

Proyecto Final de Estudios

INGENIERÍA EN MECATRÓNICA

FACULTAD DE INGENIERÍA - UNIVERSIDAD NACIONAL DE CUYO

ROBOT PARALELO PICK-AND-PLACE CON VISIÓN ARTIFICIAL PARA LA PREPARACIÓN DE PEDIDOS

Director: Ing. Eric Sanchez

Emanuel Agüero, Agustín Lezcano
Mendoza, Argentina

2026

Agradecimientos

A nuestras familias y seres queridos, por acompañarnos en cada paso de este trayecto con paciencia y cariño incondicional.

A nuestros compañeros y futuros colegas, con quienes transitamos aulas y pasillos.

A nuestro Director de Proyecto, Eric Sanchez, por su generosa guía y por representar para nosotros un referente de integridad y vocación profesional.

A la universidad pública, que nos brindó las herramientas y el espacio para formarnos como profesionales al servicio de la sociedad.

A la Asociación de Mecatrónica, que nos brindó un espacio para compartir ideas y aprender entre compañeros.

Resumen

Se desarrolló un sistema robótico paralelo pick-and-place capaz de detectar y trasladar objetos utilizando visión artificial. El proyecto integra cinemática, control de movimiento, planificación de trayectorias y un sistema de visión que identifica objetos en una cinta transportadora simulada. La comunicación entre los módulos se implementó con ROS2, coordinando el procesamiento de visión y el control del STM32.

Palabras clave: Robot paralelo, pick-and-place, cinemática, Denavit-Hartenberg, visión artificial, YOLOv11, ROS2.

Abstract

A parallel pick-and-place robotic system capable of detecting and moving objects using artificial vision was developed. The project integrates kinematics, motion control, trajectory planning, and a vision system that identifies objects on a simulated conveyor belt. Communication between modules was implemented with ROS2, coordinating vision processing and STM32 control.

Keywords: Parallel robot, pick-and-place, kinematics, Denavit-Hartenberg, artificial vision, YOLOv11, ROS2.

Índice

1. Introducción	7
2. Objetivos	7
3. Estado del arte	8
4. Funcionamiento general	9
5. Herramientas utilizadas en el proyecto	10
6. Diagrama de conexiones	11
7. Descripción del hardware	12
7.1. Microcontrolador STM32F446RE	12
7.2. CNC Shield	12
7.3. Driver de motores	13
7.4. Sensor de ángulo	14
7.5. Multiplexor I2C TCA9548A	15
8. Esquema tecnológico	15
9. Cinemática	16
9.1. Denavit-Hartenberg	16
9.2. Cinemática directa	16
9.3. Cinemática inversa	17
10. Mejoras mecánicas	20
10.1. Descripción general	20
10.2. Datos de diseño y conteo de dientes	21
10.3. Condición de distancia entre ejes	22
10.4. Relaciones de transmisión y resolución angular	22
10.5. Sensado de ángulos	23
10.6. Cambios en el modelado cinemático	24
11. Circuito electrónico del electroimán	24
12. Visión artificial	25
12.1. Dataset y entrenamiento	26
12.2. Preparación del dataset	27
12.3. Entrenamiento en Google Colab	27
12.4. Parámetros de entrenamiento: <i>epochs</i> e <i>imgsz</i>	28
12.5. Evaluación del modelo	29
12.6. Inferencia y detección de objetos	29
12.7. Cálculo de posición del objeto detectado	30

13. Desarrollo del Firmware	31
13.1. Arquitectura general	31
13.2. Módulos básicos	32
13.2.1. Programación motores	32
13.2.2. Programación sensores de ángulo	32
13.2.3. Programación cinemática	33
13.2.4. Generación de Trayectorias sincronizadas	33
14. Integración de ROS, microROS y FreeRTOS en Sistemas Embebidos	36
14.1. ROS	36
14.2. microROS	39
14.3. FreeRTOS	39
14.4. Integración: ROS2 – microROS – FreeRTOS	40
14.5. Flujo de información en trayectorias	42
15. Ensayos realizados	44
15.1. Ensayo de carga electroimán	44
15.2. Ensayo de calibración de sensores de ángulo	44
15.3. Ensayos cinemáticos	45
15.4. Comunicación ROS2/microROS	45
15.5. Ensayo de Área de trabajo del robot	46
15.6. Ensayo de visión artificial	49
15.6.1. Detección del patrón y cálculo de la relación mm/px	49
15.6.2. Ensayo de detección mediante YOLOv11n	49
15.7. Ensayo de trayectoria	50
15.8. Ensayo completo	53
16. Futuras mejoras	57
17. Conclusiones	59
18. Anexo	61
18.1. Teorema del coseno	61
18.2. Colas FreeRTOS	61
18.3. Interfaces internas de ROS	62
18.4. Task de microROS	62
18.5. ROS_DOMAIN	63
18.6. Función de control de trayectoria	64

Índice de figuras

1.	Robots con visión artificial	8
2.	Foto real del Robot	9
3.	Diagrama de conexiones	11
4.	Microcontrolador STM32 Núcleo	12
5.	Placa CNC Shield	12
6.	Driver DRV8825 con sus conexiones	13
7.	Modos de conexión para realizar micropasos	13
8.	Módulo de encoder magnético AS5600	14
9.	Módulo de encoder magnético AS5600	15
10.	Multiplexor I ² C TCA9548A	15
11.	Diagrama general	16
12.	Referencias del método geométrico	18
13.	Esquema de eslabones simplificado	18
14.	Dimensiones de interés para cálculo de q_4	19
15.	Comparación vista lateral de versión con servo vs con motores Paso a Paso	20
16.	Comparación vista en perspectiva de versión con servo vs con motores Paso a Paso	21
17.	Alineamiento axial entre sensor AS5600 e imán montado sobre eje de articulación	24
18.	Esquemático y placa electrónica de control del solenoide	25
19.	Imágenes del dataset	26
20.	Predicción de modelo YOLOv11	28
21.	Curvas de resultado de inferencia de YOLOv11n	29
22.	Esquema de posición Robot-Cámara-Objeto	31
23.	Posiciones, velocidades y aceleraciones del parámetro s	35
24.	Posiciones, velocidades y aceleraciones articulares	35
25.	Trayectoria en el espacio de trabajo interpolada con B-Spline	36
26.	Gráfico de nodos ROS	37
27.	Mapa conceptual de las conexiones entre el orquestador y el esclavo	39
28.	Diagrama de conexiones entre nodos	41
29.	Flujo de datos en trayectorias	43
30.	Espacio de trabajo teórico	47
31.	Espacio de trabajo articular con restricción $z = 0$	47
32.	Espacio de trabajo articular (Zoom)	48
33.	Espacio de trabajo real, plano XZ	48
34.	Ensayo de calibración para obtener relación mm/px	49
35.	Ensayo de detección de herramientas y posicionamiento respecto origen del robot	50
36.	Comparación de trayectorias real y teórica con sus respectivas ganancias	52
37.	Errores de posición con sus respectivas ganancias	53
38.	Interfaz de usuario	55
39.	Foto real del robot realizando <i>pick-and-place</i>	56
40.	Triángulo general donde se aplica el Teorema del Coseno.	61
41.	Funcionamiento de colas	62
42.	Interfaces internas de ROS	63

Índice de tablas

1.	Parámetros Denavit-Hartenberg	16
2.	Frecuencias de PWM y velocidades en motores/eslabones	32
3.	Descripción de tópicos más relevantes	42
4.	Comparación de máximos valores absolutos de errores en la trayectoria respecto a K_p	51
5.	Protocolo de inicio para el sistema distribuido ROS2 / microROS	54

1. Introducción

El proyecto desarrollado consiste en el diseño e implementación de un sistema robótico paralelo para operativa *pick-and-place*, cuyo objetivo principal es trasladar objetos desde un punto inicial (A) hasta un punto final (B) a partir de su detección mediante técnicas de visión artificial. El sistema está orientado a manipular objetos que arriban desde lo que se simula ser una cinta transportadora, identificarlos dentro del espacio de trabajo y posicionarlos de manera precisa en un contenedor correspondiente a un pedido, simulando un flujo típico de procesos logísticos modernos.

Para esto, se realizó la modificación estructural y funcional de un robot modelo MK2, reemplazando sus servomotores originales por motores Paso a Paso en todas sus articulaciones, con el fin de mejorar la precisión de posicionamiento y el control del movimiento. El efector final incorpora un electroimán que permite la sujeción de piezas metálicas durante el ciclo operativo.

El desarrollo abarca múltiples áreas: la modificación estructural hacia motores Paso a Paso; el cálculo de la cinemática directa e inversa del manipulador; la integración y calibración de los motores; y la planificación de trayectorias que permitan un movimiento suave, eficiente y seguro del robot. Asimismo, se integra un sistema de visión artificial basado en algoritmos de inferencia para la detección y localización de objetos en el entorno.

Finalmente, se implementa una arquitectura de comunicación mediante **ROS2**, por medio de la cual se establecen nodos dedicados a la interacción entre la computadora encargada del procesamiento de visión y el envío de consignas de posición, y el microcontrolador STM32 responsable del control de los motores. Esta infraestructura permite el intercambio confiable de información entre los distintos módulos del sistema y constituye la base para la coordinación del proceso completo de manipulación robótica.

2. Objetivos

Se busca llegar a un prototipo funcional, y que el sistema esté modularizado de tal manera que el proyecto pueda ser continuado por otros alumnos interesados. Este desarrollo, enmarcado en el Proyecto Final de Estudios, se realizó con recursos del laboratorio Rosetta.

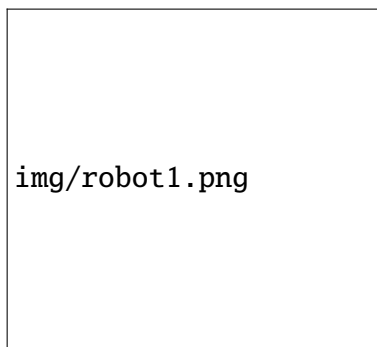
También, a tener un software bien definido y robusto para que luego se pueda cambiar el robot, si así se desea, y el enfoque del trabajo sea en el hardware o en la parte mecánica, quitando el peso de realizar un trabajo desde cero. Los objetivos específicos se listan a continuación:

- Implementación y desarrollo del firmware de bajo nivel para el control de cada motor.
- Desarrollo basado en la extensibilidad y flexibilidad del código para un posible desarrollo futuro.
- Utilización del protocolo MQTT para la comunicación entre usuarios y servidor, de manera que se pueda expandir fácilmente el número de usuarios del sistema.
- Integración de los subsistemas de control y envío/recepción de consignas.
- Diseño de un algoritmo de Visión Artificial para la detección de objetos.
- Posicionamiento para cumplir con los pedidos y confirmación de tarea completada.

