
0,0000	0,0900	0,1765	0,0865	0,0000
0,0000	-2,3400	2,3428	0,0028	0,0001
0,0000	-0,0800	0,0116	0,0684	0,0000

c) Condición múltiples → C

Tabla Nº12. Precisión y Exactitud hacia punto C

θ_{refi}	θ_{refIK}	E_{angT}	E_{angP}	σ
163,0000	163,9100	0,9088	0,0012	0,0004
36,0000	36,7500	0,7507	0,0007	0,0013
72,0000	72,1000	0,1183	0,0183	0,0054
-10,0000	-11,0700	1,0184	0,0516	0,0151
-132,0000	-132,0000	0,0022	0,0022	0,0010
50,0000	49,9300	0,0971	0,0271	0,0082

2.5.4 Análisis comparativo

A partir de los resultados obtenidos se observó lo siguiente:

Precisión (σ)

Las variaciones registradas en las tres condiciones son extremadamente bajas, evidenciando comportamiento determinístico del sistema. No se observaron diferencias significativas entre articulaciones ni entre condiciones de ensayo. Esto indica que el controlador presenta alta repetibilidad computacional.

Error total (E_{angT})

En la condición A → B, ciertas articulaciones presentan mayor error promedio. Sin embargo, en la condición B → A, el comportamiento se invierte, siendo otras articulaciones las que muestran mayor desviación.

Este patrón indica que el error no está asociado a una articulación específica, sino al proceso de aproximación numérica introducido por la transformación cinemática.

En la condición múltiples valores → C, los errores totales son considerablemente menores y homogéneos entre articulaciones, lo que refuerza la idea de que el error no depende de una articulación en particular, sino de la relación entre la configuración inicial y final del sistema. Esto sugiere que los mayores errores se producen cuando el solver debe resolver configuraciones específicas entre pares de puntos determinados

Error de planificación (E_{angP})

No se observaron diferencias sistemáticas entre articulaciones en ninguna de las tres condiciones. El error introducido por el planificador se mantiene dentro de márgenes reducidos y consistentes.

2.5.5 Resultados de tiempo promedio de respuesta

Las mediciones temporales se realizaron bajo cuatro configuraciones distintas del sistema. Las mediciones y resultados se realizaron en milisegundos

Tabla Nº13. Resultados de Tiempo promedio

$Condicion$	t_{ang}	σ
Rutinas Simples 3 o más Instrucciones	5403,00 ms	97,73

Rutinas Simples 1 o 2 Instrucciones	3177,10 ms	78,47
Rutinas con trayectorias	1353,83 ms	84,65
Puntos	263,21 ms	48,55

2.6 Resultados y Validación del Sistema

Con el objetivo de verificar el correcto funcionamiento del controlador y la interfaz desarrollados, se realizaron distintas pruebas operativas y funcionales sobre el sistema completo, evaluando estabilidad, coherencia de ejecución, comunicación y comportamiento en distintos modos de programación.

2.6.1 Validación funcional de ejecución

El sistema fue ensayado con hasta cinco rutinas independientes, múltiples puntos individuales y una trayectoria adicional, alcanzando un total de 34 indicaciones ejecutadas de manera secuencial al expandir las rutinas. Dichas indicaciones incluyeron combinaciones de movimientos punto a punto (PTP), funciones auxiliares y trayectorias internas generadas por el planificador.

Durante las pruebas realizadas:

- No se registraron interrupciones en la ejecución.
- No se detectaron bloqueos del nodo principal.
- No se produjeron pérdidas de sincronización entre etapas del controlador.
- La expansión interna de rutinas se realizó correctamente, respetando el orden programado.

Se verificó el correcto funcionamiento en:

- Movimientos tipo punto (PTP).
- Ejecución de rutinas completas.
- Llamadas a funciones internas.
- Generación y ejecución de trayectorias planificadas.

Esto confirma la estabilidad estructural del controlador frente a secuencias de ejecución extensas y combinadas.

Adicionalmente, durante las pruebas se verificó el comportamiento del robot físico, observándose que este ejecutó correctamente las trayectorias indicadas, sin movimientos erráticos. La ejecución de rutinas se realizó de forma fluida, evidenciando correspondencia entre las órdenes generadas por el sistema y el movimiento efectivo del robot

2.6.2 Cambio de modos de operación

El sistema permite alternar entre distintos modos de programación sin necesidad de reiniciar el entorno ni los nodos activos. Se validó el cambio bidireccional entre los modos de operación (Control y Lectura), comprobando que la transición se realiza sin reinicio del sistema y sin generar inconsistencias en los estados internos, la base de datos ni las variables de ejecución.

Durante las pruebas, el cambio de modo mantuvo la coherencia de los estados internos y no produjo pérdidas de información ni comportamientos inesperados en la ejecución.

Desde el punto de vista didáctico, esta separación de modos facilita la comprensión del flujo de programación por parte del usuario, permitiendo distinguir entre acciones individuales y estructuras de mayor complejidad.

2.6.3 Validación de comunicación ROS–Arduino

La comunicación entre el entorno ROS y el microcontrolador se mantuvo estable durante sesiones prolongadas de prueba.

Se adoptaron como parámetros finales:

- Frecuencia de envío de comandos: 20 Hz.
- Frecuencia de lectura de estados: 2 Hz.

Estas frecuencias fueron seleccionadas como compromiso entre fluidez de actualización, estabilidad del sistema y carga de procesamiento.

No se detectaron pérdidas de mensajes ni desconexiones durante las pruebas realizadas. Asimismo, la latencia observada en la ejecución de comandos se mantuvo dentro de valores compatibles con el funcionamiento previsto del sistema.

2.6.4 Gestión de errores y retroalimentación al usuario

Se implementó un sistema de notificación de errores en la terminal, utilizando codificación por colores para facilitar su identificación:

- Mensajes en color amarillo para advertencias o situaciones no críticas.
- Mensajes en color rojo para errores de mayor severidad que requieren intervención del usuario.

En casos específicos, el sistema complementa la notificación en terminal con ventanas emergentes, permitiendo una identificación inmediata del problema.

