



Universidad Nacional de San Luis  
Facultad de Ingeniería y Ciencias Agropecuarias

***EL TÍTULO SE ESCRIBE EN EL CENTRO SUPERIOR DE LA  
HOJA Y PODRÁ OCUPAR MÁS DE UN RENGLÓN, EN CUYO  
CASO TENDRÁ INTERLINEADO DE 1,5 pto.***

Autor o autores

Trabajo final de Ingeniería (agregar carrera que corresponda)

Director  
Codirector / Codirector Técnico  
Asesor/s

Villa Mercedes, San Luis  
año

## **DERECHO DE AUTOR**

© año, nombre y apellido del/la autor/a tal como aparece en la portada.

Se autoriza la reproducción total o parcial, con fines académicos, por cualquier medio o procedimiento, incluyendo la cita bibliográfica del documento.

## **DEDICATORIA**

*Texto de dedicatoria justificado.*

## **AGRADECIMIENTOS**

*Texto de agradecimientos justificado.*

## RESUMEN

El presente trabajo final de grado tiene como objetivo el diseño y fabricación de un controlador e interfaz gráfica modular para robots antropomorfos colaborativos (COBOTS) de 6 grados de libertad, brazos robóticos de 6 ejes los cuales se pueden programar moviéndolo con la mano indicándole a donde y/o como uno quiere que se mueva, aparte de poder programarlo de la forma habitual. Para poder probar su funcionamiento se armó un robot cuyo diseño de la estructura fue brindado por “ labme ” . Todo el proyecto, controlador mas robot, se pensó para funcione como herramienta didáctica en el Laboratorio de Mecatronica de la Facultad de Ingeniería y Ciencias agropecuarias (LABME). Para la parte principal del controlador se utilizó un sistema operativo para robots utilizado en la industria e investigación llamado ROS, el cerebro del robot, a esto se le añadió una base de datos para guardar bien los puntos y rutinas y se comunicó con un arduino que se encarga de controlar los motores del robot en base a lo que ROS le indique. En cuanto a la interfaz, se realizó una página web donde el usuario puede programar y controlar el robot. Se implementaron 2 modos de uso, uno donde el usuario puede indicarle mediante la interfaz a donde quiere que se mueva, ya sea como punto o armando una rutina de varios puntos ingresados o guardados previamente, y el otro modo donde el usuario puede manipular el robot manualmente y guardar posiciones o toda una trayectoria completa para luego correrla o modificarla con el otro modo. Se realizaron pruebas para validar la operatividad y cumplimiento de los objetivos. Los resultados obtenidos demostraron el correcto funcionamiento del controlador e interfaz. Estas experiencias validaron su funcionalidad como herramienta didáctica, proporcionando a los estudiantes experiencia práctica en robótica y control, abriendo la posibilidad de futuras ampliaciones y mejoras del sistema.

El resumen debe ser de un solo párrafo, escrito en lenguaje apto para todo público sin utilizar terminología técnica específica. Debe ser un reflejo preciso del contenido del documento. No contendrá ecuaciones matemáticas, citas de referencia ni notas al pie. Su extensión debe ser de al menos 500 palabras y no más de una carilla. Fuente Arial tamaño 11, justificado, interlineado 1.5.

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 uso didáctico. Para ello, se diseñó y construyó 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 indicarle movimientos. 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** — Cobot, Robot antropomórfico, ROS.

## **ÍNDICE DE CONTENIDO**

CAPITULO 1: Propuesta	8
Introducción	8
Objetivos	9
Objetivo general	9
Objetivos específicos	9
Alcances y limitaciones	10
Marco teórico y/o justificación y/o estado del arte	11
Marco teórico	11
Justificación	11
Estado del arte	11
CAPITULO X: Análisis y Desarrollo	12
CAPITULO X: Análisis de Costos	13
CAPITULO X: Estudio de Impacto Ambiental	14
CAPITULO X: Conclusiones	15
Glosario	16
Referencias Bibliográficas	17
Anexo/s	18
Apéndice/s	19

## **ÍNDICE DE FIGURAS**

Figura N° 1. Esquema de experiencia sobre polarización.	12
---	----

## **ÍNDICE DE TABLAS**

Tabla N°1. Abreviaturas	12
-------------------------	----



# **CAPITULO 1: Propuesta**

## **Introducción**

Texto de la introducción. Texto de la introducción.

# **Objetivos**

## **Objetivo general**

- Diseñar y desarrollar un sistema de control y una interfaz para un robot colaborativo de tipo antropomórfico, capaces de operar en dos modos principales —“movimiento” y “lectura”—, integrando el hardware necesario en un robot construido para la validación práctica del sistema y evaluando su potencial como herramienta didáctica en el ámbito universitario.

## **Objetivos específicos**

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

## **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 página 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.

# Marco teórico y/o justificación y/o estado del arte

## Marco teórico

En esta sección se consignará el contenido del marco teórico.

### Cobots

Los robots colaborativos (cobots) 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.

Montini, E., *Collaborative Robotics: A Survey From Literature and Applications*, 2024. (Revisión exhaustiva; Springer / repositorio del autor).

<https://link.springer.com/article/10.1007/s10846-024-02141-z>

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.

Salvato, E. et al., *Singularity Avoidance for Cart-Mounted Hand-Guided Cobots*, Robotics (MDPI), 2022. (discute programación por demostración y singularidades)

<https://www.mdpi.com/2218-6581/11/4/79>

Una funcionalidad central de muchos cobots es la programación por demostración o “hand-guiding”, mediante la cual un operador guía físicamente el efecto 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 efecto 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.

Safeea, M.; Bearee, R.; Neto, P., *End-Effector Precise Hand-Guiding for Collaborative Robots*, (paper sobre métodos de hand-guiding y compensación). (citado en ResearchGate / actas).

[https://www.researchgate.net/publication/321976459\\_End-Effector\\_Precise\\_Hand-Guiding\\_for\\_Collaborative\\_Robots](https://www.researchgate.net/publication/321976459_End-Effector_Precise_Hand-Guiding_for_Collaborative_Robots)

## ROS

### 1. Robot Operating System (ROS)

<https://wiki.ros.org/Books/LearningROSforRoboticsProgramming>

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 [1], [3]. 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 [1].

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 [1].

## **2. Nodos, tópicos y su relación**

### **2.1 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 [1]. 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 [1].

### **2.2 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.

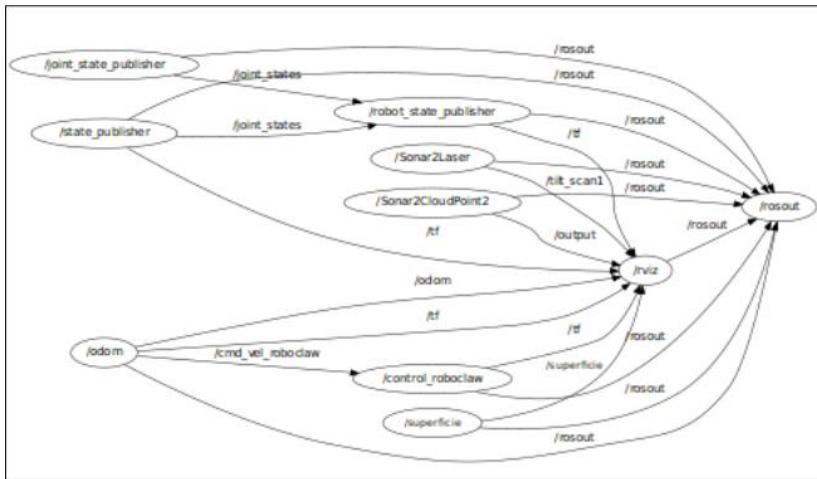
Un nodo suscriptor recibe los mensajes publicados en ese tópico.

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 [1].

### **2.3 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 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` [1].



### **3. Mensajes y tipos de mensajes**

### 3.1 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 [1].

## 3.2 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 [1].

#### 4. Movelt: planificación y control de movimiento

Movelt es un framework dentro del ecosistema ROS para la planificación, ejecución y visualización de movimientos de robots, especialmente manipuladores industriales y colaborativos [2].

Movelt 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 [2].

El framework se apoya en bibliotecas externas como OMPL y ofrece una interfaz unificada que simplifica el desarrollo de aplicaciones de movimiento complejas [2].

<https://moveit.ai/>

#### **4.1 Move Group y Movelt Commander**

El nodo move\_group actúa como el núcleo de planificación y ejecución de Movelt. Interfaces de alto nivel, como moveit\_commander en Python, permiten definir objetivos de movimiento sin necesidad de implementar directamente los algoritmos de planificación subyacentes [2].

[https://moveit.github.io/moveit\\_tutorials/doc/move\\_group\\_python\\_interface/move\\_group\\_python\\_interface\\_tutorial.html](https://moveit.github.io/moveit_tutorials/doc/move_group_python_interface/move_group_python_interface_tutorial.html)

[https://moveit.github.io/moveit\\_tutorials/doc/moveit\\_commander\\_scripting/moveit\\_commander\\_scripting\\_tutorial.html](https://moveit.github.io/moveit_tutorials/doc/moveit_commander_scripting/moveit_commander_scripting_tutorial.html)

<https://moveit.ai/documentation/concepts/>

#### **5. Planificador Pilz Industrial Motion Planner**

El Pilz Industrial Motion Planner es un planificador integrado a Movelt, 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 [4].

[https://moveit.github.io/moveit\\_tutorials/doc/pilz\\_industrial\\_motion\\_planner/pilz\\_industrial\\_motion\\_planner.html](https://moveit.github.io/moveit_tutorials/doc/pilz_industrial_motion_planner/pilz_industrial_motion_planner.html)

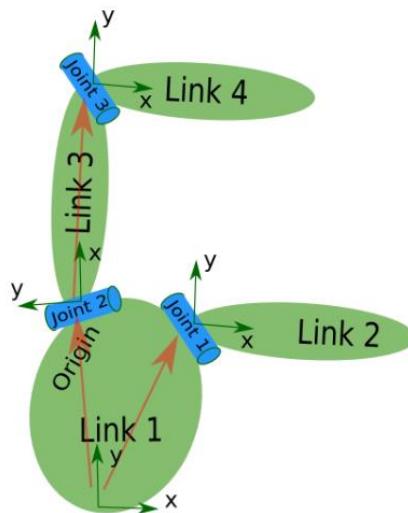
#### **6. URDF y XACRO**

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 [2][3].

[https://moveit.github.io/moveit\\_tutorials/doc/urdf\\_srdf/urdf\\_srdf\\_tutorial.html](https://moveit.github.io/moveit_tutorials/doc/urdf_srdf/urdf_srdf_tutorial.html)

<https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/>

Dado que los modelos URDF extensos pueden volverse difíciles de mantener, 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 [3][4].



<https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/>

<https://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File>

## **Rosserial y aduino**

[https://wiki.ros.org/rosserial\\_arduino/Tutorials/Arduino%20IDE%20Setup](https://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup)

## **Archivo .xacro?**

## **Moveit?**

## **Pilz?**

## **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 [1].

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 [1].

[1] S. Chen, U. R. Thaduri y V. K. R. Ballamudi, “Front-End Development in React: An Overview,” *Engineering International*, vol. 7, no. 2, pp. 117–126, 2019. DOI: 10.18034/ei.v7i2.662. Disponible en: <https://abc.us.org/ojs/index.php/ei/article/view/662>

[https://www.researchgate.net/profile/Venkata-Ballamudi/publication/374154236\\_Front-End\\_Development\\_inReact\\_An\\_Overview/links/6510d94f61f18040c220eb13/Front-End-Development-in-React-An-Overview.pdf](https://www.researchgate.net/profile/Venkata-Ballamudi/publication/374154236_Front-End_Development_inReact_An_Overview/links/6510d94f61f18040c220eb13/Front-End-Development-in-React-An-Overview.pdf)

## 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.

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.

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 combine suficiente resolución para control y suavidad de trayectorias con redundancia o filtrado que mejore robustez ante lecturas erráticas.

<https://pmc.ncbi.nlm.nih.gov/articles/PMC8069193/>

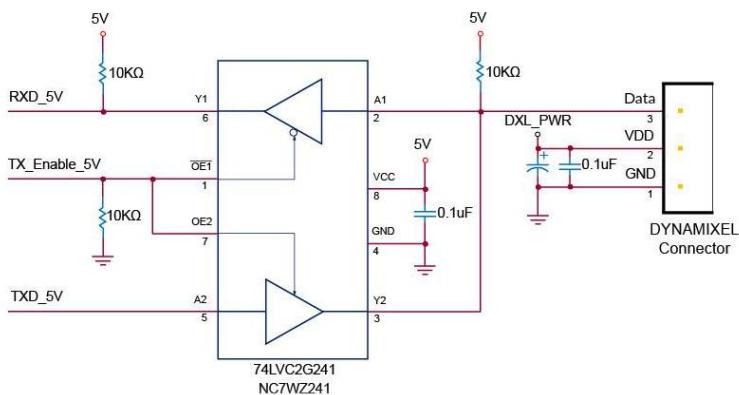
Paredes, F. et al., *Position Sensors for Industrial Applications: Optical and Magnetic Encoders* (revisión). PMC — Sensors (2021).

## Dynamixels

Los actuadores Dynamixel utilizados en este proyecto son motores inteligentes de la serie XL de Robotis, 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

<https://www.dynamixel.com/whatisdxi.php>

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



<https://emanual.robotis.com/docs/en/dxl/xl430-w250/#control-table-description>

Tabla N°1. Abreviaturas

Características	XL430-W250	XL320


<https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/#reference>

<https://emanual.robotis.com/docs/en/dxl/x/xl320/>

### 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.

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.

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.

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.

<https://repositorio.uniajc.edu.co/server/api/core/bitstreams/0732fc14-785c-4b8c-9ed1-0dec68fe7c0f7/content>

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.

<https://tinydb.readthedocs.io/en/latest/>

<https://tinydb.readthedocs.io/en/latest/intro.html>

### **Arduino+Librerías?**

## **Justificación**

En esta sección se consignará el contenido de la justificación.

Este proyecto forma parte de una iniciativa mayor del Laboratorio de Mecatrónica orientada a incorporar robots didácticos que representen modelos utilizados en la industria. El desarrollo del software de control para un cobot de 6 grados de libertad contribuye a ampliar las herramientas disponibles para la formación práctica de los estudiantes. Si bien el hardware

acompaña el proyecto con fines demostrativos, el eje principal de este trabajo final es el diseño e implementación del sistema de control, fundamental para brindar a los alumnos una experiencia más cercana a la operación y programación de robots industriales.

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

## **Estado del arte**

En esta sección se consignará el contenido del estado del arte.