3. Estado del arte

En los últimos años, la integración de visión artificial en sistemas robóticos industriales ha impulsado manipuladores más autónomos, precisos y adaptables, destacándose diversos desarrollos que han marcado el rumbo de esta tecnología, como se muestra en la Figura 1.

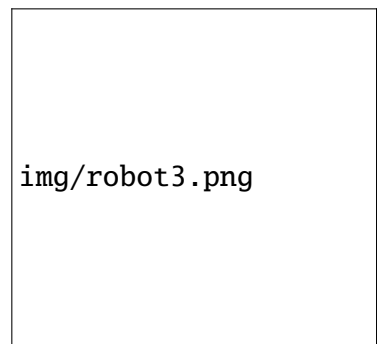
- **Dex-Net** (UC Berkeley) ha sido una de las referencias más influyentes en el área. Su contribución principal se basa en el uso de grandes bases de datos sintéticas para entrenar redes neuronales capaces de predecir puntos de agarre fiables incluso en objetos desconocidos. Este enfoque demostró que la simulación masiva combinada con *deep learning* puede ofrecer soluciones robustas para tareas de *bin picking*.
- **DoraPicker** (Dora Robotics) se orienta al ámbito logístico, integrando visión 3D y algoritmos de segmentación para identificar y extraer objetos variados de estanterías o contenedores. Su arquitectura evidencia la viabilidad de sistemas completamente autónomos para manipular productos de diferentes formas y tamaños.
- **Cartman**, desarrollado para el Amazon Robotics Challenge, propone un diseño cartesiano combinado con algoritmos de reconocimiento y planificación eficientes. Su participación demostró que una integración cuidadosa entre percepción y mecánica puede mejorar la confiabilidad en la manipulación de objetos desordenados o deformables.
- **Cambrian Vision** representa un enfoque industrial consolidado, ofreciendo módulos de visión 3D y algoritmos de IA listos para su incorporación en líneas de producción. Su fortaleza está en la detección rápida y la reducción del tiempo de configuración por parte del operador.



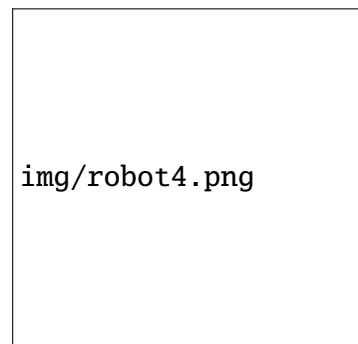
(a) Robot Dex-Net



(b) Robot DoraPicker



(c) Robot Cartman



(d) Robot Cambrian Vision

Figura 1: Robots con visión artificial

4. Funcionamiento general

El funcionamiento integral del sistema robótico desarrollado se basa en la interacción coordinada entre tres módulos principales: el sistema de visión artificial, el orquestador, y el robot físico equipado con motores Paso a Paso (ver Figura 2). Cada uno de estos componentes cumple un rol esencial dentro del ciclo operativo del robot *pick-and-place*, permitiendo que el flujo completo (desde la detección del objeto hasta su deposición final) se lleve a cabo de manera precisa, repetible y segura.

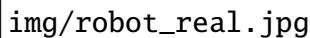


Figura 2: Foto real del Robot

En primer lugar, el proceso se inicia con la captura de imágenes de la zona de trabajo mediante una cámara montada en una posición fija y debidamente calibrada respecto al robot. Esta cámara observa el área donde los objetos -provenientes de una simulación de cinta transportadora- ingresan al campo visual. A partir de estas imágenes, un modelo de visión artificial ejecuta un algoritmo de inferencia que identifica cada pieza de interés y determina la posición bidimensional del centro de su *bounding-box*. Esta localización inicial se encuentra en coordenadas de imagen (píxeles), por lo que, mediante la utilización de un factor de conversión mm/px obtenido en la etapa de calibración geométrica, se transforma dicha posición al sistema de referencia físico del robot. Adicionalmente, la altura (Z) del objeto se obtiene a partir de una base de datos que asocia cada categoría detectada con su correspondiente dimensión real.

Una vez determinada la ubicación tridimensional del objeto, esta información es transmitida

al módulo de planificación de movimiento implementado sobre la computadora principal, donde existe un orquestador que utiliza microROS para enviar comandos de posición al robot mediante comunicación serial. Actualmente, se utiliza el envío de consignas articulares, pero el robot es capaz de resolver la cinemática inversa y dar consigna a los motores. Además se envían los comandos de *homing* y parada de emergencia.

El microcontrolador es capaz de realizar los cálculos de cinemática directa e inversa del robot paralelo, permitiendo definir los ángulos articulares requeridos para alcanzar tanto la posición de recogida como la posición final donde el objeto será depositado. Para garantizar movimientos suaves y evitar esfuerzos innecesarios sobre la estructura, cada articulación sigue un perfil de velocidad que asegura aceleraciones y desaceleraciones controladas y limitadas a las velocidades y aceleraciones máximas de los motores. Este enfoque permite un desplazamiento eficiente, reduciendo vibraciones y mejorando la repetibilidad del posicionamiento.

El control de los motores Paso a Paso se realiza a través del microcontrolador STM32, el cual recibe las consignas articulares mediante nodos de comunicación basados en ROS2. Esta arquitectura distribuida facilita el intercambio de mensajes entre los distintos módulos, desacoplando el procesamiento intensivo de visión del control de bajo nivel de los actuadores. El STM32 se encarga de generar las señales de paso y dirección, ejecutar los perfiles de velocidad y supervisar el cumplimiento temporal de cada trayectoria articulada.

Finalmente, el efector final equipado con un electroimán realiza la sujeción y liberación de las piezas metálicas durante el ciclo de trabajo. Una vez que el robot alcanza la posición de destino, el sistema desactiva el electroimán para depositar el objeto en el contenedor asignado. Con esto concluye el ciclo operativo, tras lo cual el robot retorna a una posición segura de espera, quedando preparado para procesar el siguiente objeto detectado.

Se decidió utilizar ROS (Robot Operating System) en su versión 2 debido a su amplia adopción tanto en el ámbito académico como en la industria, donde se perfila como un estándar de facto. Al tratarse de una plataforma *open-source* y multiplataforma, cuenta con una comunidad activa y extensa que garantiza su mantenimiento, evolución y actualización continua. Pese a su nombre, no es un sistema operativo sino un SDK (Software Development Kit). Este conjunto de librerías y herramientas proporciona una arquitectura altamente modular, permitiendo desarrollar componentes para distintas plataformas y en diversos lenguajes de programación. Dado que ROS2 es agnóstico al lenguaje, el desarrollador puede concentrarse en la lógica de aplicación, mientras que la infraestructura de comunicación entre módulos es gestionada de manera transparente por ROS2 ¹.

5. Herramientas utilizadas en el proyecto

Para el desarrollo integral del proyecto se emplearon un conjunto de herramientas de diseño, hardware y software que permitieron abordar de manera coordinada los distintos aspectos mecánicos, electrónicos y de control del sistema.

- **Mecánica:** El diseño estructural del robot se realizó principalmente en *Solid Edge*, donde se modelaron las piezas individuales y se verificó el ensamblaje completo del robot. Para el diseño específico de engranajes se utilizó *Fusion 360* y *OrcaSlicer* como laminador de las piezas impresas en 3D.
- **Hardware:** El sistema de control se basa en el microcontrolador *STM32F446RE*, junto con una placa CNC Shield para el control de motores Paso a Paso. La medición de posición angular

¹En el anexo 18.3 hay más información acerca de la infraestructura ROS

en cada articulación se realizó mediante sensores magnéticos **AS5600**, conectados a través de un multiplexor I²C **TCA9548A**, permitiendo la lectura de múltiples dispositivos con una única interfaz de comunicación.

- **Software:** La programación del microcontrolador se desarrolló en *STM32CubeIDE*, donde se configuraron periféricos, temporizadores y tareas en tiempo real. El modelado y análisis cinemático inicial del robot, así como la verificación del espacio de trabajo, se llevaron a cabo en *MATLAB*. La implementación de la cinemática inversa, la planificación de trayectorias y la comunicación mediante sockets con el módulo de visión se desarrollaron en *Python*, integrando estos procesos dentro del orquestador del sistema. Para la detección de piezas y determinación del punto objetivo se utilizó el modelo *YOLOv11n*, entrenado con un *dataset* propio de piezas. El diseño de la placa electrónica destinada al control del electroimán se realizó en *KiCad*, incluyendo el esquemático y el ruteo de la PCB. Por último, se empleó *microROS*, que corre sobre *FreeRTOS*, el sistema operativo en tiempo real sobre el microcontrolador. También del lado de la computadora, se tiene a *ROS2*, que actuó como orquestador general del proyecto y permitió estructurar la comunicación entre los distintos módulos del sistema.

6. Diagrama de conexiones

El sistema se encuentra compuesto por una placa de desarrollo **STM32F446RE**, sobre la que se encuentra montada una **CNC Shield** para la conexión de drivers **DRV8825** de 3 motores Paso a Paso, 3 sensores **AS5600** para la medición de ángulos absolutos mediante efecto Hall, un multiplexor **TCA9548A** que permite gestionar múltiples dispositivos I²C con direcciones idénticas en un mismo bus, y una placa electrónica propia que permite la alimentación segura hacia el electroimán.