Si bien el mecanismo actual cumple su función informativa, se considera que esta sección puede ser optimizada en futuras versiones, especialmente en lo referente a estandarización y clasificación jerárquica de errores.

2.6.5 Funcionalidades implementadas y restricciones actuales

La interfaz incluye opciones para la generación de trayectorias lineales y circulares. Estas funcionalidades se encuentran programadas a nivel del nodo correspondiente; sin embargo, no se encuentran habilitadas en la versión actual del sistema, priorizando la validación y estabilidad del modo PTP y de las rutinas.

La habilitación completa de trayectorias lineales y circulares se plantea como una mejora futura del sistema.

2.6.6 Síntesis de validación

Las pruebas realizadas permiten concluir que el controlador desarrollado

- Ejecuta secuencias extensas sin interrupciones.
- Mantiene comunicación estable entre ROS y el microcontrolador.

- Permite alternar correctamente entre distintos modos de programación.
- Proporciona retroalimentación adecuada ante errores.
- Presenta un comportamiento consistente en la ejecución de movimientos PTP y rutinas.

En consecuencia, se considera que el sistema cumple con los objetivos funcionales planteados para el desarrollo del controlador e interfaz del robot colaborativo.

CAPITULO 3: Análisis de Costos

El presente proyecto se desarrolló principalmente a partir del diseño e implementación de un sistema de control e interfaz gráfica para un robot colaborativo, utilizando recursos disponibles en el Laboratorio de Mecatrónica (LABME).

El presente análisis de costos tiene como finalidad identificar los componentes materiales necesarios para la validación y eventual replicación del sistema completo. No constituye un presupuesto comercial ni refleja el valor económico real del desarrollo académico realizado, sino una estimación referencial de los elementos involucrados.

Cabe destacar que parte del equipamiento utilizado —incluyendo la estructura del robot y ciertos actuadores— forma parte del patrimonio del laboratorio. En consecuencia, el presente análisis no representa una inversión efectivamente realizada en el marco del trabajo final, sino una valoración estimativa de los recursos empleados.

A continuación, se detallan los principales recursos físicos y tecnológicos utilizados para el desarrollo y funcionamiento del controlador.

Tabla Nº14. Recursos Físicos empleados en el desarrollo del controlador

Recurso Físico	Descripción/uso
Computadora	Controlador,Servidor de la pagina web, programación, Diseño CAD y documentación.
Impresoras 3D (personal/LABME)	Fabricación de estructura del robot en PLA.
Fuente de alimentación.	Alimentación de los motores.
Multímetro digital	Verificación de conexiones eléctricas y pruebas de continuidad.

Los softwares utilizados en el desarrollo del proyecto son de uso libre y gratuito.

Tabla Nº15. Softwares usados en el desarrollo de la CFFD

Item	Descripción	Costo
Arduino IDE	Programación del controlador de Arduinos.	Gratuito
Onshape	Diseño del contenedor de la botonera en 3D.	Licencia gratuita
Visual Studio Code	Entorno de Programación del Controlador, Pagina Web y modulo de interaccion con Base de datos.	Gratuito
TinyDB	Base de Datos	Gratuito
ROS	Sistema operativo del robot	Gratuito

Si bien el objetivo principal del proyecto es el desarrollo del controlador y la interfaz, fue necesario implementar un robot físico como plataforma de pruebas, por lo que se incluyen los costos asociados a dicho hardware. Los valores expresados en dólares (USD) han sido tomados en referencia al valor del dólar oficial en Argentina, correspondiente a 1442 ARS el día 4/2/2026.

Tabla Nº16. Elementos eléctricos y Electrónicos utilizados.

Fuente	Componente	Cantidad	Costo ($ ARS)	Costo (USD)
[31]	Plaqueta Pcb	1	$3.000,00	2,08 USD
[32]	DYNAMIXEL XL430-W250-T	3	$118.965,00	82,50 USD
[33]	DYNAMIXEL XL-320	3	$133.864,44	92,82 USD
[34]	Placa arduino armega 2560	1	$26.000,00	18,03 USD
[29]	MODULO STEP DOWN LM2596HVS	1	$4.260,00	2,95 USD
TOTAL			$286.071,44	198.39 USD

CAPITULO 4: Estudio de Impacto Ambiental

El presente proyecto, orientado al desarrollo de un sistema de control e interfaz para un robot colaborativo con fines didácticos, presenta un impacto ambiental bajo debido a su carácter experimental y de laboratorio.

El sistema no involucra procesos industriales ni genera residuos significativos durante su operación. Además, el consumo energético es reducido, ya que los componentes utilizados (placas electrónicas y actuadores) operan con baja potencia.

En cuanto a los materiales, se emplean principalmente componentes electrónicos estándar y piezas fabricadas mediante impresión 3D. Si bien este proceso puede generar residuos plásticos, estos son mínimos y se gestionan junto a los residuos de este tipo que se producen en el laboratorio.

Durante su funcionamiento, el sistema no produce emisiones contaminantes, efluentes ni niveles de ruido relevantes, ya que está diseñado para operar en entornos controlados.

Finalmente, el proyecto tiene un impacto positivo indirecto al contribuir a la formación en tecnologías de automatización y robótica. En caso de una futura ampliación, se recomienda considerar criterios de eficiencia energética y gestión adecuada de residuos electrónicos.

CAPITULO 5: Conclusiones

Conclusión general

En el presente trabajo se logró diseñar, desarrollar e implementar un sistema integral de control y una interfaz gráfica para un robot colaborativo de tipo antropomórfico, cumpliendo con los objetivos planteados inicialmente. El sistema desarrollado permite operar el robot en dos modos principales —movimiento y lectura—, integrando de manera efectiva el hardware, el software de control basado en ROS y una interfaz web accesible, logrando una solución funcional, modular y orientada al uso didáctico.

La arquitectura propuesta demostró ser flexible y escalable, facilitando la comunicación entre los distintos componentes del sistema, el manejo de trayectorias y rutinas, y la interacción del usuario con el robot. Asimismo, el uso de herramientas ampliamente adoptadas en el ámbito académico e industrial, como ROS, MoveIt y una interfaz web moderna, refuerza el valor del proyecto como plataforma de aprendizaje y experimentación en entornos universitarios.