# CAPITULO X: Análisis y Desarrollo

## 2. Análisis y Desarrollo

### 2.1 Presentacion de las partes del sistema

#### 2.1.1 Robot

##### 2.1.1.1 Estructura

##### 2.1.1.2 Actuadores

##### 2.1.1.3 Placas Electronica

##### 2.1.1.4 Arduino

#### 2.1.2 Base de Datos

##### 2.1.2.1 Tinydb

##### 2.1.2.2 Modulo db\_puntos.py

#### 2.1.3 Arquitectura ROS

##### 2.1.3.1 modo\_lectura

##### 2.1.3.2 modo\_control

##### 2.1.3.3 controlador\_cobot

##### 2.1.3.4 publicador\_posicion\_real

##### 2.1.3.5 Archivo inicializador launch

#### 2.1.4 Interfaz Grafica

### 2.2 Comunicacion o intercomunicación

### 2.3 Manual de uso

#### 2.3.1 Modo control

#### 2.3.2 Modo lectura

##### 2.3.2.1 modo punto

##### 2.3.2.2 modo trayectoria

### 2.4 Manual de instalacion

#### *2.1 Presentación general del sistema*

#### *2.1.1 Robot*

Subdividido en:

- Estructura
- Actuadores
- Electrónica

- Arduino

Perfecto. No es redundante si en 2.1 hacés solo un “pantallazo general”.

#### *2.1.2 Base de datos (*TinyDB*)*

Correcto.

Deberías incluir:

- Por qué TinyDB y no SQLite, PostgreSQL, etc.
- Cómo se organiza la información (rutas, puntos, metadatos).

#### *2.1.3 Arquitectura ROS*

Muy importante en tu tesis.

Aquí deberían ir:

- Explicación de la arquitectura ROS que diseñaste
- Función de cada nodo
- Qué hace cada servicio / tópico
- Cómo se comunican
- Por qué elegiste esa estructura

Los subapartados que pusiste están buenísimos.

#### *2.1.4 Interfaz gráfica*

Perfecto que expliques la evolución y diseño final.

También podés incluir:

- Interacción con ROS
- Tecnologías usadas (sin profundizar demasiado)

---

## *2.2 Comunicación o Intercomunicación*

Esta sección es **clave**.

Aquí explicás cómo encaja TODO:

- Arduino ↔ ROS
- ROS ↔ Web
- ROS ↔ Base de datos
- Usuario ↔ Interfaz

También describís:

- Cada tópico
- Qué publica, qué recibe
- Frecuencias
- Intercambio de mensajes
- Arquitectura general del sistema

#### **Esta sección no debe eliminarse.**

Es lo que muestra que el sistema es coherente.

---

#### *2.3 Manual de uso*

Totalmente válido.

Muchas tesis de ingeniería lo incluyen al final del capítulo técnico.

---

#### *2.4 Manual de instalación*

También es completamente válido.

De hecho, **es un aporte enorme** para que otros estudiantes puedan replicarlo.

Incluye:

- Instalación de ROS
- Dependencias
- Estructura de carpetas
- Permisos y configuraciones
- Iniciar los nodos
- Iniciar la interfaz web
- Conectar Arduino

. 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.

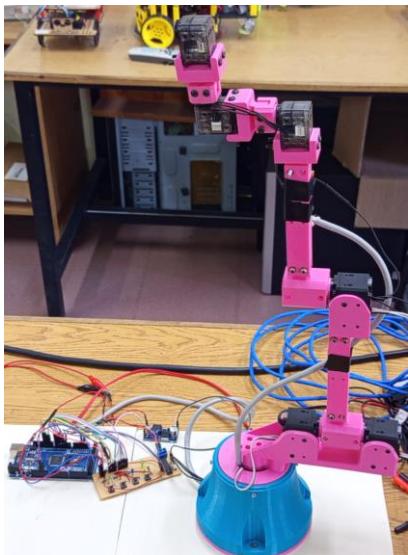
## 2.1 Presentación general del sistema

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 y escalar o 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 “**BOTONERA**”, una placa controladora dedicada a los motores, y una fuente de alimentación principal de 11.1 V junto con un variador de tensión “**NOMBRE DE VARIADOR DE TENSION**”. En la Figura: se presenta una vista general del robot con la identificación de cada uno de sus componentes

[FIGURA X]

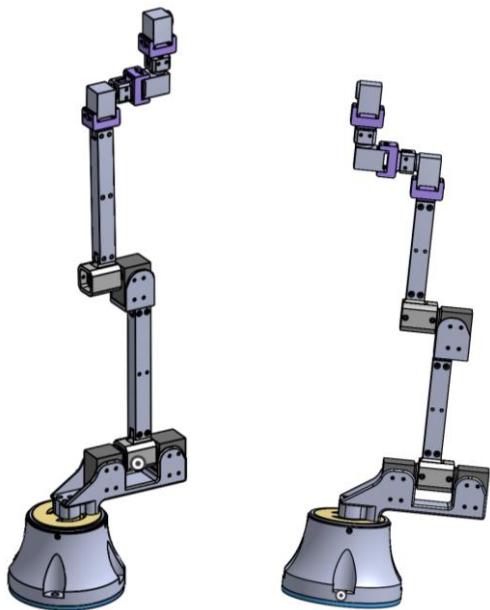


**Estructura**

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:](#), [\(REFERENCIA\)](#). Inicialmente, los eslabones correspondientes al brazo desde el hombro hasta la muñeca tenían una longitud mayor [Figura](#) (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 X]

[FIGURA X]



### Actuadores

En cuanto a los actuadores, se emplearon motores Dynamixel XL-430 ([Figura:](#)) y XL-320 ([Figura:](#)). Estos componentes habían sido adquiridos por el LabMe donde uno de los fines que era construir un cobot, 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 un integrado “**NOMBRE DE INTEGRADO**”, 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 (**Figura:**)

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 X]



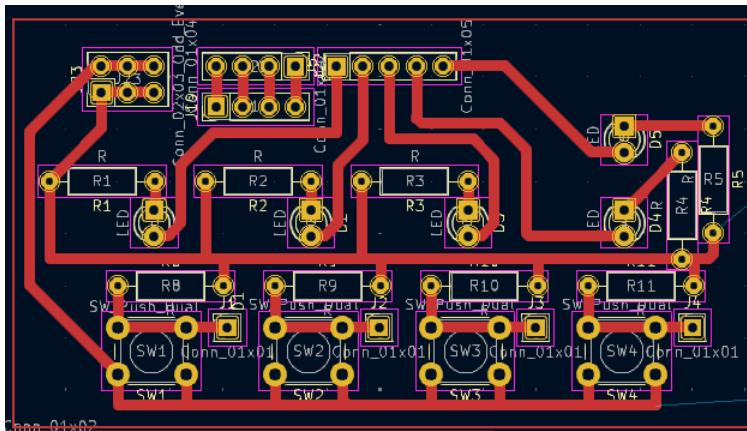
[FIGURA X]

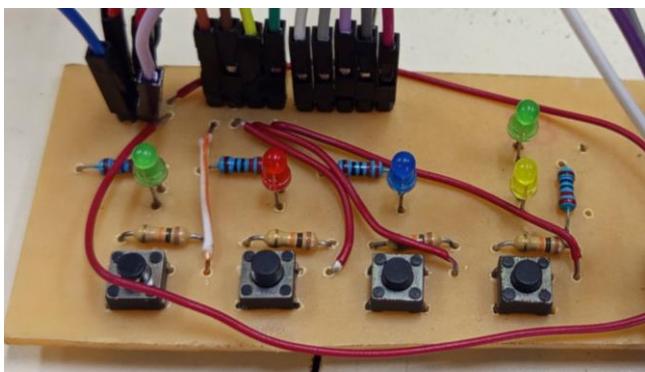
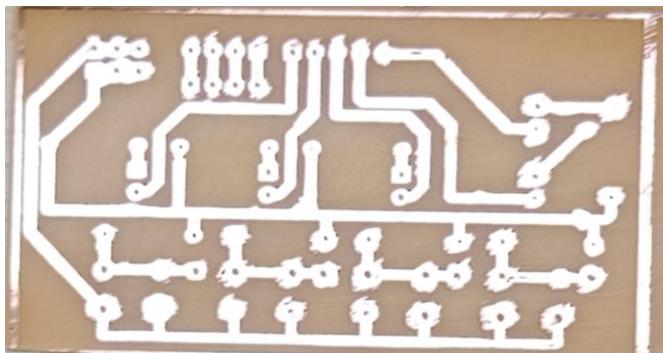


## Placas Electronica

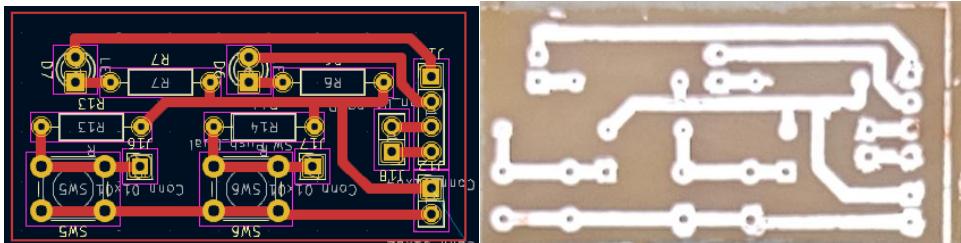
Considerando la futura migración del sistema ROS hacia una **Raspberry Pi**, se incorporó una “**BOTONERA**” destinada a permitir el control básico del robot sin depender de la interfaz gráfica. Esto es especialmente útil para demostraciones rápidas, pruebas de laboratorio o exposiciones. La **BOTONERA** 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:**), y otro ubicado en el eslabón entre el codo y la muñeca, utilizado para desenclavar los motores y registrar puntos (**Figura:**). Su funcionamiento detallado se describe más adelante.

[FIGURA X]





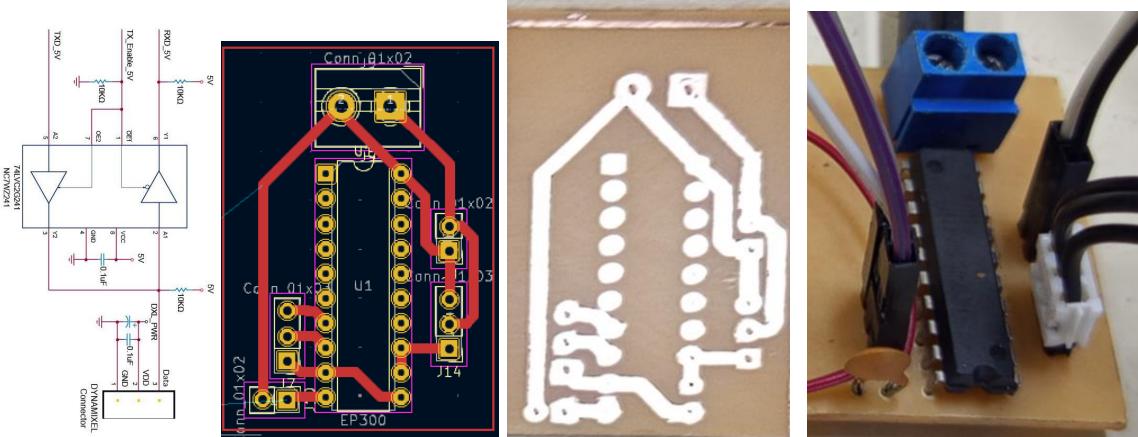
[FIGURA X]



Además, se diseñó una tercera placa destinada al control de los actuadores Dynamixel por medio de arduino ([Figura](#)), esta se hizo a parte para que se pueda utilizar en cualquier otro proyecto que utilicen estos actuadores. ([REFERENCIA?](#))

[FIGURA X]

[FIGURA X]



## 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 “**NOMBRE DE VARIADOR DE TENSION**” y el controlador Arduino ATmega2560, todos conectados a la fuente de 11.1 V y con punto de neutro común (**Figura**):.

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 DYNAMIXELS**”, 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 “**ROSERIAL**”, 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 página 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.

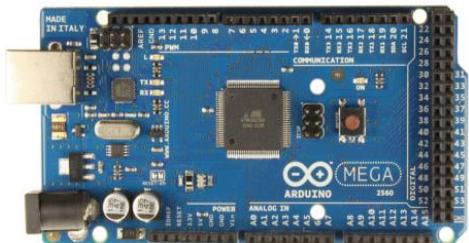
Posteriormente, se evaluó la influencia de la comunicación mediante “**ROSERIAL**”, optándose por reducir el tamaño de los mensajes transmitidos a través de los tópicos. Para ello, se reemplazaron tipos de mensajes complejos —como joint\_states— por arreglos de tipo int16, lo que implicó modificaciones 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 DYNAMIXELS**”. 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 explicará más adelante)

[FIGURA X]



## 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 [REFERENCIA], 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.

### db\_puntos6.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 Movelt/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 página 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.

#### **modo\_lectura ([modo\\_lectura18.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.

#### **modo\_control ([trayectoria37.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 página 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 página web para trayectorias simples. Con estos datos arma la secuencia correspondiente y se la envía al planificador de Movelt (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

### **controlador\_cobot (controlador\_rutina\_prueba4.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.

### **publicador\_posicion\_real (publicacion\_posicion\_real3.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

#### Extras:

Dentro del proyecto también se programaron otros modulos que si bien no realizan la parte de controlar el robot, ayudan y complementan esos nodos

#### Manejo\_Coordenadas

Este modulo es un servicio que entrega funciones que transforma las coordenadas en diferentes formatos utilizados según corresponda, ya que cada agente necesita este dato de diferentes formas

Coordenadas radianes: Es la mejor forma de indicar a ROS un movimiento PTP

Coordenadas grados

#### 2.1.4 Interfaz Grafica

Para la interfaz gráfica se decidió desarrollar una página 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 página 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 página: 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 X].

[FIGURA X]

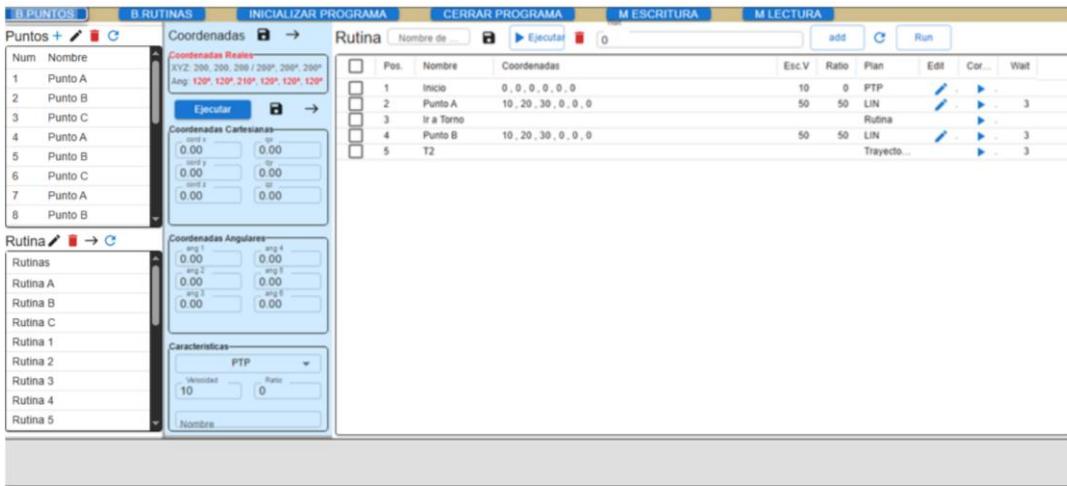
Sin embargo, durante la implementación del control de errores surgieron diversas limitaciones. En primer lugar, se evidenció que la página no podía verificar si una instrucción era válida hasta el momento de ejecutarla, lo que podía generar rutinas con puntos incorrectos o imposibles de realizar. Esto, 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 página, 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 página pasó a actuar como un visualizador del estado real del sistema. Es decir, la página 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 X].