En la Figura 3 se puede observar el diagrama de conexiones con todos los elementos mencionados. Cabe aclarar que la placa electrónica para el accionamiento del electroimán se ha representado con un transistor NPN.

img/DiagramaConex.png

Figura 3: Diagrama de conexiones

7. Descripción del hardware

7.1. Microcontrolador STM32F446RE

La placa *Nucleo-F446RE* (ver Figura 4) incorpora el microcontrolador *STM32F446RE* de 32 bits, basado en arquitectura ARM Cortex-M4 de hasta 180 MHz, con 512 KB de memoria Flash y 128 KB de SRAM. Esta placa ofrece una amplia disponibilidad de periféricos de comunicación (UART, SPI, I²C, CAN) y una distribución de pines compatible con el formato Arduino, lo que facilita la integración con expansiones como la *CNC Shield*. El microcontrolador se encarga de la generación de señales PWM, la lectura de sensores por I²C, y la gestión del sistema de control.

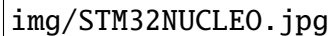
img/STM32NUCLEO.jpg

Figura 4: Microcontrolador STM32 Núcleo

7.2. CNC Shield

La *CNC Shield* (ver Figura 5) utilizada es una expansión diseñada originalmente para placas Arduino UNO, pero resulta compatible con la placa Nucleo-F446RE gracias a su distribución de pines y niveles lógicos de 3.3 V – 5 V. Esta placa permite la conexión directa de drivers de motores Paso a Paso, ofreciendo pines dedicados para STEP, DIR, y EN. Además, dispone de bornes para alimentación externa, jumpers para configuración de micropasos, y conectores auxiliares para finales de carrera. De esta forma, el STM32 puede controlar varios motores Paso a Paso mediante salidas digitales dedicadas, sin necesidad de circuitos adicionales.

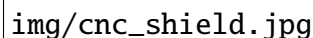
img/cnc_shield.jpg

Figura 5: Placa CNC Shield

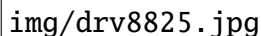
7.3. Driver de motores

El módulo *DRV8825* (ver Figura 6a) se utiliza como driver para el control de motores Paso a Paso bipolares desde el STM32. Este dispositivo integra la electrónica necesaria para generar las secuencias de corriente requeridas por las bobinas del motor, permitiendo controlar tanto la dirección de giro como el avance por pasos mediante dos señales digitales de entrada: DIR y STEP, respectivamente. Incorpora un regulador de corriente ajustable mediante un potenciómetro integrado, que permite limitar la corriente máxima por fase del motor, protegiendo al mismo y al driver ante sobrecorrientes. Para determinar el valor de corriente máxima por bobina (I_{max}), se debe medir con un voltímetro la tensión de referencia V_{ref} entre el terminal del potenciómetro y masa (GND), y luego aplicar la relación aproximada:

$$I_{max} = 2 \cdot V_{ref} \quad (1)$$

Esta ecuación constituye una aproximación útil para el dimensionamiento inicial, aunque el valor real puede variar ligeramente según tolerancias y condiciones de operación.

La conexión general del *DRV8825* con el sistema se muestra en la Figura 6b, donde se detallan las entradas de control, las salidas hacia el motor y la alimentación. Asimismo, en la Figura 7 se ilustran las configuraciones de los pines MODE0, MODE1 y MODE2 que permiten seleccionar el modo de micropaso deseado. Para este proyecto se conectaron los pines MODE0 y MODE1 al voltaje de alimentación del microcontrolador para activar el modo de 1/8 de paso.



(a) Driver DRV8825



(b) Conexiones DRV8825

Figura 6: Driver DRV8825 con sus conexiones

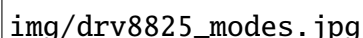


Figura 7: Modos de conexión para realizar micropasos

7.4. Sensor de ángulo

Para la medición del ángulo de cada eslabón se seleccionó el sensor *AS5600* (ver Figura 8). Este dispositivo corresponde a un sensor de posición angular sin contacto, basado en el principio de detección del campo magnético generado por un imán colocado sobre el eje de rotación. El sensor determina la orientación del campo magnético y entrega la posición angular absoluta con una resolución de 12 bits, equivalente a 4096 pasos por revolución. La elección de este sensor se basó en su alta resolución, facilidad de lectura mediante el bus I²C y buena inmunidad al ruido electromagnético.

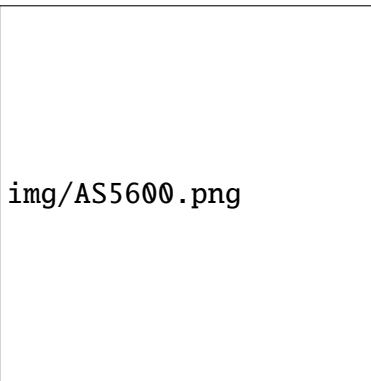


Figura 8: Módulo de encoder magnético AS5600

Sus principales características son:

- **Resolución:** 12 bits (4096 pasos/rev), equivalente a aproximadamente $0,0879^\circ/\text{paso}$
- **Rango angular:** programable; por defecto cubre $0-360^\circ$. Se puede configurar el ángulo máximo (MANG) y la posición cero (ZPOS)
- **Interfaces:**
 - I²C con dirección fija de 7 bits $0x36$.
 - Salida PWM, donde el ciclo de trabajo representa el ángulo medido (frecuencia configurable por registro)
- **Velocidad del bus I²C:** hasta 400 kHz

En el montaje de los ejes se instaló un imán de neodimio en el extremo de cada eje, ubicado frente al sensor *AS5600* de manera centrada. Se optó por utilizar el protocolo I²C por la fiabilidad en la medición y velocidad del bus. El valor angular bruto (*raw*) entregado por el módulo a través de I²C varía entre 0 y 4095, representando el rango completo de 0° a 360° de manera proporcional.

El registro de salida de ángulo está compuesto por dos bytes (MSB y LSB) que conforman el valor digital de 12 bits. El cálculo del ángulo en grados se obtiene mediante la expresión:

$$\text{angle_deg} = \frac{\text{raw}}{4096} \times 360^\circ \approx \text{raw} \times 0,0879^\circ \quad (2)$$

donde $\text{raw} \in [0, 4095]$ corresponde al valor digital leído del registro de salida.

En la Figura 9 se observa una representación esquemática del imán con sus polos (Norte y Sur) frente al sensor. Puede observarse que para un ángulo de 0° el sensor entrega un valor de

RAW_ANGLE = 0, para 90° el valor es RAW_ANGLE = 1024, para 180° RAW_ANGLE = 2048, y para 270° RAW_ANGLE = 3072. Cabe destacar que 360° coincide con 0°, y que las posiciones 0°–180° y 90°–270° presentan polaridades magnéticas invertidas, evitando así ambigüedades en la detección de posición.

img/AS5600_polos_magneticos.png

Figura 9: Módulo de encoder magnético AS5600

7.5. Multiplexor I2C TCA9548A

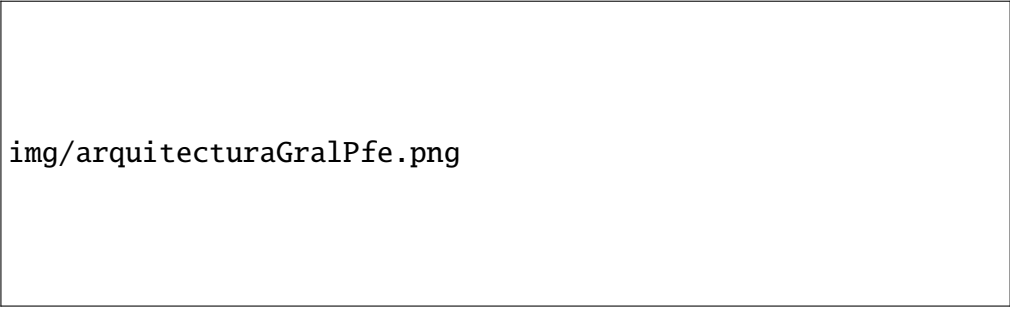
El TCA9548A (ver Figura 10) es un multiplexor I²C de 8 canales independientes que permite conectar hasta ocho dispositivos I²C con direcciones repetidas, seleccionando el canal activo mediante comandos enviados desde el microcontrolador principal. Cada canal actúa como una línea I²C aislada (SDA y SCL), evitando conflictos de dirección entre sensores idénticos como el AS5600. El STM32 se comunica con el TCA9548A mediante su interfaz I²C principal, y activa cada canal según el sensor que se desea leer. Este componente amplía la escalabilidad del sistema, permitiendo el uso de múltiples sensores sin modificar el hardware base.

img/tca9548a.jpg

Figura 10: Multiplexor I²C TCA9548A

8. Esquema tecnológico

En la Figura 11 se presenta un diagrama general simplificado que presenta los módulos principales del sistema robótico: el de visión, el orquestador (que hace uso de la información del módulo de visión) y el brazo robótico (que recibe las instrucciones de alto nivel del orquestador y realiza el control a bajo nivel).



img/arquitecturaGralPfe.png

Figura 11: Diagrama general

9. Cinemática

A continuación se presenta la resolución de los problemas de cinemática, fundamentales para poder traducir las consignas en ángulos de giro para los motores. Esto se logra mediante la cinemática inversa, es decir que se pasa del espacio de trabajo al espacio articular. Para más profundidad al respecto, se pueden consultar las siguientes referencias: [Barrientos et al., 2007], [Craig, 2006], [Corke, 2017], [Mark W. Spong, 2005], [Torres et al., 2002].

9.1. Denavit-Hartenberg

Teniendo en cuenta las dimensiones del robot, se describió la morfología del mismo siguiendo la convención Denavit-Hartenberg. Como resultado, se simplifica la estructura dividiendo al brazo en eslabones (obteniendo la Tabla 1), y se resuelve el problema de la cinemática directa a través de transformaciones homogéneas. Para esto se tuvieron en cuenta los eslabones activos; además se consideró un eslabón que actúa como articulación o grado de libertad virtual con el fin de simplificar los cálculos (se verá el desarrollo en la sección 9.3). Un desarrollo similar que puede ser consultado en [Zhou et al., 2020]. La ventaja de usar el método Denavit-Hartenberg es que permite simplificar el cambio de posición y orientación de un punto a otro en una concatenación de movimientos de traslación y rotación (dos rotaciones y dos traslaciones para cada desplazamiento desde un punto de origen a otro, en el sistema de referencia elegido).