Conclusiones en relación con los objetivos específicos

Integración del sistema con la estructura del robot

El sistema de control fue integrado exitosamente con la estructura mecánica del robot disponible en el LABME, realizando el montaje del hardware indispensable para su operación. Las pruebas iniciales confirmaron el correcto funcionamiento conjunto de los subsistemas mecánicos, electrónicos y de control.

Desarrollo de la arquitectura de software

Se diseñó e implementó una arquitectura de software robusta basada en ROS, que permite la comunicación entre los distintos dispositivos, el procesamiento de datos y la coordinación de las acciones del sistema. La modularidad lograda facilita el mantenimiento, la comprensión del código y futuras ampliaciones.

Implementación de los modos de operación

Se desarrollaron los modos de operación “movimiento” y “lectura”, permitiendo registrar posiciones, reproducir trayectorias y ejecutar movimientos definidos por el usuario. Estos modos cumplen con los requisitos funcionales planteados y constituyen una base sólida para la programación del robot.

Selección e implementación de la base de datos

Se seleccionó e implementó una base de datos liviana y adecuada al entorno del proyecto, integrándola al sistema mediante un módulo intermediario que abstrae su uso. Esta decisión permitió simplificar la programación, mejorar la organización de la información y facilitar su modificación y consulta.

Diseño de la interfaz web

Se diseñó y programó una interfaz web que permite interactuar de forma clara y sencilla con el robot, incorporando funciones de programación, control y visualización. La utilización de tecnologías web modernas mejoró la experiencia de usuario y posibilita el acceso desde distintos dispositivos conectados a la red.

Ensamblaje y ajuste del robot

El robot fue ensamblado y ajustado correctamente como plataforma de pruebas, garantizando su funcionamiento mecánico y electrónico. Este paso resultó clave para validar el sistema en condiciones reales de operación.

Pruebas y optimización del sistema

Se realizaron pruebas individuales de cada módulo y pruebas de integración del sistema completo, lo que permitió identificar oportunidades de mejora y optimizar el rendimiento general. Estas pruebas confirmaron la estabilidad y coherencia del sistema desarrollado.

Evaluación del uso didáctico

El sistema fue evaluado en escenarios representativos del uso educativo, mediante la ejecución de rutinas, carga de puntos y utilización del modo lectura, lo que permitió validar su

funcionamiento en condiciones de uso similares a las de un entorno didáctico. En este contexto, se observó que la interfaz resulta intuitiva y de fácil utilización, permitiendo al usuario interactuar con el robot sin necesidad de conocimientos previos avanzados.

CAPITULO 6: Propuestas de Mejora y Trabajos Futuros

Si bien el sistema desarrollado cumple con los objetivos planteados en el proyecto y fue validado funcionalmente en distintos escenarios de prueba, durante su implementación y evaluación se identificaron diversas oportunidades de mejora.

Las siguientes propuestas no representan deficiencias estructurales del sistema, sino posibles líneas de evolución que permitirían aumentar su robustez, precisión, seguridad y nivel de integración, acercándolo a una solución más completa desde el punto de vista técnico e industrial.

1. Reemplazo del Arduino y placa por un Convertidor serial para comunicarse entre tu PC y DYNAMIXE

Actualmente, la comunicación entre ROS y los motores Dynamixel se realiza a través de una placa Arduino que actúa como intermediaria. Esta solución permitió desarrollar y validar el controlador sin inconvenientes, pero presenta una limitación en la velocidad y cantidad de datos que pueden intercambiarse con los motores.

Como mejora futura, se propone reemplazar el Arduino y la placa asociada por una interfaz específica para motores Dynamixel. Este tipo de placa está diseñada para comunicarse directamente con los actuadores a mayor velocidad y con acceso más eficiente a sus registros internos.

La implementación de esta mejora permitiría:

- Aumentar la frecuencia de lectura de posición y estado de los motores.
- Reducir la latencia en la comunicación.
- Obtener una realimentación de posición más continua y confiable.
- Permitir utilizar un controlador real de MoveIt basado en control cerrado.
- Permitir mediciones de otros factores como corriente para implementar control de torque.

En síntesis, esta modificación no cambia la lógica del sistema desarrollado, pero mejoraría su rendimiento y lo acercaría a una implementación más profesional y robusta.

2. Utilización de motores paso a paso con encoder de efecto Hall

Otra posible mejora del sistema consiste en reemplazar los actuadores actuales por motores paso a paso equipados con encoder de efecto Hall.

La principal ventaja de esta modificación sería el aumento de torque disponible, lo que permitiría construir un robot de mayor tamaño y con mayor capacidad de carga. En la configuración actual, el sistema puede presentar limitaciones cuando debe mover su propio peso en determinadas posiciones, lo cual restringe su aplicación práctica.

Además, el uso de motores con encoder incorporado permitiría contar con una medición continua y precisa de la posición real de cada articulación. Esto habilitaría una realimentación más estable y confiable, mejorando el control del movimiento.

Desde el punto de vista del sistema de control, esta solución ofrecería beneficios similares a los planteados en la mejora anterior, ya que permitiría una comunicación más fluida con los actuadores y un seguimiento más preciso de la posición, facilitando la implementación de estrategias de control cerrado integradas con MoveIt.

En resumen, esta propuesta no solo incrementaría la capacidad mecánica del robot, sino que también mejoraría la calidad del control y ampliaría las posibilidades de aplicación del sistema.

3. Incorporación de sensores de corriente para control de torque

Otra mejora posible consiste en incorporar sensores de corriente en cada motor, o al menos en las articulaciones principales del robot.

La medición de corriente permitiría estimar el torque que está desarrollando cada actuador en tiempo real. Esto es importante porque el torque está directamente relacionado con la fuerza que el robot ejerce sobre su entorno.

Con esta información sería posible implementar un sistema de detección de sobreesfuerzos o contactos inesperados. Por ejemplo, ante un aumento brusco de corriente

provocado por una colisión o por la presencia de un obstáculo, el sistema podría detener el movimiento o reducir la velocidad automáticamente.