[FIGURA X]

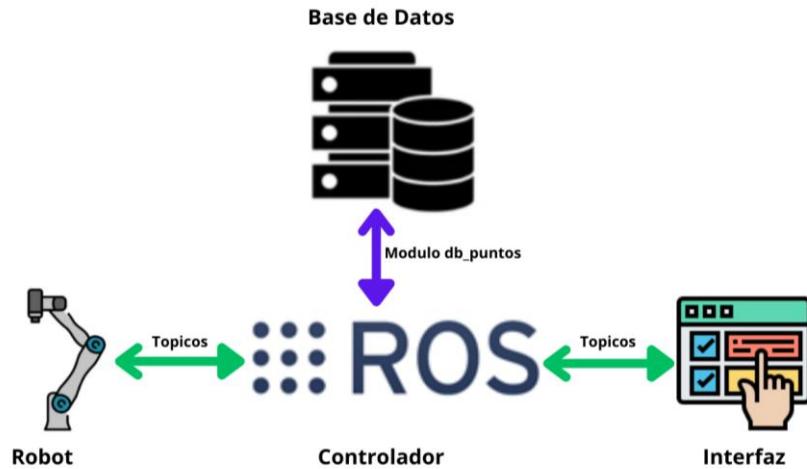


## 2.2 Conexiones

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] 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 X]



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:] del anexo 1

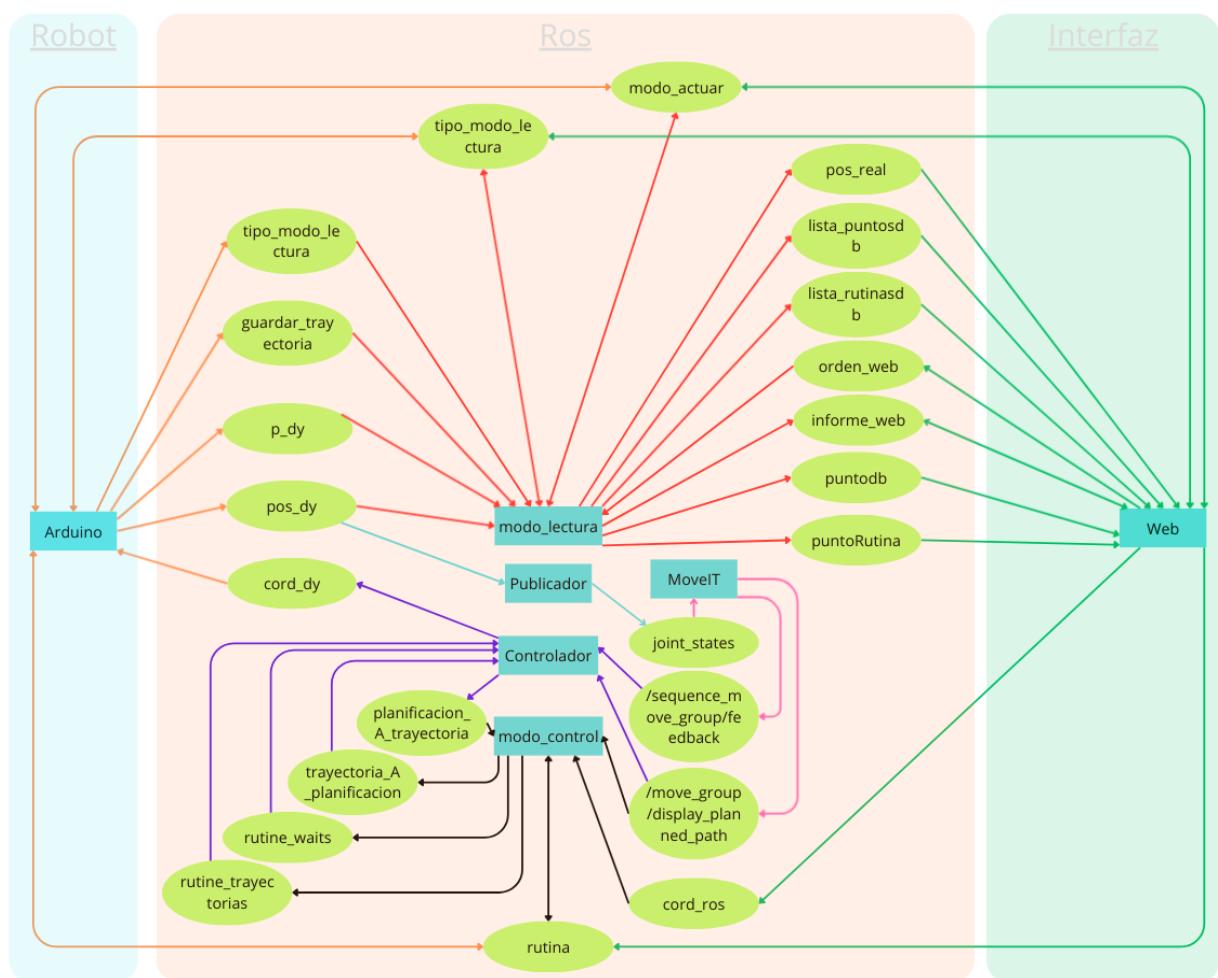
Con el fin de ofrecer una referencia precisa para futuros desarrolladores y estudiantes —especialmente quienes deseen modificar, extender o reemplazar algún módulo—, en la [TABLA:] del anexo 2 se detalla cada tópico utilizado. Para cada uno se especifica el tipo de mensaje, su función dentro del sistema 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 :] del anexo 3.

Finalmente, para facilitar aún más la comprensión de la arquitectura del sistema, en la [FIGURA] 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.



## 2.3 Funcionamiento

En este capítulo se explicara el funcionamiento y la forma de utilizar el controlador, junto con el robot utilizado para las pruebas; Primero se explicara los diferentes modos que tiene el controlador, luego se presentara cada elemento de la interfaz gráfica y la botonera, por último se explicara cómo utilizar el controlador y como funciona.

### 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 funciones programadas o puntos específicos, a su vez puede ir creando funciones y guardando puntos, pero solo a través de la interfaz gráfica ingresados a través de ella, no guardar los puntos de las posiciones del robot

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 esta manipulándolo. Solo en este modo el usuario puede guardar las posiciones del robot real, y desenclavarlo para poder moverlo manualmente. Aparte de esto el usuario puede seguir programando rutinas y guardándolas junto con puntos a través de la página 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 [BOTONERA] se guarda un único punto 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.

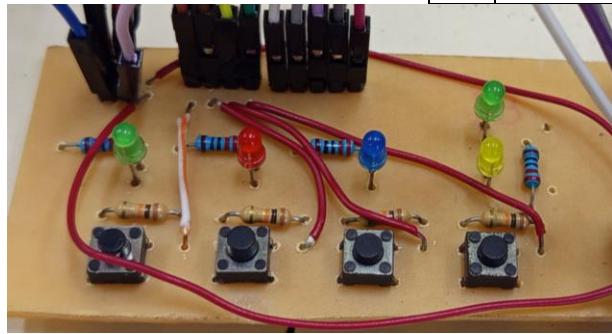
Sub-Modo Trayectoria: En este sub-modo cuando el usuario guarda la posición con la [BOTONERA] se guarda 20 puntos por segundos mientras el usuario mantenga pulsado el botón de desenclavar, y luego para guardar todos esos puntos en un único archivo el usuario debe presionar el botón de guardado. En este caso se guardan los datos tal cual entran y luego al correrlo se van a entregar 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) [FIGURA: ]

Tabla N°1. Abreviaturas

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

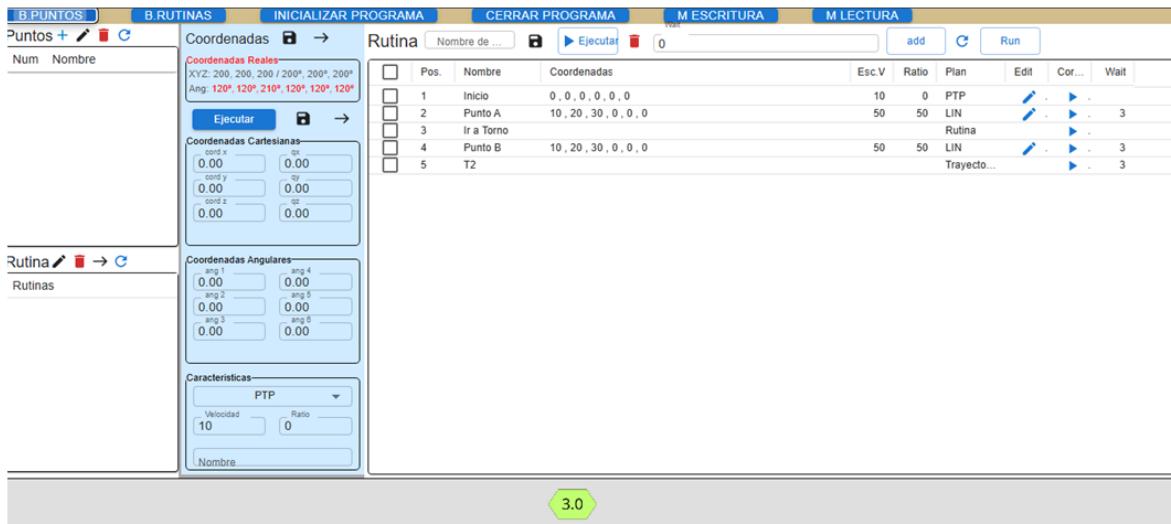


Botonera de Lectura (2.x) [FIGURA: ]

Tabla N°1. Abreviaturas

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

Página Web (3.x)[FIGURA: ]



Pantalla de mensajes (3.0) [FIGURA: ] :

Barra de herramientas (3.1.x) [FIGURA: ] :

Tabla N°1. Abreviaturas

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



Sección de Puntos Guardados (3.2.x) )[FIGURA: ] :

Tabla N°1. Abreviaturas

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



Num	Nombre
1	Punto A
2	Punto B
3	Punto C
4	Punto A
5	Punto B
6	Punto C
7	Punto A
8	Punto B

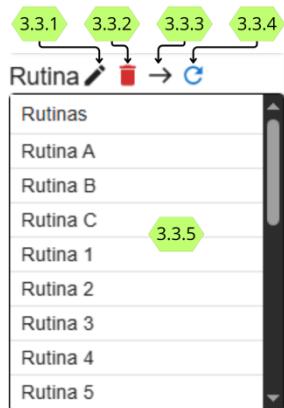
3.2.1-Boton agregar punto guardado a la rutina

3.2.2-Boton de editar Puto

**Sección de Rutinas Guardadas (3.3.x) [FIGURA: ] :**

Tabla N°1. Abreviaturas

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

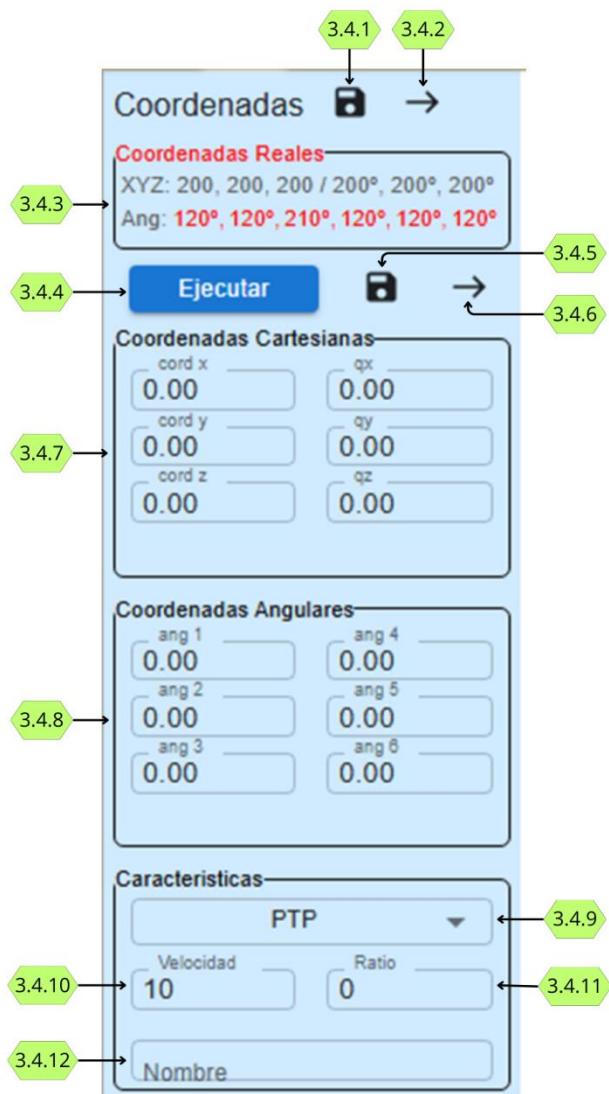


Rutinas
Rutina A
Rutina B
Rutina C
Rutina 1
Rutina 2
Rutina 3
Rutina 4
Rutina 5

**Sección de Coordenadas (3.4.x) [FIGURA: ] :**

Tabla N°1. Abreviaturas

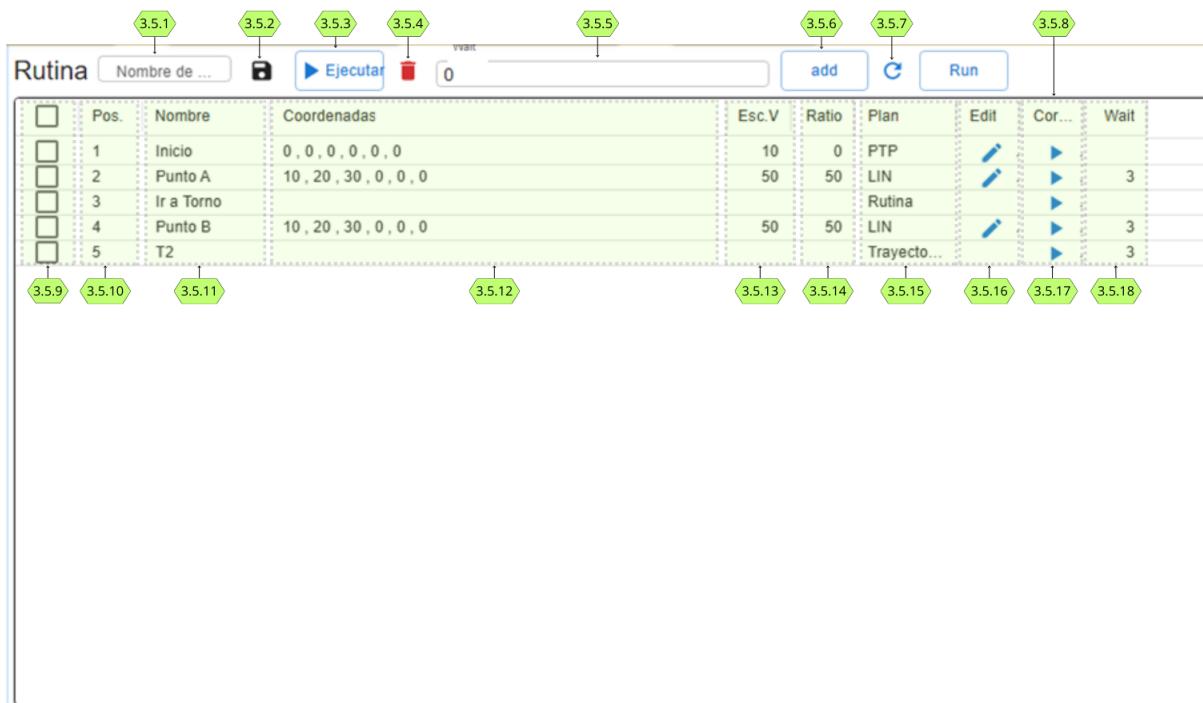
3.4.x	Sección de Coordenadas
3.4.1	Botón guardar punto real
3.4.2	Botón agregar punto real a la rutina
3.4.3	Coordenadas reales
3.4.4	Botón ejecutar punto
3.4.5	Botón guardar punto
3.4.6	Botón agregar punto a la rutina
3.4.7	Inputs <b>coordenadas cartesianas</b>
3.4.8	Inputs coordenadas angulares
3.4.9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