θ	d	a	α
q_1	L1	0	$\pi/2$
q_2	0	L2	0
q_3	0	L3	0
q_4	0	L4	0

Tabla 1: Parámetros Denavit-Hartenberg

9.2. Cinemática directa

Cada “desplazamiento” entre el origen de un sistema de referencia y otro del eslabón siguiente se puede representar mediante una matriz de transformación homogénea.

$${}^{i-1}T_i = Rot(z_{i-1}, \theta_i) \cdot Tras(z_{i-1}, d_i) \cdot Tras(x_i, a_i) \cdot Rot(x_i, \alpha_i) \quad (3)$$

A su vez, considerando el origen del primer sistema de referencia y el del último (que puede coincidir con un extremo del robot, donde irá colocada la herramienta de ser necesario), se puede relacionar en el desplazamiento del punto de origen del sistema de referencia al extremo, siguiendo la fórmula siguiente:

$${}^{base}T_{extremo} = {}^{base}T_1 \cdot {}^1T_2 \cdot \dots \cdot {}^{n-1}T_n \cdot {}^nT_{extremo} \quad (4)$$

9.3. Cinemática inversa

El robot de este proyecto, es de tipo paralelo o de cadena cinemática cerrada. Para la resolución de la cinemática inversa, se ha optado por hacer una simplificación y considerar al mismo como cadena cinemática abierta. Como se menciona en la Sección 9.1, se considera un eslabón virtual. Esta articulación siempre mantiene la horizontalidad o paralelismo respecto al plano XY sobre el que está apoyada la base del robot, siempre que el robot se encuentre dentro del espacio de trabajo. Los eslabones pasivos o conducidos se pueden calcular de manera similar, pero no inciden en los cálculos para la cinemática inversa. Dado el funcionamiento del robot y para hacer una simplificación en los cálculos, solo se considera la posición del efector final, no la orientación.

Se nombran las articulaciones tales como se indican en la Figura 12a. Otra cosa a destacar es que no todas las articulaciones se mueven de manera directa, sino que algunas de ellas son conducidas, por ejemplo la articulación 4 depende de los movimientos de las articulaciones anteriores, y la articulación pasiva 3, depende del movimiento de la articulación activa 3'. Aquí se consideró que el eslabón 3 comienza en la intersección con el extremo final del eslabón 2, evitando añadir a los cálculos el eslabón activo y el pasivo vinculados. Se puede ver un detalle de esto en la Figura 13.

Dadas las limitaciones de movimiento (límites articulares) y la configuración del robot, que tiene restricciones físicas debido a la cadena cerrada, se llega a tener una única solución posible para un punto objetivo en el espacio, que es “codo arriba”.

Otra aclaración es que los ángulos q_i obtenidos no representan los ángulos de giro de sus respectivos motores (a excepción del eslabón 1), aunque se puede llegar al valor de los mismos fácilmente. Se los representará con α y β . Una vez hechas las consideraciones previas, se pasa a hacer la representación de los parámetros Denavit-Hartenberg (ver Tabla 1). Para la resolución del problema de cinemática inversa, se utiliza el método geométrico. Se considera la estrategia de desacople cinemático [Pieper, 1968]. En otras palabras, se separa “brazo” de “muñeca”. El método utilizado es una adaptación ya que este método es válido para robots de 6 grados de libertad. Teniendo en cuenta el desacople mencionado anteriormente, se refiere a la posición de la muñeca referida a la posición del extremo del robot, menos la distancia del eslabón L_4 con valor absoluto L_4 y orientación \vec{x}_4 . Las variables articulares se indican con q_i ($i=1,2,3$) y corresponden a los eslabones 1, 2 y 3 correspondientes.

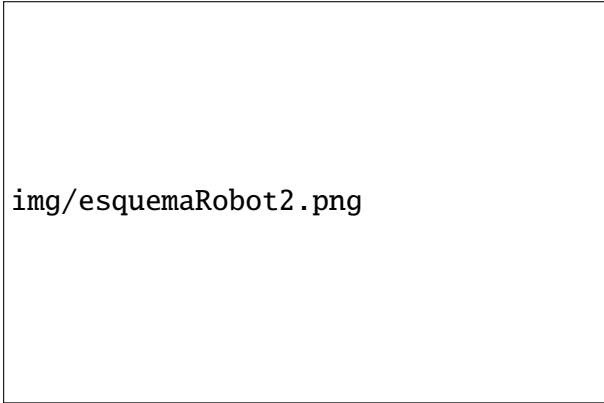
$$P_M = P_F - L_4 \cdot \vec{x}_4 \quad (5)$$

El primer ángulo se obtiene de manera directa según la proyección sobre el eje XY (véase Figura 12b):

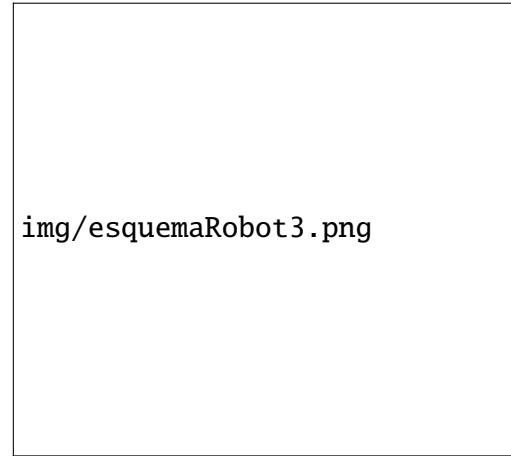
$$q_1 = \arctan\left(\frac{{}^0Y_M}{{}^0X_M}\right) \quad (6)$$

$$d = \sqrt{{}^1X_M^2 + {}^1Y_M^2} \quad (7)$$

$$B = \arctan\left(\frac{{}^0X_M}{{}^0Y_M}\right) \quad (8)$$



(a)



(b)

Figura 12: Referencias del método geométrico

271 Por el teorema del coseno, si se tienen las longitudes de los eslabones se puede determinar
 272 el ángulo entre el eslabón L_2 y el vector d que representa la distancia entre P_M y el sistema de
 273 referencia 1. Con estos valores se puede calcular el valor de la variable articular q_2 . Como se indicó
 274 previamente, se elige la disposición “codo arriba”.

$$L_3^2 = d^2 + L_2^2 - 2 \cdot d \cdot L_2 \cdot \cos(C) \quad (9)$$

$$C = \arccos\left(\frac{d^2 + L_2^2 - L_3^2}{2 \cdot d \cdot L_2}\right) \quad (10)$$

$$q_2 = B \pm C \quad (11)$$

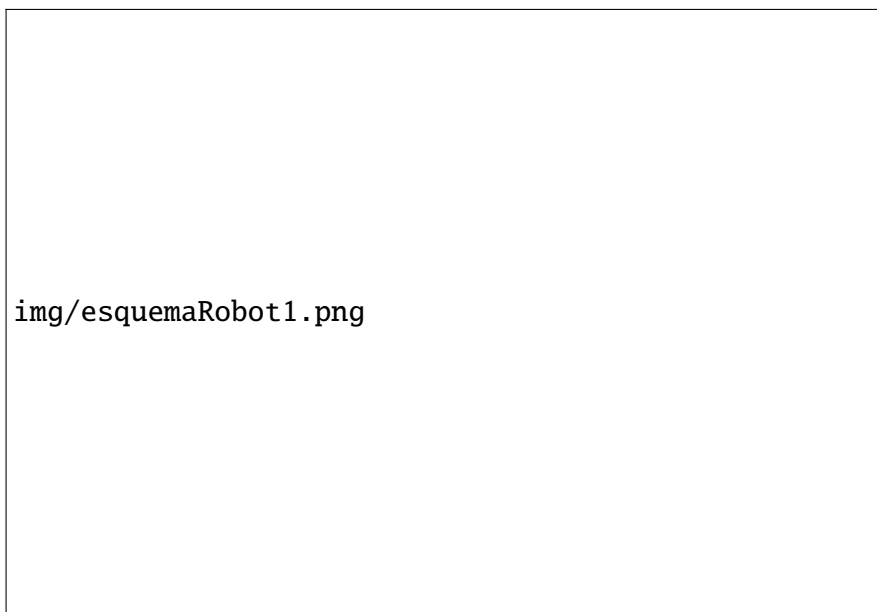


Figura 13: Esquema de eslabones simplificado

Tomando el mismo triángulo generado entre el vector distancia d , L_2 y L_3 , y haciendo uso del teorema del coseno, se puede determinar el ángulo ψ , y con este obtener q_3 .

$$d^2 = L_2^2 + L_3^2 - 2 \cdot L_2 \cdot L_3 \cos(\psi) \quad (12)$$

$$\psi = \arccos\left(\frac{L_2^2 + L_3^2 - d^2}{2 \cdot L_2 \cdot L_3}\right) \quad (13)$$

$$q_3 = \pi - \psi \quad (14)$$

Para obtener la posición del extremo final, se usa el eslabón L_4 y se considera un grado de libertad ficticio, q_4 , el cual se obtiene de manera similar a las variables articulares anteriores (ver Figura 14). Para encontrar el vector r que representa la distancia entre el extremo del eslabón 2 y P_F , se resuelve algebraicamente:

$${}^0T_1 \cdot {}^1T_2 \cdot {}^2T_3 \cdot {}^3T_4 = {}^0T_4 \Rightarrow {}^2P_F = ({}^0T_1 \cdot {}^1T_2)^{-1} \cdot {}^0T_4 \quad (15)$$

$$r^2 = L_3^2 + L_4^2 - 2 \cdot L_3 \cdot L_4 \cdot \cos(\gamma) \quad (16)$$

$$\gamma = \arccos\left(\frac{L_3^2 + L_4^2 - r^2}{2 \cdot L_3 \cdot L_4}\right) \quad (17)$$

$$q_4 = \pi - \gamma \quad (18)$$

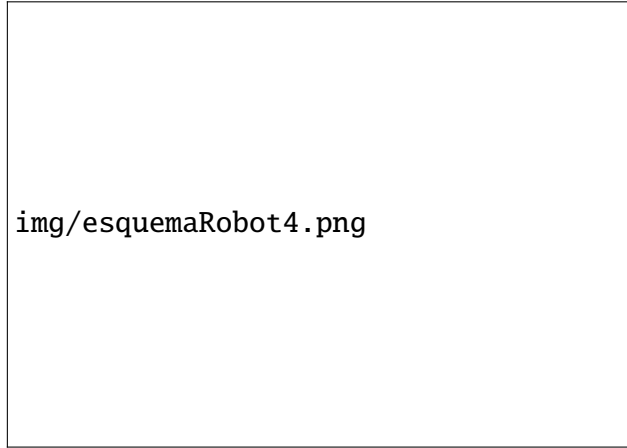


Figura 14: Dimensiones de interés para cálculo de q_4

Cabe destacar que las variables articulares representan de manera teórica los ángulos sucesivos que deben tener los eslabones entre si. Dadas las características constructivas del robot, los ángulos de giro de cada motor se calculan de la siguiente manera:

$$q_{1m} = q_1 \quad (19)$$

$$q_{2m} = \frac{\pi}{2} - q_2 \quad (20)$$

$$q_{3m} = -(q_2 + q_3) \quad (21)$$

10. Mejoras mecánicas

Durante la elaboración del proyecto, se fueron realizando distintas pruebas de manera iterativa, y se llegó a la conclusión de que para que el robot pueda cumplir con las consignas de manera correcta, deberían hacerse adaptaciones.