Esta mejora acercaría el comportamiento del sistema a la definición de robot colaborativo (cobot), ya que permitiría incorporar mecanismos básicos de seguridad activa basados en la limitación de fuerza.

En síntesis, la incorporación de sensores de corriente no solo mejoraría el control del robot, sino que también aumentaría su nivel de seguridad y ampliaría sus posibilidades de interacción con el entorno.

4. Incorporación de botón de parada inmediata

Se propone incorporar un botón de parada inmediata tanto en la interfaz web como en la botonera física del sistema.

Actualmente, una vez que el robot inicia la ejecución de una trayectoria o rutina, esta se completa hasta su finalización. Si bien el sistema funciona correctamente, no cuenta con un mecanismo rápido para interrumpir el movimiento ante una situación inesperada, como un error de programación, una posible colisión o cualquier inconveniente durante la operación.

La incorporación de un botón de parada permitiría detener la ejecución en curso de forma inmediata, aumentando la seguridad y el control por parte del usuario. Esta mejora aportaría mayor robustez al sistema y lo acercaría a las prácticas habituales en sistemas industriales, donde la posibilidad de detener el movimiento en cualquier momento es fundamental.

5. Botón de eliminación de rutina desde la botonera

Otra mejora consiste en agregar un botón físico que permita eliminar la rutina actualmente cargada en el controlador cuando el sistema se utiliza sin la interfaz gráfica.

En la configuración actual, si se desea ejecutar una rutina diferente desde la botonera, es necesario reiniciar el controlador o realizar pasos adicionales para limpiar la rutina anterior. Esto limita la flexibilidad cuando el sistema opera de manera autónoma, sin conexión a la interfaz web.

La incorporación de un botón específico para eliminar la rutina activa permitiría liberar la memoria del controlador y cargar una nueva secuencia de forma más rápida y práctica. Esta mejora simplificaría el uso del sistema en entornos donde no se dispone de la interfaz gráfica y mejoraría la experiencia operativa general.

6. Migración del sistema a una Raspberry Pi

Actualmente, el controlador (ROS), la base de datos y el servidor de la interfaz web se ejecutan en una computadora de escritorio. Si bien esta configuración es adecuada para el desarrollo y las pruebas en laboratorio, limita la movilidad del sistema y lo vuelve dependiente de un equipo externo de mayor tamaño.

Como mejora futura, se propone migrar todo el sistema a una Raspberry Pi u otra computadora embebida de bajo consumo. Esta modificación permitiría integrar el controlador directamente en la estructura del robot, haciendo que el sistema sea más compacto y transportable.

Además, esta migración otorgaría mayor sentido práctico a la botonera física y a la interfaz web, ya que el robot podría operar de manera más autónoma sin depender de una PC externa. Esto ampliaría las posibilidades de uso y facilitaría su implementación en otros entornos.

En síntesis, esta mejora aumentaría la portabilidad, autonomía y practicidad del sistema.

7. Ampliación de funciones en la botonera física

Relacionada con la mejora anterior, se propone ampliar las funciones de la botonera incorporando botones específicos para:

- Inicializar el controlador ROS.
- Reiniciar el sistema.
- Cerrar correctamente el controlador.

En la configuración actual, estas acciones se realizan directamente desde la computadora, por lo que no resulta necesario disponer de estos controles físicos. Sin embargo,

si el sistema se ejecuta en una Raspberry Pi integrada al robot, la botonera debería permitir el control completo del ciclo de funcionamiento sin depender de una terminal externa.

- La incorporación de estos botones permitiría:
- Encender y preparar el sistema de forma autónoma.
- Reiniciar ante fallas sin intervención externa.
- Apagar el controlador de manera segura.

De esta manera, el robot podría funcionar como una unidad más independiente, acercándose a una solución más integrada y profesional.

8. Optimización del sistema de notificaciones y control de errores en la interfaz web

Actualmente, el sistema cuenta con un mecanismo de notificación de errores y advertencias tanto en la terminal como en la interfaz web. Los mensajes se diferencian por niveles de importancia mediante el uso de colores y, en algunos casos, se complementan con ventanas emergentes para alertar al usuario.

Si bien este sistema resulta funcional y permite identificar la mayoría de los problemas comunes, aún puede mejorarse en términos de claridad, organización y detalle de la información proporcionada.

Como mejora futura, se propone optimizar el sistema de gestión de errores incorporando:

- Clasificación más clara entre errores críticos, advertencias e información general.
- Mensajes más descriptivos que indiquen la posible causa del problema.
- Sugerencias básicas de acción para el usuario.
- Registro histórico de eventos relevantes dentro de la interfaz.

Esta mejora permitiría una interacción más intuitiva con el sistema, reduciría la posibilidad de errores operativos y facilitaría el diagnóstico ante fallas. Además, mejoraría la experiencia general del usuario, acercando la interfaz a estándares más profesionales de control industrial.

En síntesis, fortalecer el sistema de notificaciones no modifica la lógica de funcionamiento del robot, pero sí mejora significativamente su usabilidad y robustez operativa.

9. Migración a base de datos relacional para gestión de puntos y rutinas

Actualmente, el sistema utiliza una base de datos de tipo NoSQL para el almacenamiento de puntos y rutinas. Esta solución resulta adecuada por su simplicidad y bajo consumo de recursos, especialmente considerando la futura implementación del sistema en una plataforma embebida como una Raspberry Pi.

Sin embargo, en la estructura actual, las rutinas almacenan directamente las coordenadas de los puntos en lugar de mantener una referencia a los puntos guardados en la base de datos. Esto implica que, si un punto es modificado posteriormente, las rutinas que lo utilizan no reflejan dicho cambio, generando posibles inconsistencias en la ejecución.

Por ejemplo, si se redefine una posición de referencia como el “home”, las rutinas previamente guardadas continuarán utilizando la configuración anterior, aun cuando el usuario espere que se actualicen automáticamente.