### Sección de Rutina (3.5.x) [FIGURA: ] :

Tabla Nº1. Abreviaturas

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



### 2.3.3 Descripción:

Primero hay que explicar unos conceptos usados:

**Rutina actual (RA):** es el conjunto de instrucciones guardada en una tabla volátil de la base de datos, esta es la que se ejecutara cuando se ordene correr la rutina, y donde el usuario programara. 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 borrara automáticamente al iniciar el controlador, en esta se copiaran las rutinas guardadas y solo ingresaran 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”\( 3.5.8\)](#)

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

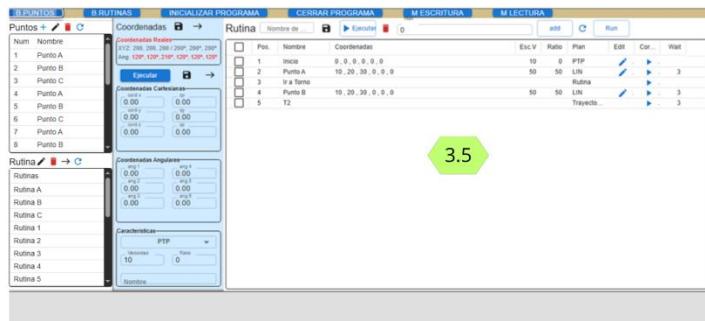
**Punto:** El robot va de la posición actual del a las coordenadas indicadas, si se ordena desde la [sección de Coordenadas \(3.4.x\)](#) cada motor va al ángulo indicado, en cambio si se ordena desde [sección de Rutina \(3.5.x\)](#) el **TCP** del robot es el que va a la coordenada cartesiana indicada.

Dentro de este tipo de instrucción se puede correr con 2 planes diferentes PTP(punto a punto) donde el robot va a las coordenadas indicadas con el menor movimiento de los motores, y el plan LIN(lineal) donde el robot va al punto indicado con una trayectoria recta

**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 la esta sección que es la parte principal del proyecto, en esta se puede visualizar, editar y guardar la (RA),



**“Cuadro de rutina actual”(3.5.8)** se visualiza la (RA), en el que cada fila es una instrucción que al momento de correrlas y las columnas características importantes de estas, las cuales se pueden modificar en su mayoría ahí mismo en la tabla

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
	1	Inicio	0 , 0 , 0 , 0 , 0 , 0	10	0	PTP			
	2	Punto A	10 , 20 , 30 , 0 , 0 , 0	50	50	LIN			3
	3	Ir a Torno				Rutina			
	4	Punto B	10 , 20 , 30 , 0 , 0 , 0	50	50	LIN			3
	5	T2				Trayecto...			3

**“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.

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
	1	Inicio	0 , 0 , 0 , 0 , 0 , 0	10	0	PTP			
	2	Punto A	10 , 20 , 30 , 0 , 0 , 0	50	50	LIN			3
	3	Ir a Torno				Rutina			
	4	Punto B	10 , 20 , 30 , 0 , 0 , 0	50	50	LIN			3
	5	T2				Trayecto...			3

**“Posiciones de las instrucciones”(3.5.10)** sirve para indicar el orden con el que se ejecutarán las instrucciones y por dentro como un identificador para algunos procesos

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
	1	Inicio	0 , 0 , 0 , 0 , 0 , 0	10	0	PTP			
	2	Punto A	10 , 20 , 30 , 0 , 0 , 0	50	50	LIN			3
	3	Ir a Torno				Rutina			
	4	Punto B	10 , 20 , 30 , 0 , 0 , 0	50	50	LIN			3
	5	T2				Trayecto...			3

**“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 esta vacía

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
<input type="checkbox"/>	1	Inicio	0 , 0 , 0 , 0 , 0	10	0	PTP		-	-
<input type="checkbox"/>	2	Punto A	10 , 20 , 30 , 0 , 0	50	50	LIN		-	-
<input type="checkbox"/>	3	Ir a Torno				Rutina		-	-
<input type="checkbox"/>	4	Punto B	10 , 20 , 30 , 0 , 0	50	50	LIN		-	-
<input type="checkbox"/>	5	T2				Trayecto...		-	-

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 **4rad/seg**, solo para puntos, para rutina y trayectoria esta casilla está vacía

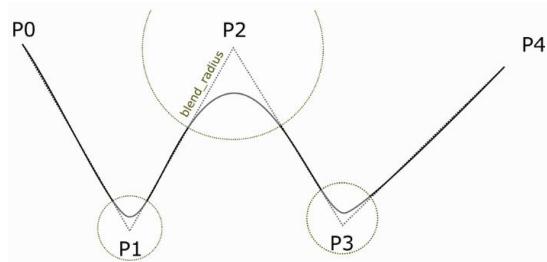
	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
<input type="checkbox"/>	1	Inicio	0 , 0 , 0 , 0 , 0	10	0	PTP		-	-
<input type="checkbox"/>	2	Punto A	10 , 20 , 30 , 0 , 0	50	50	LIN		-	-
<input type="checkbox"/>	3	Ir a Torno				Rutina		-	-
<input type="checkbox"/>	4	Punto B	10 , 20 , 30 , 0 , 0	50	50	LIN		-	-
<input type="checkbox"/>	5	T2				Trayecto...		-	-

3.5.13

**Precisión de la instrucción”( 3.5.14):** 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 mas 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

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
<input type="checkbox"/>	1	Inicio	0 , 0 , 0 , 0 , 0	10	0	PTP		-	-
<input type="checkbox"/>	2	Punto A	10 , 20 , 30 , 0 , 0	50	50	LIN		-	-
<input type="checkbox"/>	3	Ir a Torno				Rutina		-	-
<input type="checkbox"/>	4	Punto B	10 , 20 , 30 , 0 , 0	50	50	LIN		-	-
<input type="checkbox"/>	5	T2				Trayecto...		-	-

3.5.14



[https://moveit.github.io/moveit\\_tutorials/doc/pilz\\_industrial\\_motion\\_planner/pilz\\_industrial\\_motion\\_planner.html](https://moveit.github.io/moveit_tutorials/doc/pilz_industrial_motion_planner/pilz_industrial_motion_planner.html)

**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

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
	1	Inicio	0 , 0 , 0 , 0 , 0 , 0		10	0	PTP		
	2	Punto A	10 , 20 , 30 , 0 , 0 , 0		50	50	LIN		
	3	Ir a Torno					Rutina		
	4	Punto B	10 , 20 , 30 , 0 , 0 , 0		50	50	LIN		
	5	T2					Trayecto...		

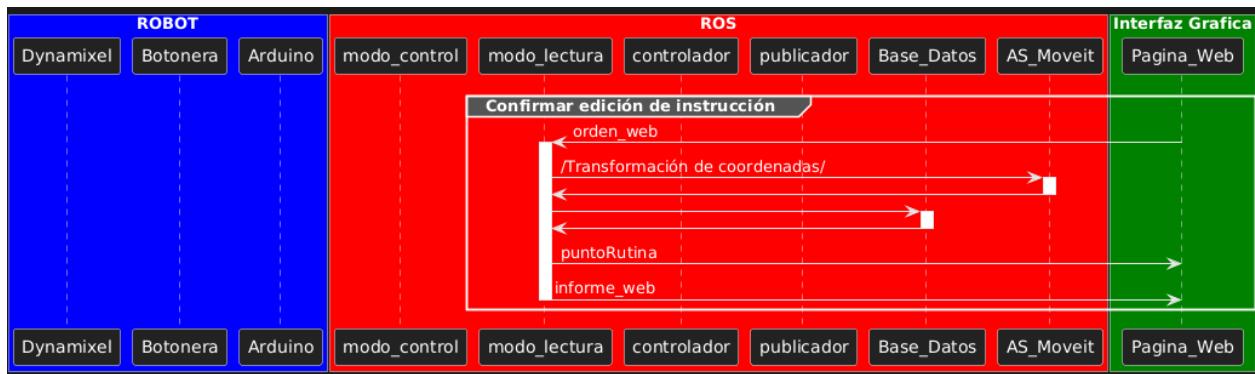
3.5.15

**3.5.16-Boton de habilitar/confirmar edición de instrucción:** Normalmente en ninguna casilla se puede ingresar y en la casilla de esta columna aparece un **lápiz azul**, al apretarlo cambia a un símbolo **✓** y permite que las columnas de esta fila puedan ser modificadas, 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)** 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.

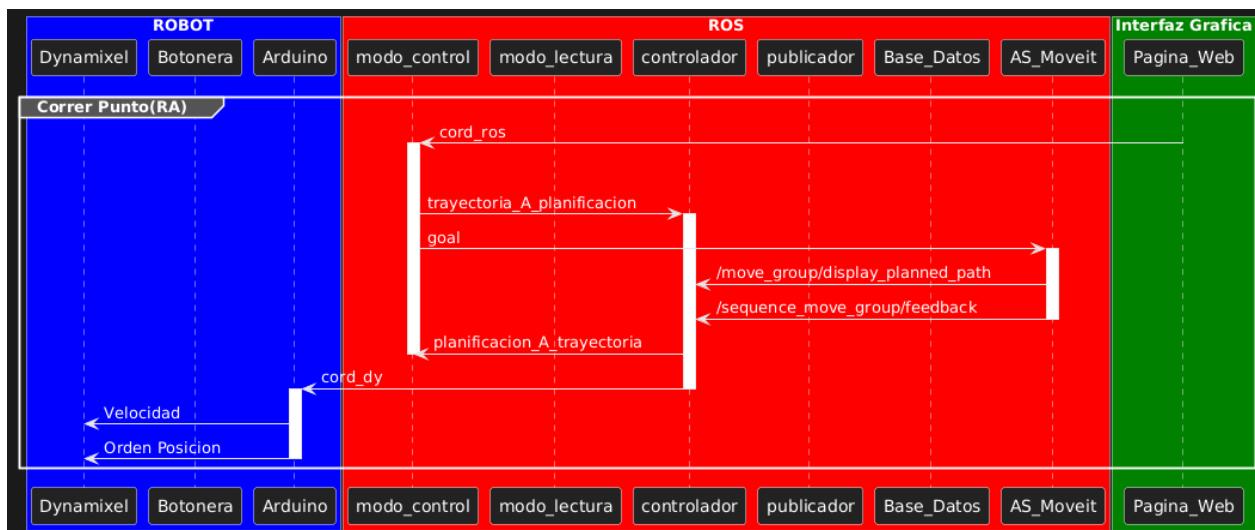
Al volver a apretar sobre el símbolo de esta columna, que ahora es **✓**, indica al sistema que quiere guardar la modificación y recién ahí lo envía al controlador, este verifica si es una instrucción valida e informa a la página web, si no es aceptable se muestra el error en la **Pantalla de mensajes** (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 invalidadas.

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
	1	Inicio	0 , 0 , 0 , 0 , 0 , 0		10	0	PTP		
	2	Punto A	10 , 20 , 30 , 0 , 0 , 0		50	50	LIN		
	3	Ir a Torno					Rutina		
	4	Punto B	10 , 20 , 30 , 0 , 0 , 0		50	50	LIN		
	5	T2					Trayecto...		

3.5.16 3.5.17



**“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.



**“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.

	Pos.	Nombre	Coordenadas	Esc	V	Ratio	Plan	Edit	Cor...	Wait
<input type="checkbox"/>	1	Inicio	0,0,0,0,0,0			10	0	PTP		
<input type="checkbox"/>	2	Punto A	10,20,30,0,0,0			50	50	LIN		
<input type="checkbox"/>	3	Ir a Torno						Rutina		
<input type="checkbox"/>	4	Punto B	10,20,30,0,0,0			50	50	LIN		
<input type="checkbox"/>	5	T2						Trayecto...		

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.

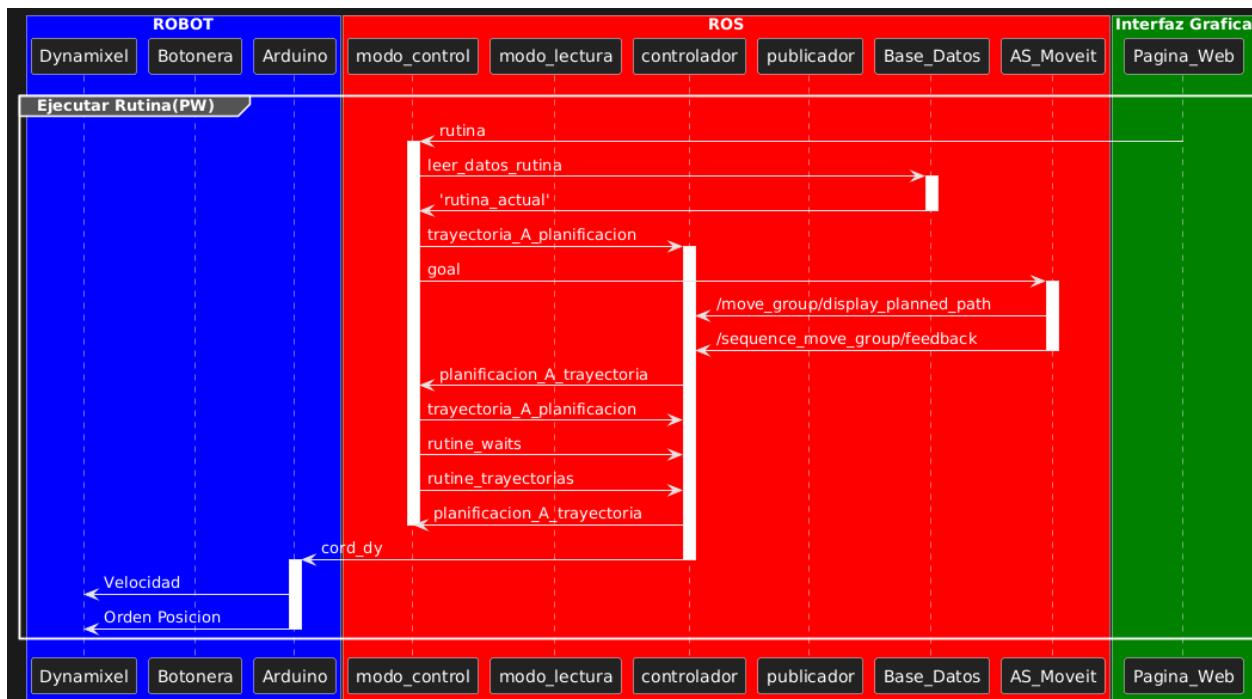
Si no se ingresó ningún nombre el sistema le crea uno genérico que nos e repita con los existentes.