10.1. Descripción general

Se reemplazaron los servomotores MG996R que poseía originalmente por motores Paso a Paso en las articulaciones 2 y 3 del Robot. La articulación 1 (base) ya contaba con un motor Paso a Paso previamente instalado pero era accionada mediante una correa tipo GT2. En la nueva versión, se diseñaron pares de engranajes de transmisión para cada articulación. Para la base se empleó un módulo de transmisión $m = 1$. Para las articulaciones 2 y 3 se seleccionó $m = 1,5$ con el fin de aumentar el tamaño de los dientes, mejorando así el contacto entre engranajes. Esto permite una transmisión de carga más robusta y reduce la probabilidad de deslizamientos o pérdida de pasos bajo esfuerzo. Estas mejoras se pueden ver en las Figuras 15 y 16.

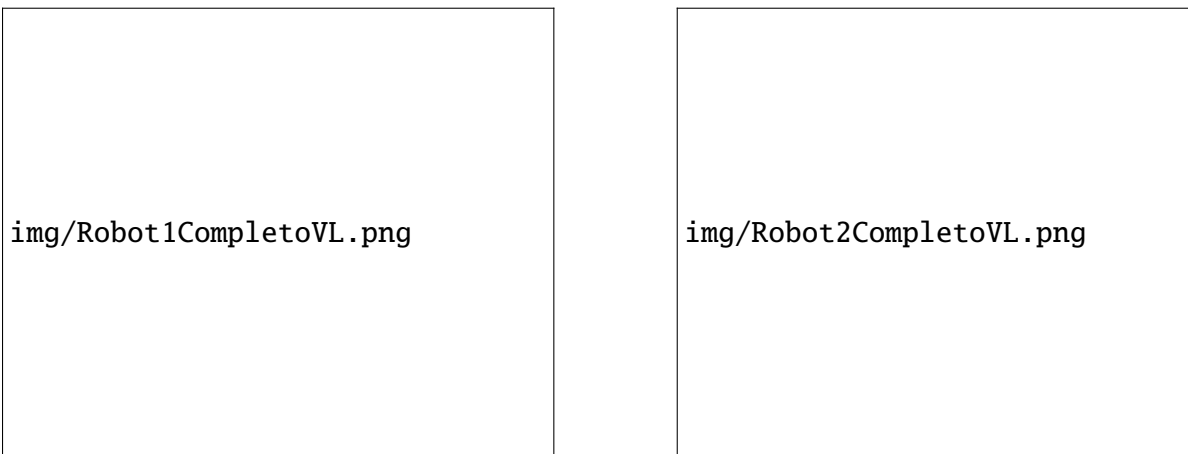


Figura 15: Comparación vista lateral de versión con servo vs con motores Paso a Paso



Figura 16: Comparación vista en perspectiva de versión con servo vs con motores Paso a Paso

10.2. Datos de diseño y conteo de dientes

Los pares de ruedas dentadas definidos son:

- **Base (articulación 1):** engranaje de la articulación $z_{a1} = 108$ dientes; engranaje del motor $z_{m1} = 19$ dientes; módulo $m_1 = 1$.
- **Articulaciones 2 y 3:** engranaje de la articulación $z_{a2} = z_{a3} = 32$ dientes; engranaje del motor $z_{m2} = z_{m3} = 12$ dientes; módulo $m_2 = m_3 = 1,5$.

Para referencia, el diámetro primitivo de un piñón/rueda se calcula como:

$$D = m \cdot z \quad (22)$$

donde D es el diámetro primitivo (en mm), m el módulo (mm) y z el número de dientes.

Aplicando la relación anterior se obtienen los diámetros primitivos:

$$\begin{cases} D_{m1} &= m_1 \cdot z_{m1} = 1 \times 19 = 19 \text{ mm} \\ D_{a1} &= m_1 \cdot z_{a1} = 1 \times 108 = 108 \text{ mm} \end{cases} \quad (23)$$

$$\begin{cases} D_{m2} &= m_2 \cdot z_{m2} = 1,5 \times 12 = 18 \text{ mm} \\ D_{a2} &= m_2 \cdot z_{a2} = 1,5 \times 32 = 48 \text{ mm} \end{cases} \quad (24)$$

$$\begin{cases} D_{m3} &= m_3 \cdot z_{m3} = 1,5 \times 12 = 18 \text{ mm} \\ D_{a3} &= m_3 \cdot z_{a3} = 1,5 \times 32 = 48 \text{ mm} \end{cases} \quad (25)$$

10.3. Condición de distancia entre ejes

El proceso de cálculo de engranajes y pruebas fue iterativo, pero la base del robot ya se encontraba diseñada. Para no cambiar el diseño 3D de la base, se impuso la restricción de que la distancia entre ejes para las articulaciones 2 y 3 es igual a 33 mm. La suma de los radios primitivos debe cumplir:

$$r_m + r_a = \text{distancia entre ejes} \quad (26)$$

Considerando a r_m y r_a como los radios de D_m y D_a respectivamente. Expresado en función de los diámetros, la ecuación queda:

$$D_m + D_a = 2 \times \text{distancia entre ejes} \quad (27)$$

para el caso de las articulaciones 2 y 3 se verifica:

$$D_{m2} + D_{a2} = D_{m3} + D_{a3} = 18 + 48 = 66 \text{ mm} \Rightarrow \text{distancia entre ejes} = 33 \text{ mm} \quad (28)$$

Este resultado explica por qué la combinación $m_2 = m_3 = 1,5$, $z_{m2} = z_{m3} = 12$ y $z_{a2} = z_{a3} = 32$ fue elegida: satisface exactamente la restricción geométrica impuesta por el ensamble y además da números de dientes enteros. El cálculo se realizó de manera iterativa (búsqueda de pares (z_m, z_a) para que sean números enteros compatibles con $D_m + D_a = 66 \text{ mm}$) hasta encontrar combinaciones prácticas que cumplieran con los requisitos geométricos y de relación de transmisión.

Para la base, con $m_1 = 1$ se tiene:

$$D_{m1} + D_{a1} = 19 + 108 = 127 \text{ mm} \Rightarrow \text{distancia entre ejes (base)} = 63,5 \text{ mm} \quad (29)$$

A diferencia de las articulaciones 2 y 3, en la base del robot no fue necesario realizar un proceso iterativo para ajustar la relación de engranajes con el objetivo de agrandar los dientes del mismo. Esto se debe a que la fuerza peso del robot no es transmitida a través de los dientes del engranaje de la base, sino que es soportada estructuralmente por los elementos mecánicos. En consecuencia, el dimensionamiento del tren de engranajes no estuvo condicionado por esfuerzos verticales significativos, permitiendo adoptar directamente un módulo de $m_1 = 1$, el cual satisface adecuadamente la distancia entre ejes requerida de 63,5 mm.

10.4. Relaciones de transmisión y resolución angular

La relación de transmisión (reductor) entre motor y articulación se obtiene por

$$i = \frac{z_a}{z_m} \quad (30)$$

Para los pares diseñados:

$$\begin{cases} i_1 &= \frac{108}{19} \approx 5,6842 \\ i_2 &= i_3 = \frac{32}{12} = \frac{8}{3} \approx 2,6667 \end{cases} \quad (31)$$

Si el motor Paso a Paso presenta un ángulo de avance por paso completo igual a $\theta_{\text{step}} = 1,8^\circ$ (por ejemplo, en motores de 200 pasos por revolución), entonces el incremento angular que experimenta la articulación por cada paso del motor viene dado por:

$$\theta_{out} = \frac{\theta_{step}}{i} \quad (32)$$

Reemplazando los respectivos valores:

$$\begin{cases} \theta_{out1} &= \frac{1,8^\circ}{5,6842} \approx 0,3167^\circ \text{ por paso del motor 1} \\ \theta_{out2} &= \theta_{out3} = \frac{1,8^\circ}{2,6667} \approx 0,6750^\circ \text{ por paso de motores 2 y 3} \end{cases} \quad (33)$$

Para mejorar la resolución del sistema de accionamiento, se implementó *micropaso* de 1/8 en los motores. Bajo este esquema, el ángulo por micropaso está dado por:

$$\theta_{microstep} = \frac{1,8^\circ}{8} = 0,225^\circ \quad (34)$$

Dada una relación de reducción i , el incremento angular de cada articulación por micropaso del motor es:

$$\theta_{out} = \frac{\theta_{microstep}}{i} \quad (35)$$

Por lo tanto, para las tres articulaciones del robot:

$$\begin{cases} \theta_{out1} &= \frac{0,225^\circ}{5,6842} \approx 0,0396^\circ \\ \theta_{out2} &= \theta_{out3} = \frac{0,225^\circ}{2,6667} \approx 0,0843^\circ \end{cases} \quad (36)$$

El número total de micropasos por revolución del motor pasó a ser:

$$N_{microsteps_per_rev} = 200 \times 8 = 1600 \quad (37)$$

10.5. Sensado de ángulos

Para la medición de la posición angular en las articulaciones se incorporaron sensores magnéticos AS5600, ubicados enfrentados a los engranajes correspondientes. Cada sensor funciona en conjunto con un imán de neodimio que se encuentra acoplado mecánicamente al eje del propio engranaje, de modo que la rotación del conjunto se traduzca directamente en la variación del campo magnético percibido por el sensor. El ángulo medido es relativo, por lo tanto requiere de una meticulosa calibración que permita tener una referencia consistente respecto a los ejes del robot y garantizar mediciones precisas.