Como mejora futura, se propone migrar a un modelo de base de datos relacional (SQL), donde los puntos y las rutinas se encuentren vinculados mediante relaciones explícitas. De esta manera, las rutinas podrían referenciar a los puntos en lugar de duplicar sus valores.

La implementación de esta mejora permitiría:

- Mantener coherencia entre los puntos y las rutinas asociadas.
- Evitar la duplicación de datos.
- Facilitar la actualización global de configuraciones.

- Mejorar la integridad y trazabilidad de la información.

Cabe destacar que, debido a las limitaciones de recursos de la plataforma objetivo, esta migración debería realizarse utilizando motores livianos, como SQLite, que permiten implementar un modelo relacional sin comprometer el rendimiento del sistema.

En síntesis, esta mejora no modifica la funcionalidad del sistema desde el punto de vista del usuario, pero sí fortalece la estructura interna de datos, aumentando la consistencia, mantenibilidad y escalabilidad del proyecto.

10. Habilitación de trayectorias lineales (LIN) y circulares (CIRC)

El sistema actualmente permite seleccionar trayectorias lineales (LIN) y circulares (CIRC) desde la interfaz web, y el nodo de trayectoria cuenta con la estructura necesaria para procesarlas. Sin embargo, estas trayectorias no se encuentran habilitadas en la versión actual, ya que requieren la ejecución de un controlador de MoveIt con realimentación continua del estado del robot.

En la configuración actual, el sistema opera únicamente con movimientos tipo PTP (punto a punto), lo cual resulta suficiente para la validación funcional del proyecto, pero limita la posibilidad de ejecutar trayectorias con control geométrico más preciso en el espacio cartesiano.

Como mejora futura, se propone habilitar la ejecución real de trayectorias LIN y CIRC mediante alguna de las siguientes alternativas:

- Implementar un controlador real de MoveIt, lo cual requiere previamente mejorar la comunicación y la realimentación de posición.

- Desarrollar un algoritmo propio que genere este tipo de trayectorias fuera del entorno MoveIt/ROS, calculando los puntos intermedios necesarios para aproximar movimientos lineales o circulares.

La implementación de esta mejora permitiría ampliar significativamente las capacidades del robot, mejorando la precisión del movimiento en el espacio de trabajo y acercando el sistema a funcionalidades típicas de robots industriales.

En síntesis, habilitar trayectorias LIN y CIRC representaría una evolución natural del proyecto hacia un nivel superior de planificación y control.

11. Edición directa de puntos y rutinas almacenadas en la base de datos

Actualmente, el sistema permite guardar puntos y rutinas en la base de datos para su reutilización posterior. Sin embargo, no es posible editar directamente un punto o una rutina ya almacenada.

En caso de necesitar una modificación, el procedimiento consiste en eliminar el elemento existente y volver a guardarlo con el mismo nombre. Este método, si bien es funcional, presenta algunas limitaciones:

- No constituye una edición directa sino una eliminación seguida de una nueva creación.
- Puede generar inconsistencias, especialmente en el caso de puntos utilizados dentro de rutinas ya guardadas.
- Obliga a realizar pasos adicionales que podrían evitarse con un sistema de edición más flexible.

Como mejora futura, se propone incorporar la posibilidad de editar directamente los puntos y rutinas almacenados en la base de datos, permitiendo modificar sus parámetros sin necesidad de eliminarlos previamente.

Esta funcionalidad permitiría:

- Mantener la coherencia entre puntos y rutinas asociadas.
- Simplificar el proceso de corrección.
- Reducir la posibilidad de errores involuntarios.
- Mejorar la experiencia general del usuario.

En síntesis, esta mejora optimizaría la gestión de datos del sistema y aportaría mayor robustez a la estructura de almacenamiento.

12. Incorporación de registro de fecha y verificación de coherencia entre rutinas

Se propone incorporar el registro automático de fecha y hora al momento de crear, editar y ejecutar una rutina.

Actualmente, el campo correspondiente ya existe en la base de datos dentro del elemento identificado como “control”, pero aún no se utiliza para almacenar información.

El objetivo de esta mejora es que, al momento de ejecutar o cargar una rutina, el sistema verifique no solo la existencia de las rutinas internas asociadas, sino también si estas han sido modificadas con posterioridad a la última edición de la rutina principal.

Por ejemplo, si una rutina “padre” contiene llamadas a rutinas “hijas”, el sistema podría comparar la fecha de última modificación de cada rutina hija con la fecha de edición de la rutina padre. En caso de detectar que alguna rutina interna fue modificada posteriormente, se mostraría una advertencia al usuario.

De esta manera, el operador sabría que el comportamiento final del robot podría no coincidir exactamente con la configuración original de la rutina principal. El sistema no impediría la ejecución, pero permitiría que el usuario confirme conscientemente la acción.

Esta mejora sería especialmente relevante en un entorno donde el robot puede ser utilizado por múltiples usuarios, ya que ayuda a mantener la coherencia entre configuraciones y evita ejecuciones inesperadas.

En síntesis, la incorporación de registro temporal y verificación de modificaciones aportaría mayor trazabilidad, seguridad lógica y control sobre la evolución de las rutinas almacenadas.

13. Ajuste y optimización del control PID de los actuadores

Durante las pruebas del sistema se observó la presencia de un error en estado estable en la posición de algunos actuadores, particularmente en las articulaciones de hombro y codo.

Este error se manifiesta principalmente cuando las articulaciones soportan mayor carga, generando pequeñas desviaciones angulares que, acumuladas, producen un error apreciable en la posición final del TCP.

Si bien se realizaron intentos de ajuste de los parámetros PID internos de los actuadores Dynamixel, no se logró alcanzar un resultado completamente satisfactorio dentro del tiempo disponible para el desarrollo del proyecto.

Cabe aclarar que el robot físico funciona como plataforma de validación del sistema de control e interfaz desarrollados, pero no constituye el eje central del trabajo. Por este motivo, se decidió no destinar más tiempo a la optimización fina del control interno de los motores, priorizando el desarrollo del controlador, la arquitectura de comunicación y la interfaz gráfica.