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



**“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

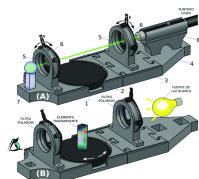


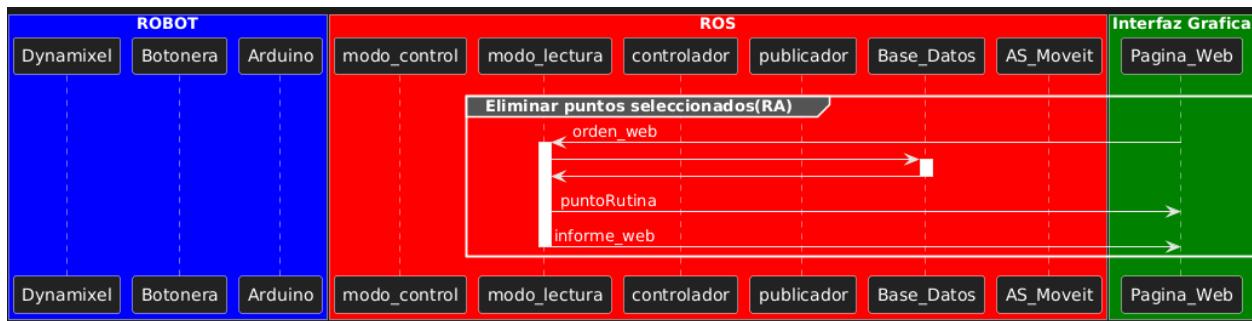


**“Botón eliminar puntos seleccionados”( 3.5.4):** al cliquear este botón se eliminaran las instrucciones previamente seleccionadas (**“check de instrucciones de rutina” (3.5.9)** ), si alguna instrucción no existe se indicara al usuario por medio de una **“ventana emergente”**.  
**[FIGURA: ].**

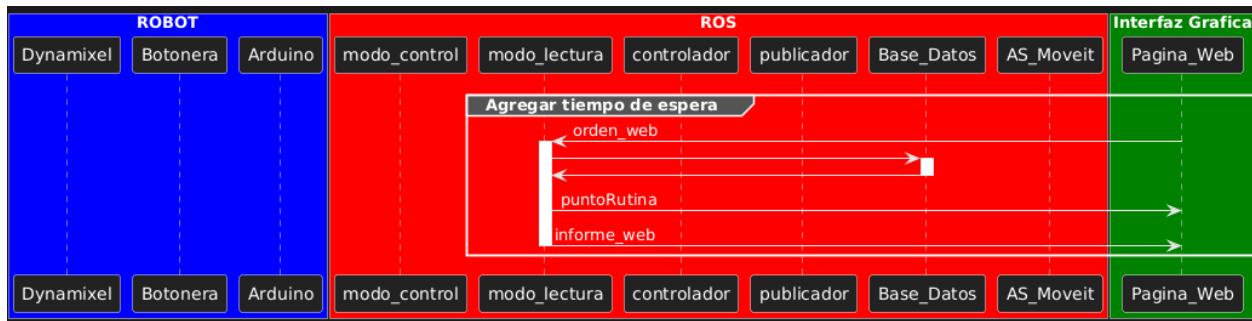
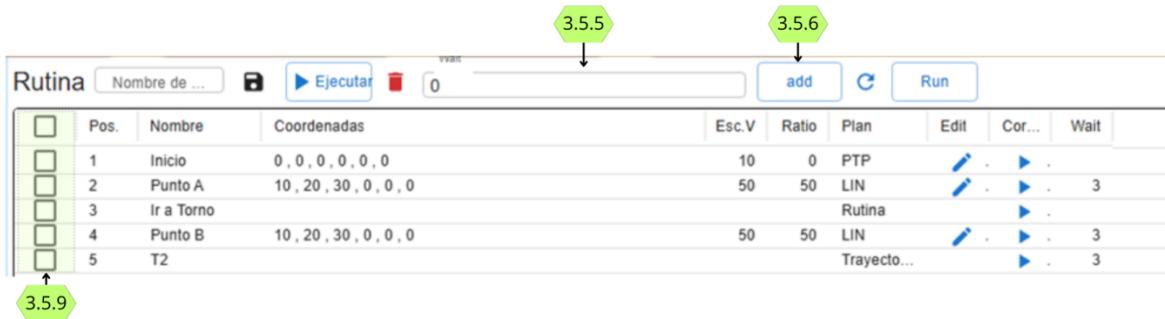


**[FIGURA X]**





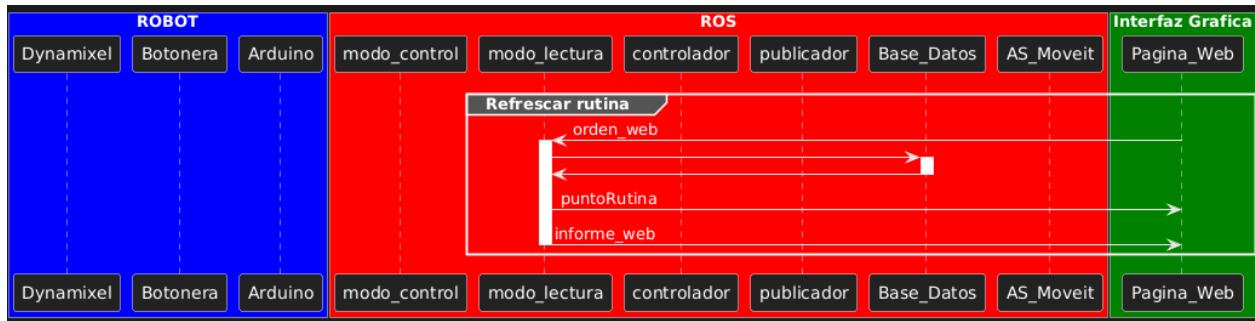
**“Botón agregar tiempo de espera”( 3.5.6):** al cliquear este botón se agregara a las instrucciones previamente seleccionadas (**“check de instrucciones de rutina” (3.5.9)** ) el tiempo indicado ingresado en **“Input tiempo de espera”( 3.5.5)**



**“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.

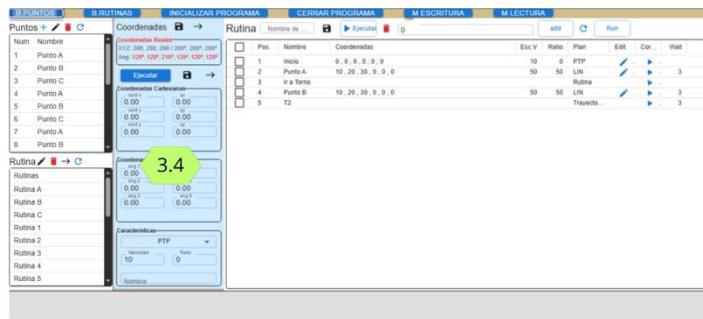
Rutina

	Pos.	Nombre	Coordenadas	Esc.V	Ratio	Plan	Edit	Cor...	Wait
<input type="checkbox"/>	1	Inicio	0, 0, 0, 0, 0, 0	10	0	PTP	<input type="button" value="Edit"/>	<input type="button" value="Cor..."/>	<input type="button" value="Wait"/>
<input type="checkbox"/>	2	Punto A	10, 20, 30, 0, 0, 0	50	50	LIN	<input type="button" value="Edit"/>	<input type="button" value="Cor..."/>	3
<input type="checkbox"/>	3	Ir a Torno				Rutina	<input type="button" value="Edit"/>	<input type="button" value="Cor..."/>	
<input type="checkbox"/>	4	Punto B	10, 20, 30, 0, 0, 0	50	50	LIN	<input type="button" value="Edit"/>	<input type="button" value="Cor..."/>	3
<input type="checkbox"/>	5	T2				Trayecto...	<input type="button" value="Edit"/>	<input type="button" value="Cor..."/>	3



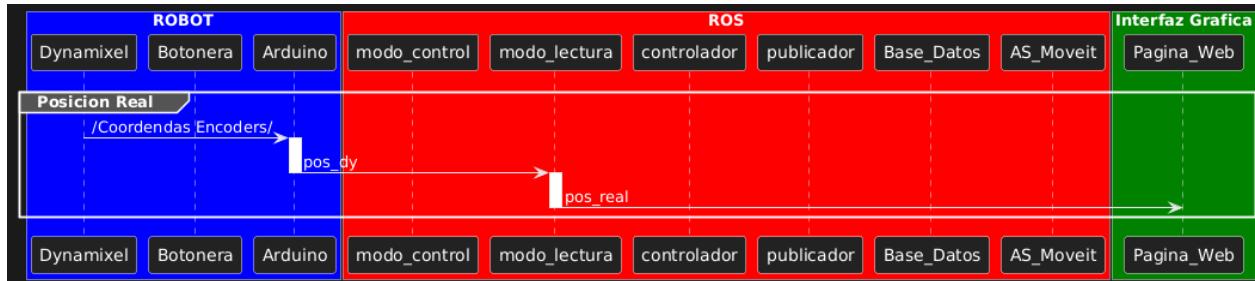
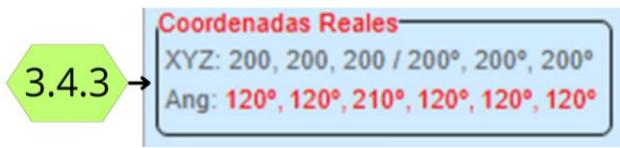
## Sección de Coordenadas

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



**"Coordenadas reales"( 3.4.3):** aquí se visualiza las coordenadas angulares y cartesianas reales que se obtiene de los encoder de los motores, 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: ]

Se podrá roja la coordenada angular del vector que este desconectado



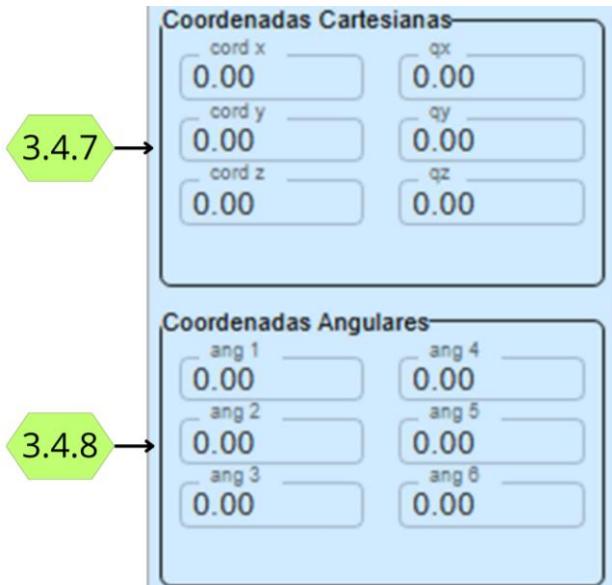
**“Inputs coordenadas cartesianas”( 3.4.7) y**

**“Inputs coordenadas angulares”( 3.4.8):**

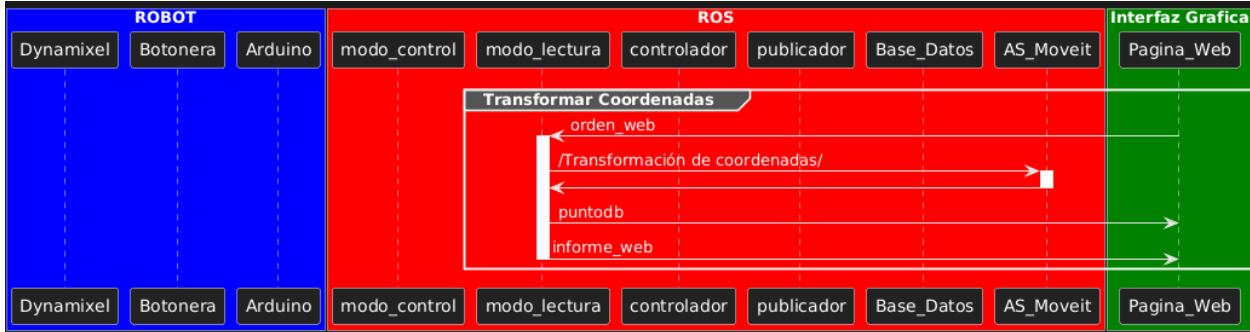
sirven para ingresar las coordenadas que quiero utilizar, ya sean angulares(cada una representa un los ángulos que se moverán los motores, siendo 1 el de la base y 6 el del extremo del robot) en grados o cartesianas(que representa las coordenadas del TCP) en milímetros (x,y,z) y grados (qx, qy, qz).

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 pagina web para que siempre estén sincronizados ambos conjuntos y el usuario tenga una mejor idea de como se va a mover el robot en base a sus instrucciones, asi si ingresa unas coordenadas angulares, puede visualizar la posición que tendrá el TCP, y viceversa

En caso de ingresar coordenadas angulares el controlador realizara un transformación por cinemática directa lo que casi siempre da un resultado. En cambio al realizarlo al revés, el controlador tratará de pasar de cartesianas a angulares, en este caso lo hace por cinematica inversa que no siempre encuentra resultados por mas que el robot pueda moverse allí **ya que se pueden producir singularidades por la posición actual del robot, intenta ver como llegar y alguno puntos tienen soluciones infinitas lo que no permite seguir con los cálculos;** esto nos sirve también como control de errores informándole al usuario que ese punto puede generar



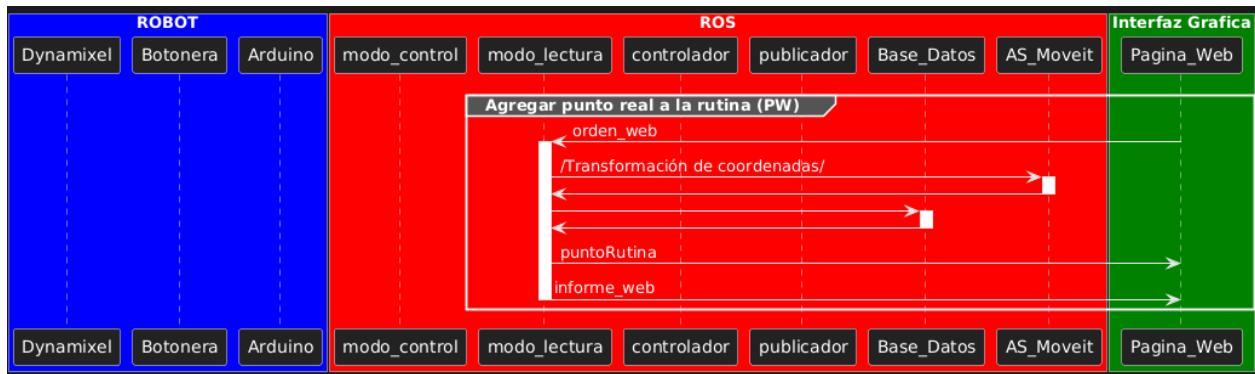
conflicto si se quiere usar en la rutina. Cuando genera un error, tanto para pasar de angulares a cartesianas y viceversa, se informa por la **Pantalla de mensajes** (3.0) y **no se modifica el otro conjunto de coordenadas**.