El principio de funcionamiento del AS5600 se basa en la detección de la orientación del campo magnético generado por el imán. Para garantizar una lectura estable, lineal y precisa, el *datasheet* especifica que la separación axial entre el sensor y el imán debe mantenerse dentro del rango recomendado de [0,5; 3] mm., además admite un desalineamiento axial de 0,25mm. Fuera de este intervalo pueden presentarse errores de medición, pérdida de resolución o incluso la imposibilidad de detectar correctamente el campo magnético.

Entonces, durante el montaje se prestó especial atención a la alineación del imán con el centro del chip y al control de la distancia sensor-imán, asegurando que ambos parámetros se mantuvieran dentro de las tolerancias indicadas por el fabricante. Como se puede observar en la Figura 17, se diseñó una pieza que fue capaz de posicionar el sensor centrado respecto al imán. Su diseño permite absorber pequeños desplazamientos generados por las tolerancias que manejan las piezas (impresas en 3D) y los movimientos relativos al conjunto imán-sensor. Esto permitió obtener una señal angular confiable, que es una condición fundamental para el control preciso de las articulaciones.



Figura 17: Alineamiento axial entre sensor AS5600 e imán montado sobre eje de articulación

10.6. Cambios en el modelado cinemático

Dado que se conservaron las longitudes de los eslabones principales, la matriz de parámetros Denavit–Hartenberg del robot no sufrió grandes modificaciones salvo por el primer parámetro de la primera articulación (desplazamiento a lo largo del eje z). En términos prácticos:

- Los parámetros $\{a_i, \alpha_i, d_i, \theta_i\}$ permanecieron invariantes para los eslabones superiores salvo por d_1 (altura de la articulación 1 a lo largo del eje z) que se ajusta para reflejar la nueva geometría del alojamiento y la interfase entre la base y el primer engranaje.
- Esto implica que las transformaciones homogéneas y la cinemática directa/inversa se conservaron esencialmente, siendo necesario únicamente actualizar el parámetro afectado y volver a validar la cinemática numérica.

11. Circuito electrónico del electroimán

Se evaluaron dos opciones de conmutación: transistor NPN y MOSFET de canal N. Teniendo la resistencia de la bobina medida ($123,5\Omega$) y sabiendo que se la va a alimentar con 12V, por Ley de Ohm la corriente de trabajo es $\frac{12V}{123,5\Omega} \approx 97mA$, la cual es reducida para los dos componentes propuestos, por lo que ambas soluciones son realizables. Sin embargo, por criterios de eficiencia, escalabilidad y fiabilidad, se optó por utilizar MOSFET de canal N.

Como se tuvo a disposición un MOSFET no-logic-level, es decir, con un voltaje Gate-Source $V_{GS} > 5V$ y el nivel lógico de salida del STM32 es de 3,3V, fue necesario utilizar un transistor como level-shifter/pull-down. De esta manera, se llevó la Gate del MOSFET a 12 V para garantizar conducción plena ($V_{GS} \approx 12V$). Esta solución invirtió la lógica de la señal (el electroimán se enciende con una salida lógica en *LOW*), pero evitó la limitación de 3,3V del STM32 sin necesidad

de drivers dedicados. El circuito actúa como una etapa de conmutación de potencia controlada por el microcontrolador: cuando la salida digital del STM32 conmuta, el transistor NPN modula el voltaje en la compuerta del MOSFET, saturándolo o bloqueándolo según el nivel lógico. El transistor se satura con una salida lógica en *LOW* y se bloquea con una salida lógica en *HIGH*. De este modo, el MOSFET conmuta la alimentación del electroimán, permitiendo su activación o desactivación desde la señal de control.

Se desarrolló una placa electrónica para controlar la alimentación del electroimán, acorde al diagrama esquemático de la Figura 18a. Se agregó un diodo flyback en paralelo al mismo y en polarización inversa para proteger los componentes electrónicos de los picos de voltaje generados por el colapso del campo magnético cuando se desenergiza. El diodo ofrece un camino de baja impedancia para la corriente inducida, disipándola de manera segura y previniendo daños a otros componentes sensibles como el MOSFET. Puede observarse en la Figura 18b la placa electrónica en 3D.

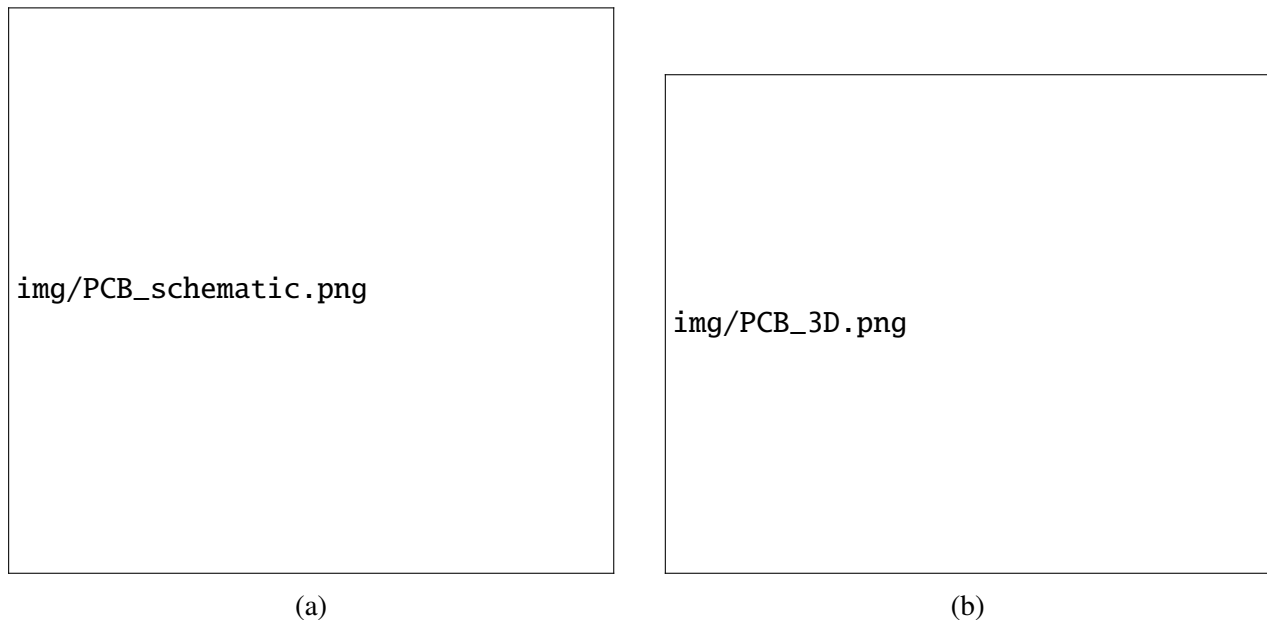


Figura 18: Esquemático y placa electrónica de control del solenoide

12. Visión artificial

Para la detección automática de objetos se empleó un modelo basado en **YOLOv11n** (You Only Look Once, versión 11 nano), desarrollado por Ultralytics. Este modelo se caracteriza por su equilibrio entre precisión y velocidad, siendo ampliamente utilizado en tareas de visión artificial en tiempo real. YOLOv11 realiza la detección de objetos mediante una red neuronal convolucional que procesa una imagen completa en una sola pasada, dividiéndola en celdas que predicen simultáneamente las coordenadas de los objetos y sus respectivas probabilidades de pertenencia a una clase.

12.1. Dataset y entrenamiento

Se generó un dataset propio con las siguientes categorías:

- Llave fija.
- Llave tubo.
- Mecha.
- Tornillo.
- Tuerca.
- Resorte.
- Rodamiento.
- Punta de destornillador.

La elección de estas piezas se fundamenta en que son piezas metálicas, de un tamaño y peso adecuado que es soportado por el electroimán. Además, otro aspecto importante es que son piezas simétricas, considerando un eje longitudinal y uno transversal. Esto último, aporta al hecho de que al reconocer el objeto dentro de la imagen capturada de una vista superior, el centro de la *bounding-box* de los objetos coincidirá con mucha precisión con el centro de gravedad del objeto. De esta manera, asumiendo esta simplificación, se facilita encontrar el punto por el que debe sujetar el electroimán a las piezas en cuestión. Se puede observar en la Figura 19 algunas imágenes que forman parte del dataset.