Como mejora futura, se propone realizar un ajuste más profundo del control PID, ya sea mediante una caracterización experimental sistemática de cada articulación bajo carga o mediante la implementación de estrategias de compensación adicionales. Esta optimización permitiría reducir el error en estado estable y mejorar la precisión global del robot, especialmente en posiciones donde las cargas generan mayores exigencias dinámicas.

En síntesis, la corrección del PID representa una mejora orientada a incrementar la precisión mecánica del sistema y complementar las mejoras propuestas en control y realimentación.

Glosario

Cobot (Robot colaborativo)	Robot diseñado para interactuar de forma directa y segura con personas, compartiendo el mismo espacio de trabajo. En este proyecto, el cobot permite tanto la ejecución automática de movimientos como la guía manual para el registro de puntos y trayectorias con fines didácticos.
Robot antropomórfico	Tipo de robot manipulador cuya estructura cinemática imita la disposición de un brazo humano, con articulaciones rotacionales encadenadas. El robot desarrollado en este trabajo responde a esta arquitectura.
ROS (Robot Operating System)	Middleware de código abierto utilizado para desarrollar sistemas robóticos modulares. Proporciona mecanismos de comunicación, herramientas de visualización y bibliotecas para planificación y control de movimiento
Nodo (ROS)	Proceso ejecutable dentro de ROS que realiza una función específica. En el sistema desarrollado, cada nodo cumple un rol definido, como la planificación de trayectorias, la comunicación con Arduino o la interacción con la interfaz gráfica.
Tópico (ROS)	Canal de comunicación utilizado por ROS para el intercambio de información entre nodos bajo el modelo publicador–suscriptor. Permite desacoplar los distintos módulos del sistema
Mensaje (ROS)	Estructura de datos utilizada para transmitir información a través de un tópico. Define el formato y el tipo de los datos intercambiados entre nodos
MoveIt	Framework de ROS utilizado para la planificación, ejecución y visualización de movimientos de robots. En este proyecto se emplea para calcular trayectorias del brazo robótico considerando su modelo cinemático
Pilz Industrial Motion Planner	Planificador integrado en MoveIt orientado a aplicaciones industriales, que permite generar trayectorias del tipo PTP y LIN con perfiles de velocidad controlados
URDF	Formato estándar de ROS para describir la estructura física de un robot, incluyendo enlaces, articulaciones y límites de movimiento. Es utilizado por MoveIt y RViz para la planificación y visualización
XACRO	Lenguaje de macros utilizado para generar archivos URDF de forma modular y parametrizable, facilitando la modificación y el mantenimiento del modelo del robot
Arduino	Plataforma de microcontrolador utilizada como interfaz entre el hardware del robot y la arquitectura de control basada en ROS. En este proyecto se encarga del control directo de los actuadores y de la lectura de la

	botonera.
Rosserial	Conjunto de herramientas que permiten la comunicación entre ROS y microcontroladores como Arduino a través de un enlace serie, tratándolos como nodos dentro del sistema ROS.
Dynamixel	Actuadores inteligentes que integran motor, reductor, electrónica de control y encoder en un único módulo. Se utilizan como articulaciones del robot desarrollado.
Dynamixel2Arduino	Librería de software que permite controlar actuadores Dynamixel desde Arduino, simplificando la implementación del protocolo de comunicación y el acceso a variables como posición y velocidad.
Encoder	Sensor que permite medir la posición angular de una articulación. En los actuadores Dynamixel se utilizan encoders absolutos para conocer la posición real del robot.
Interfaz gráfica	Sistema de interacción con el usuario basado en una plataforma web, que permite programar, ejecutar y visualizar movimientos del robot de manera intuitiva.
React	Biblioteca de JavaScript utilizada para el desarrollo de la interfaz gráfica del sistema, basada en componentes y actualización dinámica de la vista sin recargar la página.
Base de datos	Sistema utilizado para almacenar puntos, trayectorias y rutinas del robot. En este proyecto se emplea una base de datos NoSQL ligera para facilitar la integración con el controlador.
TinyDB	Base de datos NoSQL escrita en Python que almacena información en formato JSON. Se utiliza para guardar configuraciones, puntos y rutinas del robot sin requerir un servidor externo.
Modo control	Modo de operación en el cual el robot ejecuta movimientos y rutinas programadas a partir de órdenes enviadas por la interfaz gráfica o la botonera.
Modo lectura	Modo de operación que permite desenclavar los motores y guiar manualmente el robot para registrar posiciones o trayectorias, priorizando la seguridad del usuario.
Programación por demostración	Metodología mediante la cual el usuario enseña movimientos al robot guiándolo físicamente, permitiendo registrar y reproducir trayectorias sin programación explícita.

Referencias Bibliográficas

[1] E. Montini, “Collaborative robotics: A survey from literature and applications,” J. Intell. Robot. Syst., Springer, 2024. doi: 10.1007/s10846-024-02141-z.

[2] E. Salvato et al., “Singularity avoidance for cart-mounted hand-guided cobots,” Robotics, vol. 11, no. 4, art. 79, 2022. doi: 10.3390/robotics11040079.

[3] M. Safeea, R. Bearee, and P. Neto, “End-effector precise hand-guiding for collaborative robots,” in Proc. Int. Conf. Robotics and Automation, 2017. [Online]. Available: https://www.researchgate.net/publication/321976459 . [Accessed: Jan 3, 2026].

[4] Universal Robots, “Universal Robots – Collaborative industrial robots,” 2025. [Online]. Available: https://www.universal-robots.com . [Accessed: Jan 3, 2026].

[5] ROS Wiki, “Learning ROS for Robotics Programming,” 2025. [Online]. Available: https://wiki.ros.org/Books/LearningROSforRoboticsProgramming . [Accessed: Jan 3, 2026].

[6] ROS Wiki, “Learning ROS for Robotics Programming – Second Edition,” 2025. [Online]. Available: https://wiki.ros.org/Books/LearningROSforRoboticsProgramming_second_edition . [Accessed: Jan 3, 2026].

[7] ROS Wiki, “Robot Operating System (ROS),” 2025. [Online]. Available: https://wiki.ros.org/ . [Accessed: Jan 3, 2026].