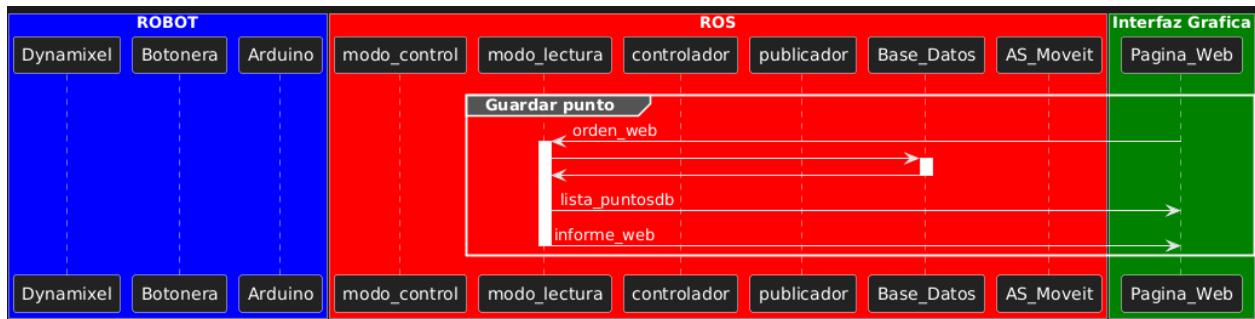
**“Botón guardar punto real”( 3.4.1):** Guarda el punto donde se encuentra el robot real en la base de datos para luego ser usado de forma rápida, es muy útil para puntos que piense repetir en una rutina o puntos de referencia muy utilizados(como un home). Para esto utiliza las coordenadas angulares 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)



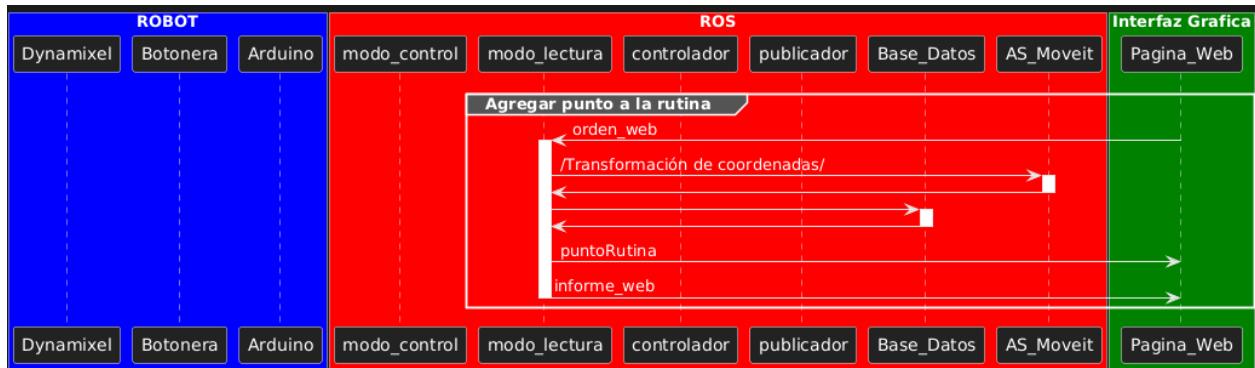
**“Botón agregar punto real a la rutina”( 3.4.2):** Agrega a la **RA** una instrucción del tipo punto al final de esta utilizando todos los datos de **“Desplegable tipo de trayectoria”**( 3.4.9), **“Velocidad del punto”**( 3.4.10) porcentual(max:4rad/seg), **“Precisión del punto”** (3.4.11) en milímetro, **“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 guarda 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 cartesianas que es la que el usuario más comprende.



**“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)**.

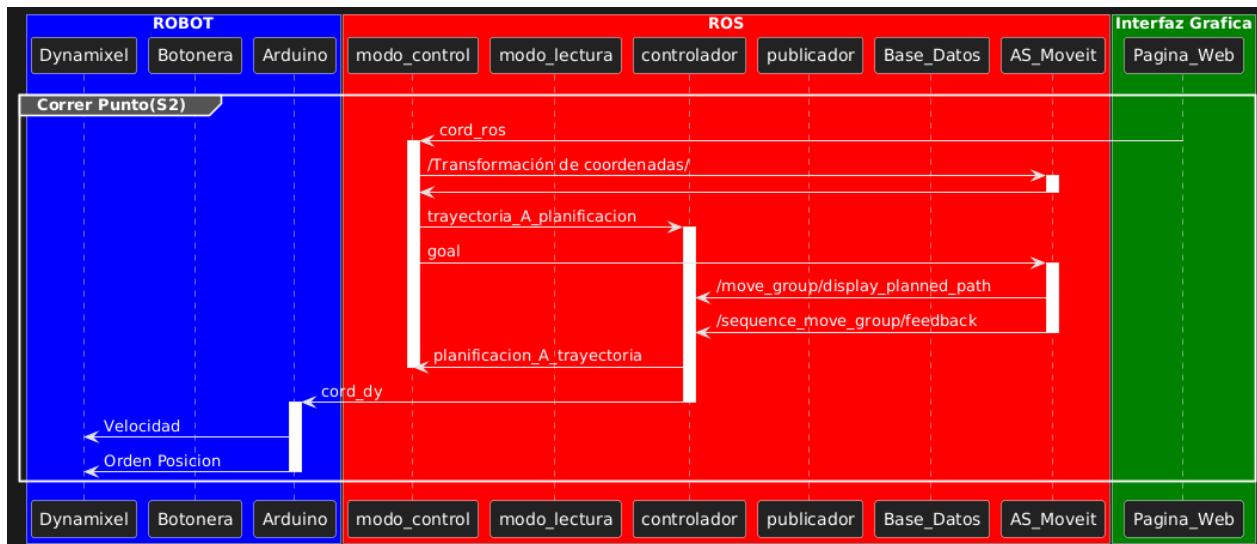


**“Botón agregar punto a la rutina”( 3.4.6):** funciona igual que **“Botón agregar punto real a la rutina”( 3.4.2)** salvo que utiliza las coordenadas de **“Inputs coordenadas angulares”( 3.4.8)**



**3.4.4-Boton ejecutar punto:** 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:4rad/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”



The screenshot shows a software interface with two main sections. On the left is a dialog box for entering coordinates, and on the right is a table of saved points.

**Left Side (Coordinate Entry Dialog):**

- Top Bar:** Contains "Coordenadas" with a save icon and an arrow icon.
- Real Coordinates Section:** Displays "XYZ: 200, 200, 200 / 200°, 200°, 200°" and "Ang: 120°, 120°, 210°, 120°, 120°, 120°".
- Ejecutar Button:** A blue button labeled "Ejecutar" with a save icon and an arrow icon.
- Cartesian Coordinates Section:** Contains input fields for "cord x" (0.00), "qx" (0.00), "cord y" (0.00), "qy" (0.00), "cord z" (0.00), and "qz" (0.00).
- Angular Coordinates Section:** Contains input fields for "ang 1" (0.00), "ang 4" (0.00), "ang 2" (0.00), "ang 5" (0.00), "ang 3" (0.00), and "ang 6" (0.00).
- Characteristics Section:** Contains dropdown menu "PTP", "Velocidad" (10), "Ratio" (0), and "Nombre" input field.

**Right Side (Points Database):**

Num	Nombre
1	Punto A
2	Punto B
3	Punto C
4	Punto A
5	Punto B
6	Punto C
7	Punto A
8	Punto B

Annotations with green numbered circles point to specific elements:

- 3.4.1: Points to the "Coordenadas" button in the top bar.
- 3.4.2: Points to the arrow icon in the top bar.
- 3.4.3: Points to the "XYZ" text in the Real Coordinates section.
- 3.4.4: Points to the "Ejecutar" button.
- 3.4.5: Points to the arrow icon next to the "Ejecutar" button.
- 3.4.6: Points to the "Ang" text in the Real Coordinates section.
- 3.4.7: Points to the "cord x" input field.
- 3.4.8: Points to the "ang 1" input field.
- 3.4.9: Points to the "PTP" dropdown in the Characteristics section.
- 3.4.10: Points to the "Velocidad" input field.
- 3.4.11: Points to the "Ratio" input field.
- 3.4.12: Points to the "Nombre" input field.
- 3.2.3: Points to the "Nombre" column header in the table.

### La 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

The screenshot shows the "PUNTOS" section of the software. It includes a list of saved points, a detailed view of a selected point, and a table of stored routines.

**Left Panel (PUNTOS):**

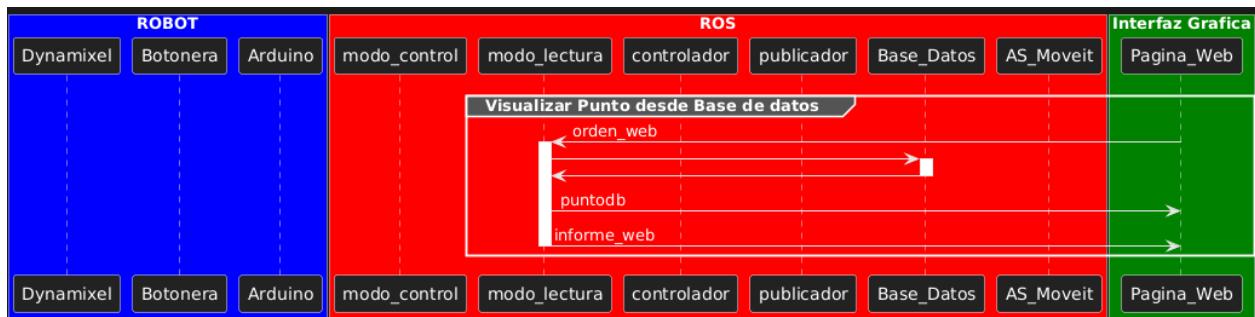
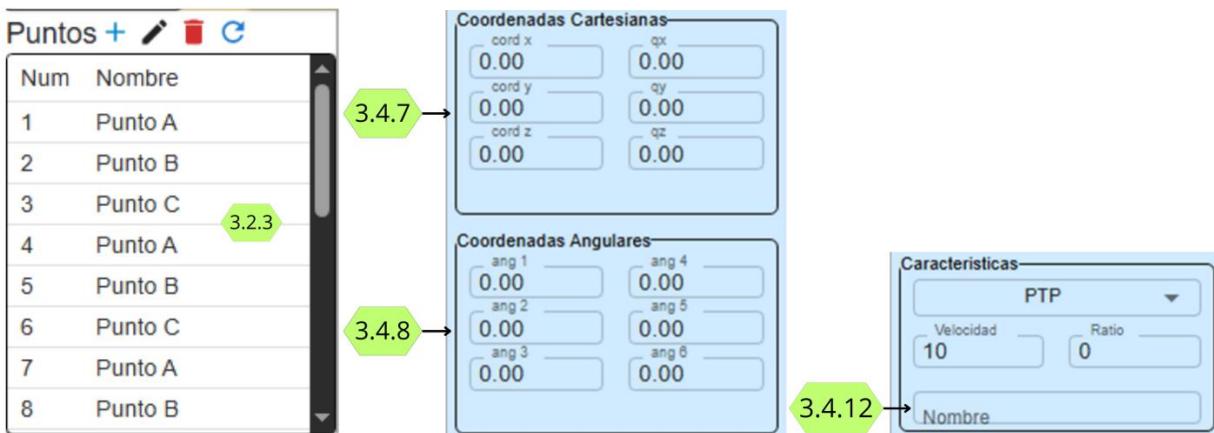
- Top Bar:** Buttons for "PUNTOS", "RUTINAS", "INICIAR PROGRAMA", "CERRAR PROGRAMA", "ESCRITURA", and "LECTURA".
- Point List:** Shows a list of points with columns "Num" and "Nombre".
- Detailed View:** Shows a "Coordenadas" section with "XYZ: 200, 200, 200 / 200°, 200°, 200°" and "Ang: 120°, 120°, 210°, 120°, 120°, 120°". Below it are sections for "Rutinas" and "Características".

**Right Panel (RUTINAS):**

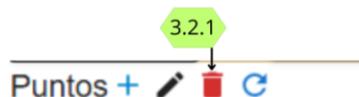
Rutina	Nombre de	Ejecutar	Plan	Edt	Cor	Wait
1	Inicie	0, 0, 0, 0, 0, 0	10	0	PTP	
2	Punto A	10, 20, 30, 0, 0, 0	50	50	LIN	
3	ir a Torno					
4	Punto B	10, 20, 30, 0, 0, 0	50	50	LIN	
5	T2					

Annotation 3.2: Points to the "Nombre" column header in the table.

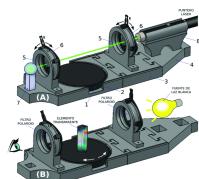
**3.2.3-Lista de Puntos Guardados:** aquí se muestran los puntos guardados en la base de datos. En esta el usuario puede cliquear cualquiera y los datos de este se mostraran en los “**Inputs coordenadas cartesianas**”( 3.4.7) y “**Inputs coordenadas angulares**”( 3.4.8) y “**Nombre del punto**” (3.4.12)

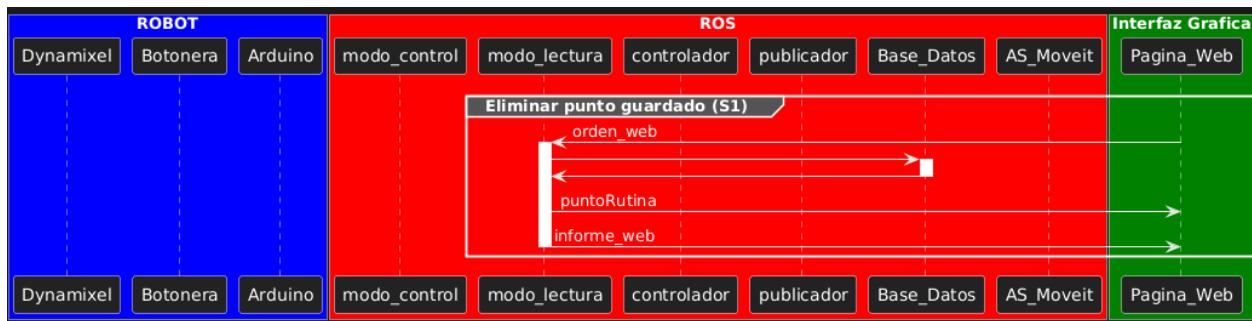


**3.2.1-Boton eliminar punto guardado:** manda la orden al controlador de borrar de la base de datos el punto seleccionado previamente, si este no existe o genera un error se informara al usuario por medio de una “ventana emergente”. [FIGURA: ].