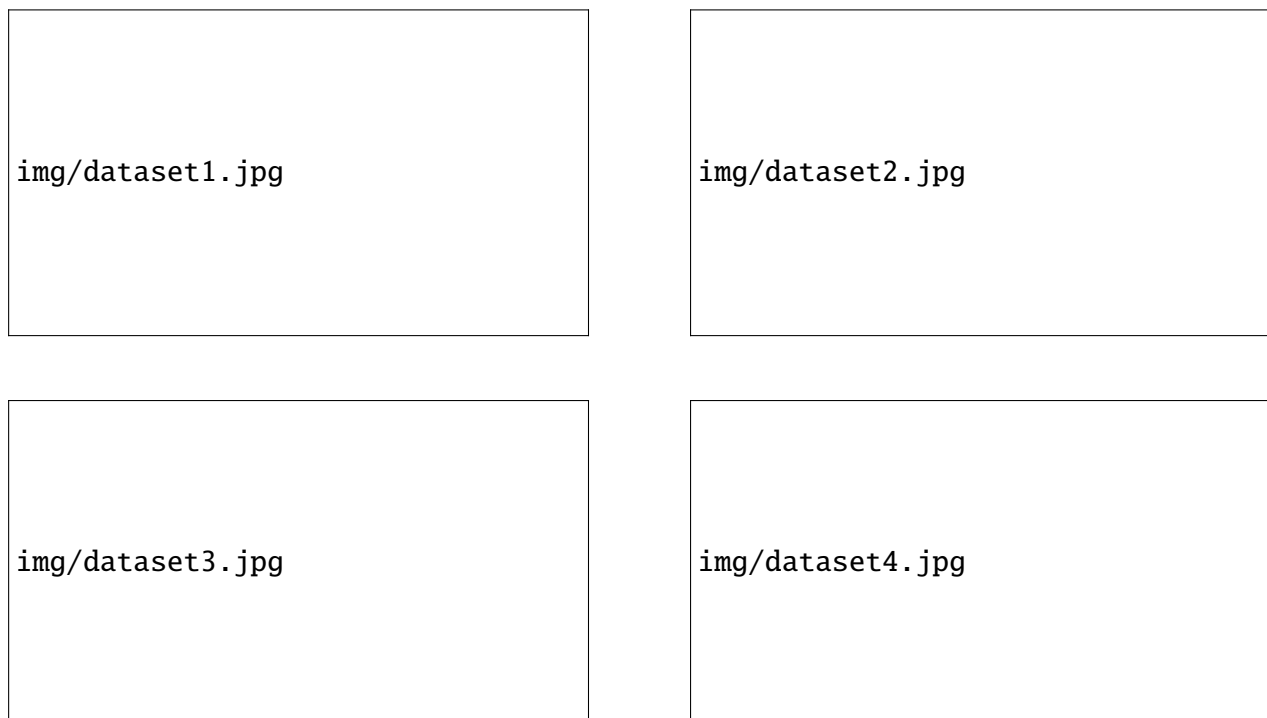


Figura 19: Imágenes del dataset

12.2. Preparación del dataset

- **Captura de imágenes:** el conjunto de datos (dataset) fue elaborado con imágenes capturadas de las herramientas a detectar en distintas condiciones de iluminación, distancia y orientación, de modo que el modelo pueda generalizar adecuadamente ante variaciones del entorno.
- **Etiquetado:** cada imagen fue anotada manualmente usando una herramienta llamada Label-Studio, el cual es un software Open-Source de etiquetado de datos. Una vez terminado se exportó empleando el formato YOLO, el cual asocia a cada objeto detectado una etiqueta que incluye su clase y las coordenadas normalizadas del cuadro delimitador (*bounding-box*).
- **Entrenamiento y validación:** Los archivos de anotación se organizaron junto a las imágenes dentro de una estructura de directorios compatible con YOLOv11, y se definió un archivo `data.yaml` donde se especifican las rutas a las carpetas de entrenamiento y validación, así como los nombres de las clases. Posteriormente, el conjunto de datos total se dividió en un 90 % para entrenamiento y un 10 % para validación, asegurando una distribución balanceada de las clases.

12.3. Entrenamiento en Google Colab

El proceso de entrenamiento se realizó en la plataforma Google Colab, aprovechando la disponibilidad de unidades GPU para acelerar el cálculo. En primer lugar, se instaló el paquete oficial de Ultralytics mediante el comando:

```
!pip install ultralytics
```

Luego, se ejecutó el entrenamiento con el siguiente comando base:

```
!yolo detect train data=/content/data.yaml model=yolov11n.pt epochs=60
imgsz=640
```

En este comando se especifica:

- **data:** ruta al archivo de configuración del conjunto de datos
- **model:** arquitectura base seleccionada. En este caso, YOLOv11n, que es una versión liviana adecuada para pruebas iniciales
- **epochs:** número de iteraciones completas sobre el conjunto de entrenamiento
- **imgsz:** tamaño al que se redimensionan las imágenes de entrada (640 píxeles)

Durante el entrenamiento, YOLOv11 optimiza los pesos de la red minimizando una función de pérdida compuesta, que considera la precisión de las *bounding-box*, la clasificación correcta de las clases y la confianza de detección. La fase de validación se ejecuta automáticamente al finalizar cada *epoch*, utilizando el 10 % del conjunto de datos reservado específicamente para este propósito. Esta separación garantiza que la evaluación del desempeño del modelo se realice sobre muestras que no fueron empleadas durante el entrenamiento, evitando así una estimación artificialmente optimista de su capacidad predictiva. De lo contrario, el modelo podría incurrir en *overfitting*, fenómeno en el cual aprende patrones excesivamente específicos del conjunto de entrenamiento —incluyendo ruido o particularidades irrelevantes— y, como consecuencia, presenta un deterioro significativo en su desempeño cuando se enfrenta a nuevas imágenes. La validación con datos no vistos permite monitorear este comportamiento y constituye una herramienta fundamental para asegurar la correcta generalización del modelo.

12.4. Parámetros de entrenamiento: *epochs* e *imgsz*

Durante el proceso de entrenamiento del modelo YOLOv11, dos parámetros influyen significativamente en el rendimiento final: el número de *epochs* y el tamaño de imagen (*imgsz*).

El parámetro *epochs* define la cantidad de veces que el modelo recorre completamente el conjunto de datos durante el entrenamiento. Un número mayor de *epochs* permite que la red neuronal ajuste mejor sus pesos y mejore su capacidad de generalización; sin embargo, un valor excesivo puede conducir al *overfitting*.

Por otro lado, el parámetro *imgsz* especifica la resolución a la cual se redimensionan las imágenes antes de ingresar a la red. Un tamaño mayor proporciona más detalle y mejora la precisión de detección, aunque también incrementa el tiempo de entrenamiento y el uso de memoria. En cambio, un tamaño menor reduce los requisitos computacionales y acelera el proceso, pero puede disminuir la exactitud en la identificación de objetos pequeños o con bordes finos.

En este proyecto se seleccionaron los valores de *epochs* = 60 y *imgsz* = 640, donde se puede afirmar que se pudo optimizar la precisión del modelo sin comprometer significativamente el tiempo de entrenamiento ni la velocidad de inferencia.

En la Figura 20 se presenta un ejemplo de las predicciones obtenidas tras el entrenamiento del modelo. Cada elemento identificado en la escena se encuentra delimitado mediante una *bounding-box*, acompañada por su correspondiente probabilidad de clasificación -expresada como un valor entre 0 y 1- que refleja el nivel de certeza del modelo respecto de dicha detección.



Figura 20: Predicción de modelo YOLOv11

Se observa que todos los objetos presentes en la imagen fueron reconocidos correctamente, alcanzando valores de confianza elevados en todas las categorías, con un mínimo de 0,80. Este comportamiento evidencia una adecuada capacidad del modelo para discriminar entre clases visualmente similares, incluso bajo condiciones de iluminación natural y fondos no totalmente uniformes.

De esta manera, resulta destacable que el modelo logra identificar objetos que se encuentran parcialmente solapados. A pesar de esta superposición, las *bounding-boxes* se ajustan correctamente a los contornos de cada herramienta, lo que indica que la red es capaz de extraer características

robustas y mantener un desempeño consistente ante escenas con oclusiones leves. Este tipo de comportamiento es especialmente relevante para aplicaciones prácticas, donde las piezas rara vez se encuentran totalmente aisladas o perfectamente separadas dentro del campo visual.

12.5. Evaluación del modelo

Al concluir el entrenamiento, se generan métricas de desempeño como:

- **Precision (P)**: proporción de verdaderos positivos respecto a todas las detecciones positivas.
- **Recall (R)**: proporción de verdaderos positivos detectados respecto al total de objetos reales.
- **mAP-0.5** (*mean Average Precision*): promedio de precisión considerando un umbral de intersección sobre unión de 0.5, utilizado como indicador global del rendimiento del modelo.

A partir de estas métricas, se selecciona la mejor versión del modelo (archivo `best.pt`), la cual presenta el equilibrio óptimo entre precisión y generalización. Este modelo puede posteriormente exportarse a formatos optimizados como `.tflite` o `.onnx`, según el entorno de ejecución deseado (por ejemplo, para una Raspberry Pi). En la Figura 21 se observa el resultado de inferencia obtenido con el modelo entrenado.

img/yolo_results.png

Figura 21: Curvas de resultado de inferencia de YOLOv11n

12.6. Inferencia y detección de objetos

Una vez finalizado el entrenamiento, el modelo es capaz de realizar inferencia sobre nuevas imágenes o secuencias de video. El proceso de detección consiste en los siguientes pasos:

1. La imagen de entrada se redimensiona al tamaño requerido (640×640 píxeles) y se normaliza.
2. El modelo predice, para cada celda de la imagen, un conjunto de posibles cuadros delimitadores con sus respectivas probabilidades de clase.

3. Se aplica un filtrado de confianza y una técnica denominada *Non-Maximum Suppression* (NMS) para eliminar detecciones redundantes.

4. Finalmente, se generan los cuadros delimitadores (*bounding-boxes*) y etiquetas de clase correspondientes a los objetos detectados.

El resultado es una salida visual o estructurada en la que cada objeto reconocido aparece identificado por su categoría y su nivel de confianza, representado como un porcentaje sobre la imagen original. El modelo resultante logró una detección precisa y estable, manteniendo un equilibrio adecuado entre velocidad de procesamiento e identificación confiable de objetos.

12.7. Cálculo de posición del objeto detectado

Una vez verificada la detección correcta del objeto en la imagen, junto con la delimitación precisa de su *bounding-box*, y considerando la simplificación previamente establecida de que el centro de la *bounding-box* coincide con un alto grado de precisión con el centro de gravedad de la pieza, el siguiente paso consiste en traducir esta información visual en datos útiles para el robot. En particular, el robot requiere como entrada la posición espacial del objeto detectado, expresada en coordenadas compatibles con su plano de trabajo.

Para posibilitar esta conversión, resulta indispensable ubicar la cámara en una posición fija y conocida respecto de la base del robot, manteniendo constante la altura durante todo el proceso de operación. Esta condición permite asegurar que la relación geométrica entre el sistema de visión y el espacio de trabajo permanezca invariable, habilitando así una transformación coherente entre coordenadas de imagen y coordenadas físicas.