[8] F. Paredes et al., “Position sensors for industrial applications: Optical and magnetic encoders,” Sensors, vol. 21, no. 8, 2021. doi: 10.3390/s21082715.

[9] Universidad Antonio José Camacho, “Documento académico sobre bases de datos,” 2020. [En línea]. Disponible en: https://repositorio.uniajc.edu.co/server/api/core/bitstreams/0732fc14-785c-4b8c-9ed1-0ec68fe7c0f7/content . [Accedido: 3-ene-2026].

[10] MoveIt, “MoveIt Motion Planning Framework,” 2025. [Online]. Available: https://moveit.ai/ . [Accessed: Jan 3, 2026].

[11] MoveIt, “Move Group Python Interface Tutorial,” 2025. [Online]. Available: https://moveit.github.io/moveit_tutorials/doc/move_group_python_interface/move_group_python_interface_tutorial.html . [Accessed: Jan 3, 2026].

[12] MoveIt, “MoveIt Concepts,” 2025. [Online]. Available: https://moveit.ai/documentation/concepts/ . [Accessed: Jan 3, 2026].

[13] MoveIt, “MoveIt Commander Scripting Tutorial,” 2025. [Online]. Available: https://moveit.github.io/moveit_tutorials/doc/moveit_commander_scripting/moveit_commander_scripting_tutorial.html . [Accessed: Jan 3, 2026].

[14] MoveIt, “Pilz Industrial Motion Planner,” 2025. [Online]. Available: https://moveit.github.io/moveit_tutorials/doc/pilz_industrial_motion_planner/pilz_industrial_motion_planner.html . [Accessed: Jan 3, 2026].
[15] MoveIt, “URDF and SRDF Tutorial,” 2025. [Online]. Available: https://moveit.github.io/moveit_tutorials/doc/urdf_srdf/urdf_srdf_tutorial.html . [Accessed: Jan 3, 2026].
[16] Articulated Robotics, “URDF for ROS,” 2025. [Online]. Available: https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/ . [Accessed: Jan 3, 2026].
[17] Articulated Robotics, “URDF for ROS,” 2025. [Online]. Available: https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/ . [Accessed: Jan 3, 2026].
[18] ROS Wiki, “Using Xacro to Clean Up a URDF File,” 2025. [Online]. Available: https://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File . [Accessed: Sep 15, 2025].
[19] ROS Wiki, “Create Your Own URDF File,” 2025. [Online]. Available: https://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file . [Accessed: Jan 3, 2026].
[20] ROS Wiki, “rosserial Arduino IDE Setup,” 2025. [Online]. Available: https://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup . [Accessed: Jan 3, 2026].
[21] S. Chen, U. R. Thaduri, and V. K. R. Ballamudi, “Front-end development in React: An overview,” Eng. Int., vol. 7, no. 2, pp. 117–126, 2019. doi: 10.18034/ei.v7i2.662. [19] S. Chen, U. R. Thaduri y V. K. R. Ballamudi, “Front-end development in React: An overview,” Eng. Int., vol. 7, no. 2, pp. 117–126, 2019. doi: 10.18034/ei.v7i2.662.
[22] React, “React Documentation,” 2025. [En línea]. Disponible en: https://es.react.dev/. . [Accedido: 3-ene-2026].
[23] ROBOTIS, “What is DYNAMIXEL?,” 2025. [Online]. Available: https://www.dynamixel.com/whatisdxl.php . [Accessed: Jan 3, 2026].
[24] ROBOTIS, “DYNAMIXEL XL430-W250 Reference Manual,” 2025. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/. [Accessed: Jan 3, 2026].
[25] ROBOTIS, “DYNAMIXEL XL320 Reference Manual,” 2025. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xl320/ . [Accessed: Jan 3, 2026].
[26] ROBOTIS, “Dynamixel2Arduino Library,” 2025. [Online]. Available: https://github.com/ROBOTIS-GIT/Dynamixel2Arduino . [Accessed: Jan 3, 2026].

[27] TinyDB, “TinyDB Documentation,” 2025. [Online]. Available: https://tinydb.readthedocs.io/en/latest/ . [Accessed: Jan 3, 2026].
[28] TinyDB, “TinyDB Introduction,” 2025. [Online]. Available: https://tinydb.readthedocs.io/en/latest/intro.html . [Accessed: Jan 3, 2026].
[29] MercadoLibre, “Módulo Step Down LM2596HV ajustable,” 2026. [En línea]. Disponible en: https://articulo.mercadolibre.com.ar/MLA-719205672-modulo-step-down-generico-lm2596hv-lm2596s-ajustable-_JM . [Accedido: 4-feb-2026].
[30] Geeetech Wiki, “Arduino Mega 2560,” 2025. [Online]. Available: https://wiki.geeetech.com/index.php/Arduino_Mega_2560 . [Accessed: Jan 3, 2026].
[31] MercadoLibre, “Placa plaqueta fenólico simple faz cobre PCB pertinax,” 2026. [En línea]. Disponible en: https://www.mercadolibre.com.ar/placa-plaqueta-fenolico-5x10-simple-faz-cobre-pcb-pertinax/up/MLAU279832176 . [Accedido: 4-feb-2026].
[32] ROBOTIS, “DYNAMIXEL XL430-W250-T,” 2026. [Online]. Available: https://www.robotis.us/dynamixel-xl430-w250-t/ . [Accessed: Feb 4, 2026].
[33] ROBOTIS, “DYNAMIXEL XL-320,” 2026. [Online]. Available: https://www.robotis.us/dynamixel-xl-320/ . [Accessed: Feb 4, 2026].
[34] MercadoLibre, “Mega 2560 R3 CH340G ATmega2560 compatible Arduino,” 2026. [En línea]. Disponible en: https://www.mercadolibre.com.ar/mega-2560-r3-ch340g-atmega2560-compatible-ardu-usb-nodo/p/MLA2039365926 . [Accedido: 4-feb-2026].