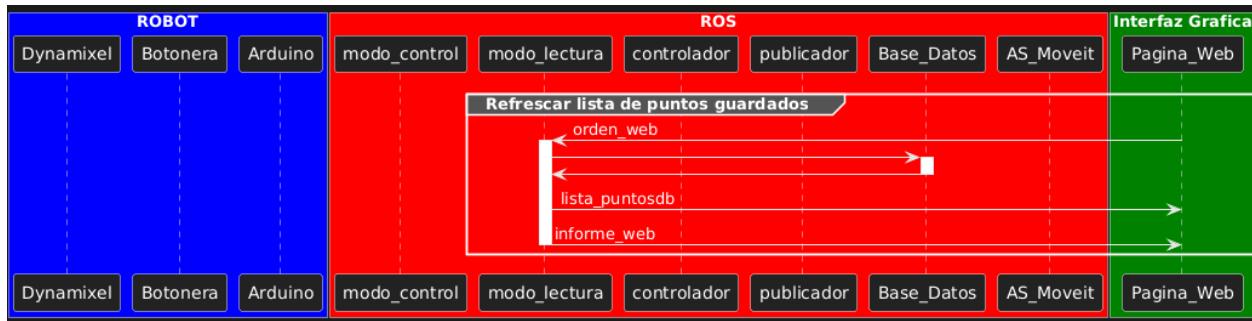


[FIGURA X]



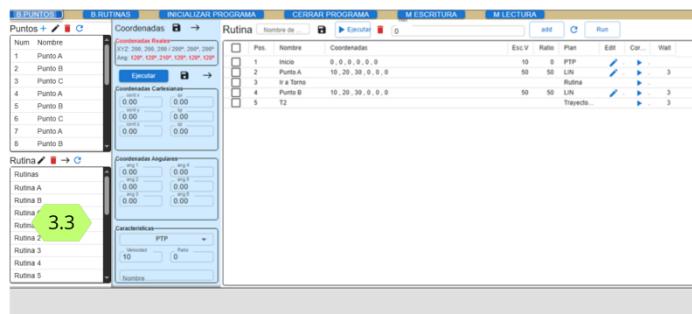


**3.2.2-Boton refreshar lista de puntos guardados:** sincroniza los datos con los de la base de datos por si ha habido un error y la página web no está mostrando los datos reales



### 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**



**3.3.5-Lista de rutina guardadas:** 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 sino es así

informa en **Pantalla de mensajes** (3.0) así el usuario no está trabajando con información adecuada



**3.3.1-Boton Editar Rutina:** 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.



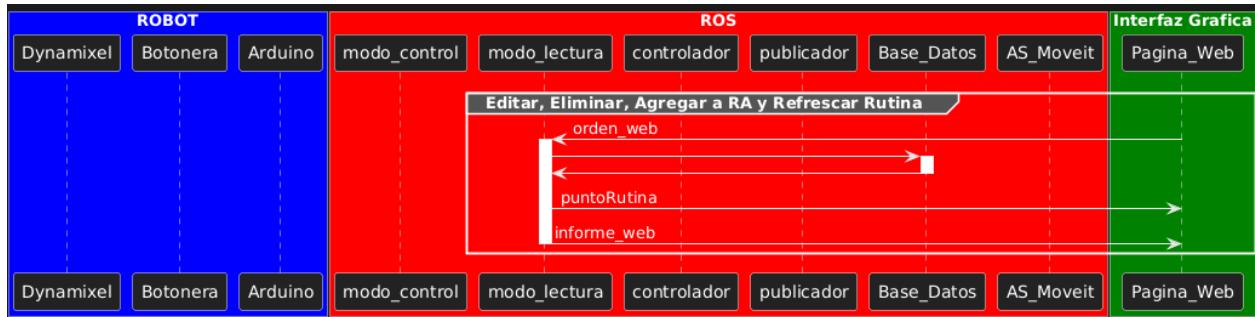
**3.3.2-Boton eliminar rutina guardada:** elimina la rutina seleccionada previamente, antes de esto la pagina pregunta por medio de una "**ventana emergente**". [FIGURA: ] para evitar borrar elementos por error.



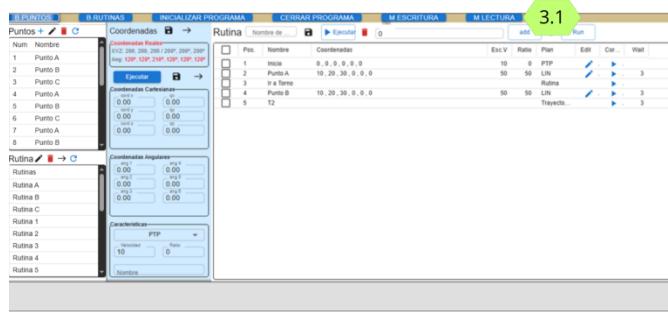
**3.3.3-Boton agregar rutina guardada a la rutina:** agrega a la **RA** la rutina como una instrucción



**3.3.4-Boton refrescar lista de rutinas guardadas:** 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



Barra de herramientas (3.1.x) [FIGURA: ] :



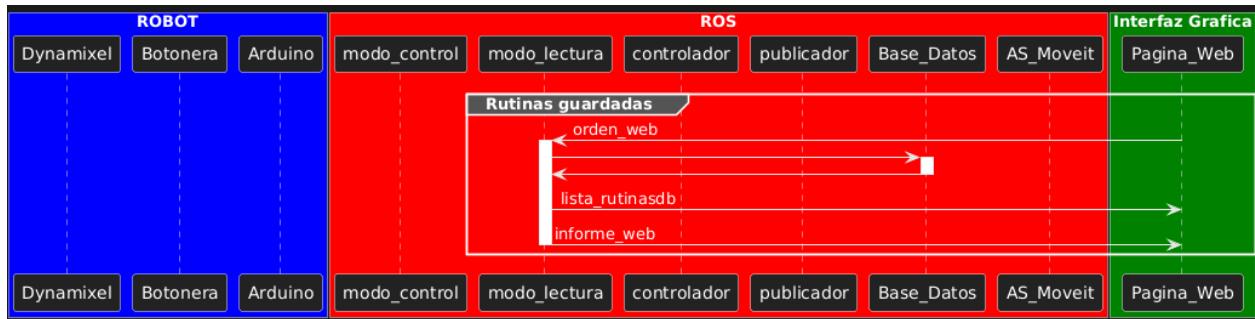
**3.1.1-Boton de puntos guardados:** permite mostrar y ocultar la **Sección de Puntos Guardados (3.2.x)**



**3.1.2-Boton de Rutinas guardadas:** permite mostrar y ocultar la **Sección de Rutinas Guardadas (3.3.x)**

3.1.2

B PUNTOS B RUTINAS INICIALIZAR PROGRAMA CERRAR PROGRAMA M ESCRITURA M LECTURA



**3.1.3-Boton de Inicialización de programa:** 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 iniciarla por comando, se configuro el sistema operativo para que pueda recibir ordenes de la pagina web y se realizo un programa de python que al recibir la orden al cliquear este botón se inicialice el controlador

3.1.3

B PUNTOS B RUTINAS INICIALIZAR PROGRAMA CERRAR PROGRAMA M ESCRITURA M LECTURA

**3.1.4-Boton de Cierre de programa:** 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.

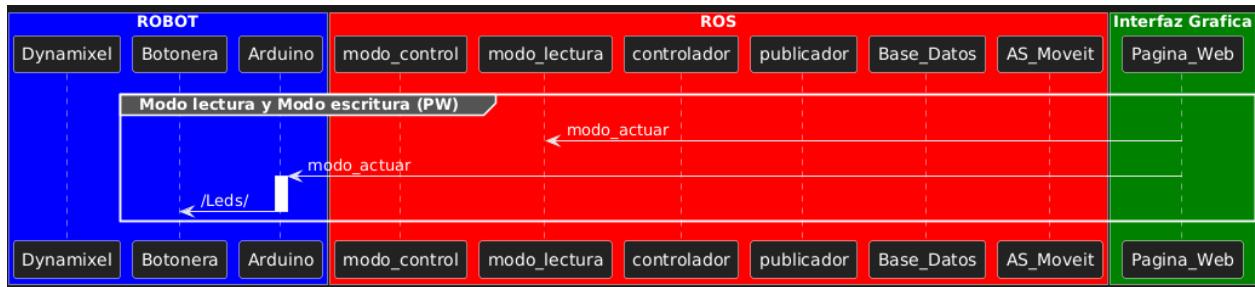
3.1.4

B PUNTOS B RUTINAS INICIALIZAR PROGRAMA CERRAR PROGRAMA M ESCRITURA M LECTURA

**3.1.5-Boton de Modo Control PW:** pone el robot en modo control se puede visualizar por el cambio de color (azul) en la pagina web, y confirmar al encenderse el **1.5-Led Modo Control**.

3.1.5

B PUNTOS B RUTINAS INICIALIZAR PROGRAMA CERRAR PROGRAMA M ESCRITURA M LECTURA

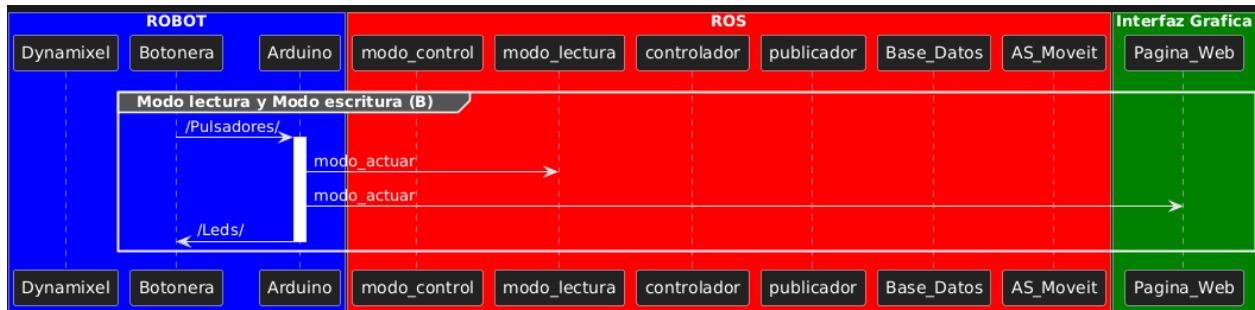
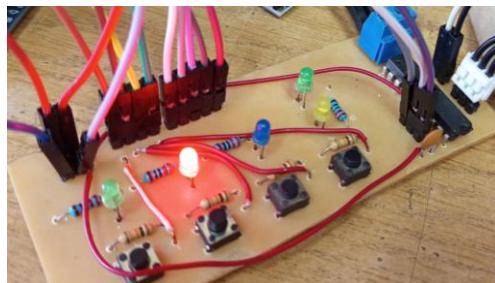


**3.1.6-Boton de Modo Lectura PW:** pone el robot en modo Lectura se puede visualizar por el cambio de color (verde) en la pagina web, y confirmar al encenderse el **1.6-Led Modo Lectura**.

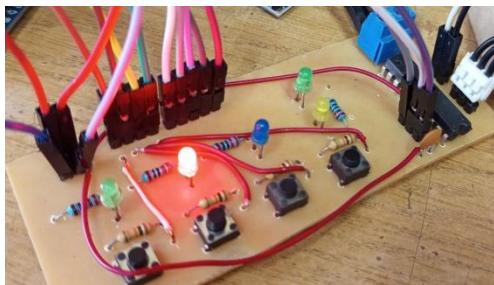


**Botonera de control (1.x)** [FIGURA: ]

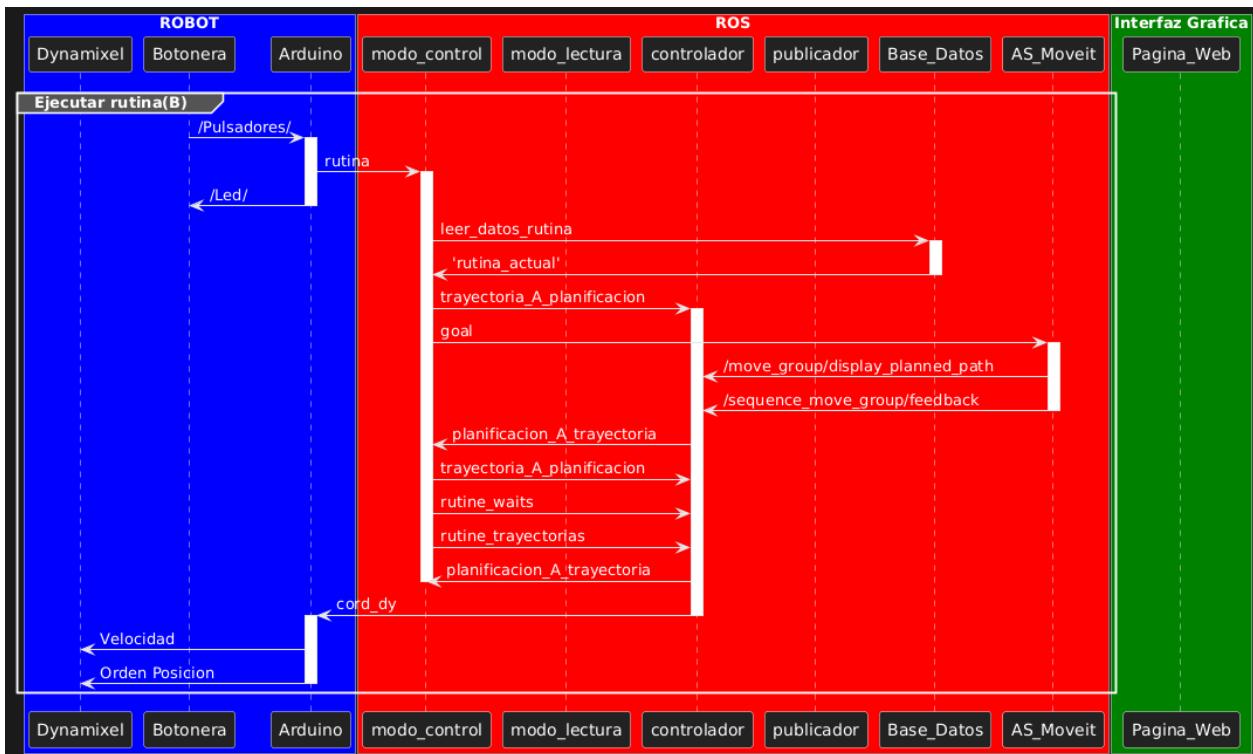
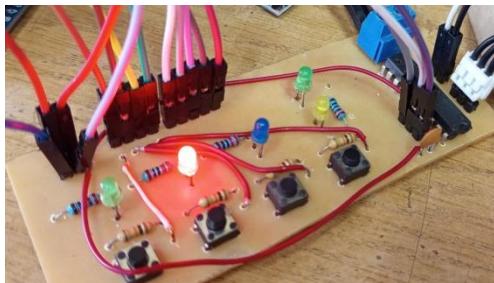
**1.1-Boton Modo Control B:** pone el robot en modo control se puede encendiendo **1.5-Led Modo Control**. El cambio también se visualiza en la pagina web



**1.2-Boton Modo Lectura B:** pone el robot en modo Lectura se puede encendiendo **1.6-Led Modo Lectura**. El cambio también se visualiza en la pagina web

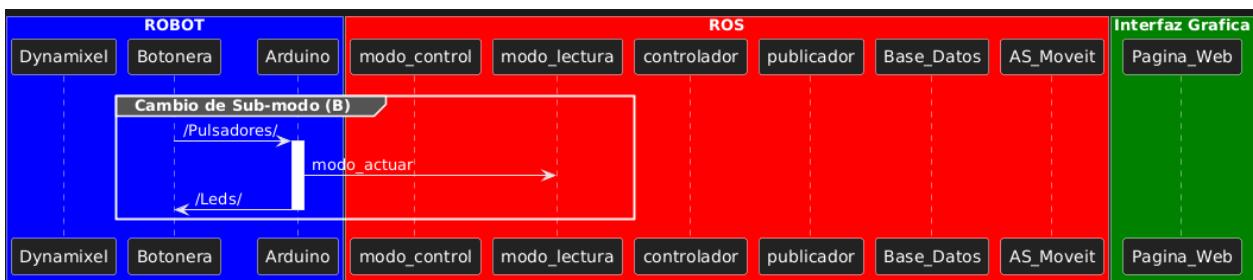
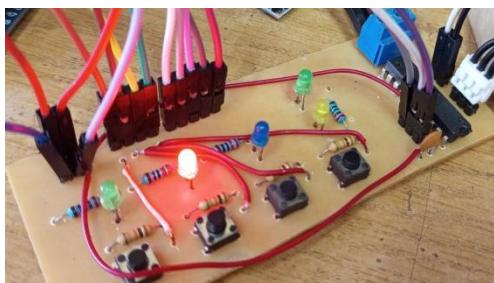


**1.3-Boton Ejecutar Rutina:** manda la orden al controlador para que ejecute la RA



**1.4-Boton Sub-Modos:** al apretarlo cambia de sub-modo, de punto a trayectoria y viceversa encendiendo y apagando los **1.8-Led Sub-Modo Punto** y **1.9-Led Sub-Modo Trayectoria** según corresponda.