Anexos

Anexo 1: Tabla de tipos de mensajes de ROS

Tabla Nº17. Tipo de Mensajes

Tipo de msg	indicador	Tipo de dato
punto_correr	descriocion	int16[]
	coordenadas	float32[]
msg_web	mensaje	string
	tipo	int16
nombresPuntos	nombres	string[]
punto_web	orden	string[]
	coordenadas	float32[]
punto_real	cart	int16[]
	ang	int16[]
	error	bool[]
trayectorias	indice	int16[]
	posicion	int16[]
	Rutina	string[]
waits	indice	int16[]
	wait	int16[]

Anexo 2: Tabla de tópicos de ROS

Tabla Nº18. Tópicos

Topico	Tipo de msg	Para que se usa	Publicador	Subscriptor
guardar_trayectoria	Bool	Bandera para guardar una Trayectoria	arduino	modo_lectura
/move_group/display_planned_path	DisplayTrajectory	Plan generado por MoveIt	Moveit	modo_control
				controlador_cobot
/sequence_move_group/feedback	MoveGroupSequenceActionFeedback	feedback de MoveIT indica si se generó bien el plan o hubo error	Moveit	controlador_cobot
cord_dy	Int16MultiArray	Coordenadas a donde se tienen que mover los motores	controlador_cobot	arduino
cord_ros	punto_correr	informacion de punto a ejecutar individualmente	Pagina Web	modo_control
informe_web	msg_web	Mensajes de todos los procesos realizados	Pagina Web	Pagina Web
			modo_lectura	
joint_states	JointState	Coordenadas de la posicion del robot simulado por MoveIT	publicador_posicion_rea	Moveit
lista_puntosdb	nombresPuntos	Lista de Puntos Guardados	modo_lectura	Pagina Web
lista_rutinasdb	nombresPuntos	Lista de Rutinas Guarddas	modo_lectura	Pagina Web
modo_actuar	Bool	Indicador del modo del	modo_lectura	modo_lectura

		robot(Control-Lectura)	Pagina Web	Pagina Web
			arduino	arduino
orden_web	punto_web	manda la ordende la pagina web y la informacion necesaria	Pagina Web	modo_lectura
p_dy	Int16MultiArray	coordenadas de puntos guardados	arduino	modo_lectura
planificacion_A_trayectoria	Int16	comando para planificacion de ejecuacion rutinas y puntos entre nodos "controlador_cobot" y "modo_control"	controlador_cobot	modo_control
pos_dy	Int16MultiArray	coordenadas de la posicion real desde el robot	arduino	modo_lectura
				publicador_posicion_rea
pos_real	punto_real	coordendas convertidas para mostrar de la posicion real	modo_lectura	Pagina Web
puntodb	punto_web	informacion de puntos la base de datos para mostrar en la pagina web	modo_lectura	Pagina Web
puntoRutina	punto_web	informacion de rutinas la base de datos para mostrar en la pagina web	modo_lectura	Pagina Web
rutina	Bool	bandera para ordenar la ejecución de la rutina	modo_control	modo_control
			arduino	arduino
			Pagina Web	
rutine_trayectorias	trayectorias	nombre de las trayectoria y posicion a ejecutar de la rutina	modo_control	controlador_cobot
rutine_waits	waits	tiempo de espera y la posicion en el que se tiene que ejecutar en la rutina	modo_control	controlador_cobot
tipo_modo_lectura	Bool	Indicicador modo actual de modo lectura(Punto-Trayectoria)	Pagina Web	Pagina Web
			arduino	arduino
			modo_lectura	modo_lectura
trayectoria_A_planificacion	Int16	comando para planificacion de ejecuacion rutinas y puntos entre nodos "controlador_cobot" y "modo_control"	modo_control	controlador_cobot

Anexo 3: Tabla de nodos de ROS

Tabla Nº19. Nodos

Programa	Nodo	Tópico	USO	Tipo de msg
trayectoria37	modo_control	rutina	sp	Bool
		cord_ros	s	punto_correr
		planificacion_A_trayectoria	s	Int16

		/move_group/display_planned_path	s	DisplayTrajectory
		trayectoria_A_planificacion	p	Int16
		rutine_waits	p	waits
		rutine_trayectorias	p	trayectorias
Modo lectura 18	modo_lectura	p_dy	s	Int16MultiArray
		orden_web	s	punto_web
		puntodb	p	punto_web
		informe_web	p	String
		pos_dy	s	Int16MultiArray
		/tipo_modo_lectura	s	Bool
		/guardar_trayectoria	s	Bool
		puntoRutina	p	punto_web
		lista_puntosdb	p	nombresPuntos
		lista_rutinasdb	p	nombresPuntos
		pos_real	p	punto_real
		modo_actuar	sp	Bool
publicacion_posicion_real3	publicador_posicion_rea	pos_dy	s	Int16MultiArray
		joint_states	p	JointState
controlador_rutina_prueba4	controlador_cobot	cord_dy	p	Int16MultiArray
		planificacion_A_trayectoria	p	Int16
		/move_group/display_planned_path	s	DisplayTrajectory
		/sequence_move_group/feedback	s	MoveGroupSequenceActionFeedback
		trayectoria_A_planificacion	s	Int16
		rutine_waits	s	waits
		rutine_trayectorias	s	trayectorias
Pagina Web		cord_ros	p	punto_correr
		orden_web	p	punto_web
		puntodb	s	punto_web
		puntoRutina	s	punto_web
		lista_puntosdb	s	nombresPuntos
		lista_rutinasdb	s	nombresPuntos
		informe_web	sp	String
		pos_real	s	punto_real
		modo_actuar	sp	Bool
		tipo_modo_lectura	sp	Bool
		rutina	p	Bool

arduino	Rosserial	pos_dy	p	Int16MultiArray
		p_dy	p	Int16MultiArray
		rutina	p	Bool
		modo_actuar	sp	Bool
		tipo_modo_lectura	sp	Bool
		cord_dy	s	Int16MultiArray
		guardar_trayectoria	p	Bool

Anexo 4: Link de Repositorio del proyecto

[35] Repositorio de github de Proyecto completo. [En línea]. Disponible en: https://github.com/Diego26194/Cobot_Final_Proyect . [Accedido: 30-mar-2026].