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



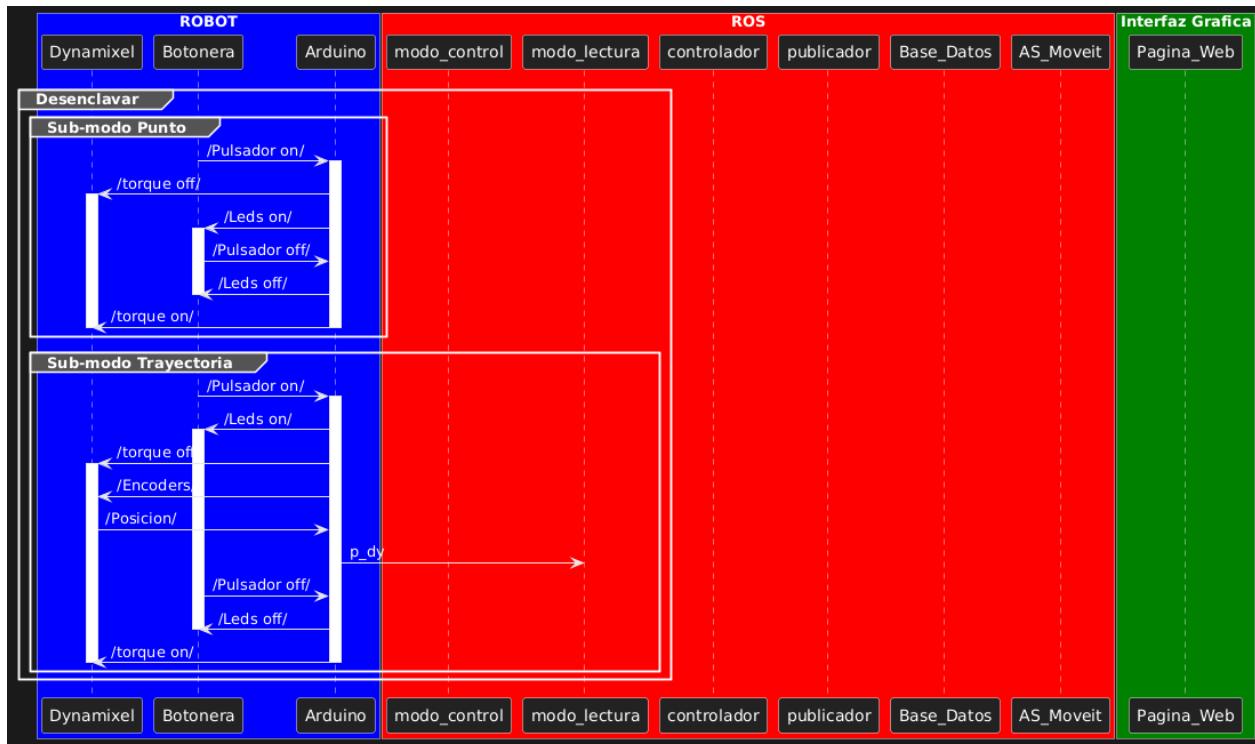
### **Botonera de Lectura (2.x)**

Todos los elementos de esta placa solo reaccionan en “modo lectura”

**2.1-Boton de Desenclavar:** mientras se mantenga apretado desenclava los motores permitiendo que el usuario pueda mover el robot con la mano, si se esta en el sub-modo trayectoria envía al controlador las posiciones de los motores cada 0.05 segundos para que los almacene hasta que sean guardados con en la base de datos con **2.2-Boton de Guardado** o eliminados al presionar **1.4-Boton Sub-Modos** (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)

Mientras se mantenga apretado enciende el **2.3-Led de Desenclavar**





**2.2-Boton de Guardado:** 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.

Al apretarlo enciende el **2.4-Led de Guardado**



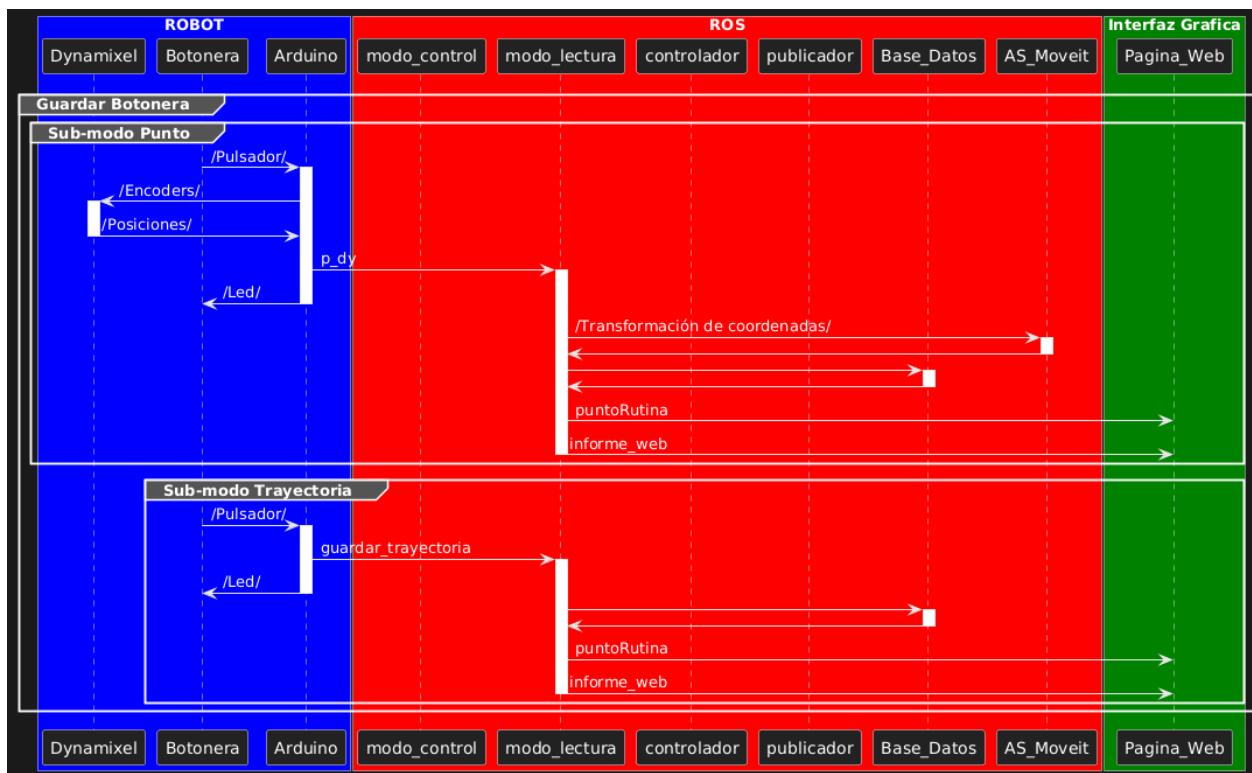


Tabla N°1. Abreviaturas

Concepto	Abreviatura
alternating current	ac
analog-to-digital	A-D, A/D
audio frequency*	AF
automatic frequency control*	AFC
automatic gain control*	AGC
amplitude modulation	AM
avalanche photodiode	APD
antireflection*	AR

Fuente: adaptación de [2]

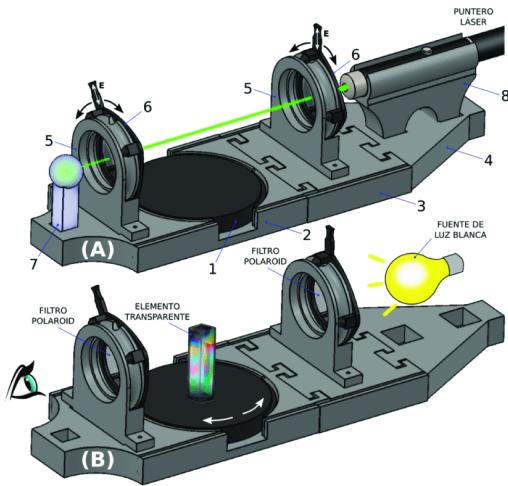


Figura N°1. Esquema de experiencia sobre polarización.

Fuente: tomado de [4]

## **CAPITULO X: Análisis de Costos**

El Análisis de Costos, conlleva un análisis pormenorizado de los costos involucrados en el proyecto. Puede contar con análisis comparativos, márgenes de rentabilidad, estudios de amortización, etc.

## **CAPITULO X: Estudio de Impacto Ambiental**

El Estudio de Impacto Ambiental, comprende la identificación, evaluación y descripción de los impactos ambientales que producirá el proyecto en su entorno en caso de ser ejecutado, y la descripción de la o las acciones que se ejecutarán para impedir o minimizar sus potenciales efectos adversos. Asimismo, deberá verificarse el cumplimiento de toda la normativa que deba cumplir la temática abordada.

## **CAPITULO X: Conclusiones**

Son las interpretaciones finales que recopilan los datos del trabajo, describe lo que se obtuvo, qué se logró y cuáles son los resultados.

## Glosario

WIMAX	Técnica de modulación FDM (empleada por el 802.11a y el 802.11g) para transmitir grandes cantidades de datos digitales a través de ondas de radio. OFDM divide la señal de radio en múltiples subseñales más pequeñas que luego serán transmitidas de manera simultánea en diferentes frecuencias al receptor. OFDM reduce la cantidad de ruido (crosstalk) en las transmisiones de señal.
Abstracción	Característica principal de la programación orientada a objetos que se refiere a la capacidad de que un objeto cumpla sus funciones independientemente del contexto en el que se lo utilice; o sea, un objeto "cliente" siempre expondrá sus mismas propiedades y dará los mismos resultados a través de sus eventos, sin importar el ámbito en el cual se lo haya creado.
AES	De las siglas Advanced Encryption Standard o estándar de encriptación avanzada para redes inalámbricas de área local, establecido en la 802.11i, ofrece un nivel de seguridad mayor que el encontrado en el actual estándar de seguridad WPA (Wi-Fi Protected Access).
Agenda Electrónica	Dispositivo con funciones limitadas a anotaciones, contactos, calendario y, en ocasiones, correo electrónico.

## Referencias Bibliográficas

- [1] "IEEE Reference guie v11.12.2018". IEEE Periodicals Transactions/Journal Departament, 2018. [En línea]. Disponible en: <https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf>
- [2] "IEEE Editorial Style Manual", *IEEE Author Center Journals*, 2022. <https://journals.ieeeauthorcenter.ieee.org/create-your-ieee-journal-article/create-the-text-of-your-article/ieee-editorial-style-manual/> (consultado el 10 de septiembre de 2022).
- [3] R. Hernández Sampieri, C. Fernández Collado, y P. Baptista Lucio, *Metodología de la investigación*, 5a ed. México, D.F: McGraw-Hill, 2010.
- [4] R. Peyton, D. Presti, J. H. Martínez Valdiviezo, F. Videla, y G. A. Torchia, "Desarrollo de experiencias para la enseñanza y difusión de la Óptica con impresión 3D [Not available in English]", en *2020 IEEE Congreso Bienal de Argentina (ARGENCON)*, dic. 2020, pp. 1–6. doi: 10.1109/ARGENCON49523.2020.9505322.

## Anexo/s

El documento puede incluir anexos que añadan elementos no esenciales, relativos al tema tratado. Se enumeran sucesivamente. Los anexos contienen información generada por el/la estudiante.

### Anexo 1

Tabla Nº1. Abreviaturas

Tipo de msg	indicador	Tipo de dato
punto_correr	descriccion	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 Nº1. Abreviaturas

Topico	Tipo de msg	Para que se usa	Publicador	Subscriptor
guardar_trayectoria	Bool	Bandera para guardar una Trayectoria	arduino	modo_lecatura
/move_group/display_planned_path	DisplayTrajectory	Plan generado por MoveIt	Moveit	modo_control
/sequence_move_group/feedback				controlador_cobot
	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 Guardadas	modo_lectura	Pagina Web
modo_actuar	Bool	Indicador del modo del robot(Control-Lectura)	modo_lectura	modo_lectura
			Pagina Web	Pagina Web
			arduino	arduino
orden_web	punto_web	manda la orden de 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 ejecucion 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	coordenadas 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 ejecucion de la rutina	modo_control	modo_control
			arduino	arduino
			Pagina Web	
routine_trayectorias	trayectorias	nombre de las trayectoria y posicion a ejecutar de la rutina	modo_control	controlador_cobot
routine_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	Indicador 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 ejecucion rutinas y puntos entre nodos "controlador_cobot" y "modo_control"	modo_control	controlador_cobot

### Anexo 3

Tabla Nº1. Abreviaturas

Programa	Nodo	Topico	USO	Tipo de msg
trayectoria 37	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
		routine_waits	p	waits
Modo lectura 18	modo_lectura	routine_trayectorias	p	trayectorias
		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
publicacion _posicion_ real3	publicador _posicion_ rea	modo_actuar	sp	Bool
		pos_dy	s	Int16MultiArray
controlador _rutina_pr ueba4	controlado r_cobot	joint_states	p	JointState
		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
		routine_waits	s	waits
Pagina Web		routine_trayectorias	s	trayectorias
		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

## **Apéndice/s**

Podrán incluirse apéndices que añadan elementos no esenciales, pero que son factibles de considerar en alguna oportunidad, al ser de algún modo relativo al tema tratado. Deben ubicarse luego de los anexos y se enumerarán consecutivamente. Los apéndices contienen información generada por terceros.