En la Figura 22 se puede apreciar una representación de la vista superior del sistema robótico, donde el círculo representa el robot, que su base es el origen de referencia. El cuadrado es la cámara y el punto es el centro de gravedad del objeto detectado. De esta manera, la cámara existirá en la posición (X_{cam}, Y_{cam}) respecto del origen del robot. Mientras que el objeto detectado, se encuentra en una posición relativa respecto a la cámara (X'_{obj}, Y'_{obj}) . Manteniendo fija la posición de la cámara, podremos encontrar con precisión la posición del objeto detectado respecto el origen del robot. El hecho de que el eje vertical sea X (positivo hacia arriba) y el horizontal sea Y (positivo hacia la izquierda), no es arbitrario sino que coincide con la dirección de los ejes de la base del robot.

La localización del objeto en la imagen se obtiene a partir del centro de la *bounding-box*, sin embargo, dicha medida está expresada en píxeles. Por lo tanto, es necesario aplicar un factor de conversión entre píxeles y milímetros (mm/px), determinado por la calibración previa del sistema. Una vez conocido este factor, es posible calcular la posición del objeto en el plano $X-Y$ mediante la siguiente relación:

$$\mathbf{pos}_{obj_mm} = \mathbf{pos}_{cam} + \mathbf{pos}_{obj_px} \times f \quad (38)$$

donde f representa el factor de conversión mm/px obtenido experimentalmente, \mathbf{pos}_{cam} corresponde a la posición fija de la cámara en el plano, y \mathbf{pos}_{obj_px} es la posición del centro del objeto detectado en coordenadas de imagen (píxeles).

Finalmente, para completar la terna de coordenadas del objeto, el valor de Z se determina a partir de una base de datos que almacena la altura asociada a cada categoría de pieza reconocida. De esta manera, el sistema es capaz de reconstruir la posición tridimensional del objeto. Se puede entender la posición según la ecuación:

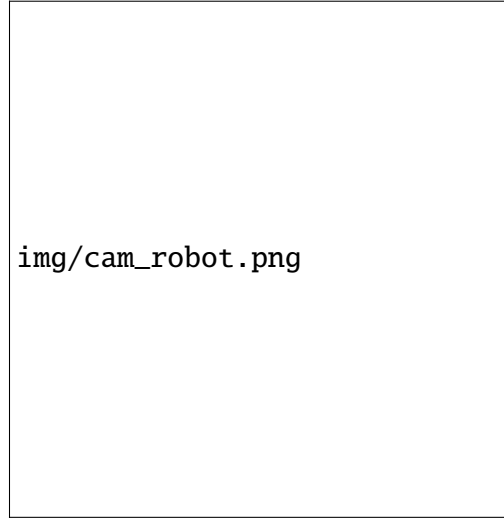


Figura 22: Esquema de posición Robot-Cámara-Objeto

$$pos_{obj_mm} = \begin{bmatrix} x_{obj} \\ y_{obj} \end{bmatrix} \quad (39)$$

549 donde se puede observar que el vector posición pos_{obj_mm} se desglosa en sus componentes x_{obj}
 550 e y_{obj} . Así, el vector de posición del objeto queda:

$$\begin{bmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \end{bmatrix} = \begin{bmatrix} pos_{obj_mm} \\ z_{database} \end{bmatrix} \quad (40)$$

551 13. Desarrollo del Firmware

552 En esta sección se describe el funcionamiento de los elementos que componen el firmware, así
 553 como el diseño de la arquitectura definida para poder comunicar al robot con el orquestador. Se optó
 554 por usar microROS ya que permite darle modularidad al sistema, permitiendo que sea escalable y en
 555 caso de requerir más funcionalidades, se puede crear un tópico y añadir un *callback* que responda a
 556 este mismo.

557 13.1. Arquitectura general

558 El sistema consta de un orquestador, que es el encargado de enviar consignas de alto nivel al robot.
 559 En el proyecto se ejecuta sobre una computadora personal, pero podría ser un microprocesador, como
 560 por ejemplo alguno integrado en las placas Raspberry[®]. Desde el orquestador se envían comandos
 561 de posición al robot, tanto articular como en el espacio cartesiano; además se envían los comandos
 562 de Homing y de parada de emergencia. El envío de consignas de velocidad y aceleración se deja
 563 como trabajo futuro.

564 Es una variante de ROS para microcontroladores. Diseñado por *eProxima*, presenta una evolución
 565 a la comunicación entre microcontrolador y otro microprocesador o computadora respecto a *ROS1*.
 566 La elección de este enfoque se debe a que se puede adaptar rápidamente al ecosistema ROS, y
 567 reemplaza al ROS-bridge *rosserial*, ya que ROS1 ya llegó a su *EOL* (no tiene más soporte oficial).

Para que la comunicación entre el nodo ROS del orquestador y el robot se pueda dar, se debe tener un agente que se encarga de realizar una “traducción” de DDS a micro-XRCE DDS.

13.2. Módulos básicos

13.2.1. Programación motores

Se programaron los motores Paso a Paso de forma que la generación de pulso en el pin STEP de cada driver DRV8825 sea controlada por PWM. Un gran beneficio de esto es que la generación de pulso por PWM no consume recursos de software del STM32 ya que de esta tarea se encarga el hardware del mismo. De esta manera, se dedicó a cada motor un timer correspondiente.

La programación por PWM requiere calcular el valor requerido en los registros ARR (Auto Reload Register) y PSC (Prescaler). Al cambiar la velocidad requerida por cada motor en trayectorias complejas, es necesario recalculer estos valores constantemente para generar la frecuencia correspondiente por PWM. El período se calcula como:

$$Periodo = \frac{(ARR + 1) \times (PSC + 1)}{Freq_Clock} \quad (41)$$

Despejando de la ecuación, el cálculo de ARR para una frecuencia determinada de *Periodo* es:

$$ARR = \frac{Periodo \times Freq_Clock}{PSC + 1} - 1 \quad (42)$$

Considerando que en la interrupción de PWM los motores realizan un micropaso a la vez, se puede entender a la frecuencia del PWM como *steps/s*. En la Tabla 2 se detallan los valores de frecuencia por PWM con la velocidad de los motores y en los eslabones (carga de cada motor) en múltiples unidades.

	Frecuencia PWM [Hz]						
	50	100	200	300	400	500	1000
Motores 1,2,3 [rev/s]	0.03125	0.0625	0.125	0.1875	0.25	0.3125	0.625
Motores 1,2,3 [rpm]	1.875	3.75	7.5	11.25	15	18.75	37.5
Motores 1,2,3 [°/s]	11.25	22.5	45	67.5	90	112.5	225
Carga 1 [rev/s]	0.0055	0.0110	0.0220	0.0330	0.0440	0.0550	0.1100
Carga 1 [rpm]	0.3299	0.6597	1.3194	1.9792	2.6389	3.2986	6.5972
Carga 1 [°/s]	1.9792	3.9583	7.9167	11.8750	15.8333	19.7917	39.5833
Carga 2 y 3 [rev/s]	0.0117	0.0234	0.0469	0.0703	0.0938	0.1172	0.2344
Carga 2 y 3 [rpm]	0.7031	1.4063	2.8125	4.2188	5.6250	7.0313	14.0625
Carga 2 y 3 [°/s]	4.2188	8.4375	16.8750	25.3125	33.7500	42.1875	84.3750

Tabla 2: Frecuencias de PWM y velocidades en motores/eslabones

13.2.2. Programación sensores de ángulo

Para la adquisición de los ángulos articulares se utiliza un multiplexor I²C TCA9548A, cuya función es permitir la lectura independiente de varios sensores AS5600 que comparten la misma dirección I²C. El microcontrolador habilita uno de los ocho canales del multiplexor escribiendo un

byte en su registro de control, donde cada bit activa un canal distinto. De esta manera, sólo el sensor asociado al canal seleccionado queda accesible en el bus I²C para la siguiente lectura.

Una vez elegido el canal correspondiente, se accede a los registros del AS5600 para obtener los dos bytes que contienen la medida angular en formato *raw*. Dichos bytes se combinan y se extraen los 12 bits significativos del valor, que representan el ángulo absoluto del imán en un rango de 0 a 4095. Dependiendo de la disposición mecánica del sensor respecto del eje, puede ser necesario invertir el sentido de algunas lecturas para mantener un sistema de referencia coherente entre todas las articulaciones.

El valor *raw* obtenido se convierte a grados mediante la relación:

$$\theta = \frac{\text{raw}}{4096} \cdot 360^\circ \quad (43)$$

Posteriormente, el ángulo se reexpresa en el rango $[-180, 180)$, lo cual simplifica los cálculos de control y evita discontinuidades cuando el valor se hace negativo.

Dado que en algunas ocasiones pueden producirse lecturas erróneas por ruidos en la comunicación I²C o por pequeñas perturbaciones durante la adquisición, se incorpora un mecanismo de rechazo de picos. Este consiste en comparar la lectura actual con la lectura previa: si la diferencia entre ambas supera un umbral preestablecido, se considera que la medición no es válida y se mantiene el último valor fiable. Si la variación se encuentra dentro del límite permitido, la lectura se acepta como correcta. Este criterio resulta suficiente para los sensores AS5600, ya que el ángulo real normalmente varía de manera continua y suave entre muestras, por lo que los saltos abruptos suelen indicar claramente una lectura incorrecta.

13.2.3. Programación cinemática

Inicialmente, la cinemática del manipulador fue desarrollada y verificada en *MATLAB*, lo que permitió asegurar la coherencia entre el modelo teórico y el comportamiento físico del robot. Una vez validado el modelo, se realizaron pruebas donde se implementó el cálculo de la cinemática inversa directamente sobre el microcontrolador *STM32F446RE*, aprovechando la *Floating Point Unit (FPU)* integrada, para operar con aritmética en coma flotante simple y alcanzar desempeño en tiempo real. Sin embargo, la resolución final de la cinemática se trasladó a Python, ya que forma parte de una función que es llamada por el orquestador del sistema, responsable también del cálculo de la trayectoria. Esta decisión permitió simplificar los procesos de cálculo y validación dentro del espacio de trabajo, además de contribuir a minimizar el consumo de memoria en el microcontrolador. Asimismo, el tiempo de cálculo de la trayectoria no supera los *35ms*, por lo que resulta lo suficientemente bajo como para justificar su ejecución en Python. De este modo, la generación de consignas articulares y la planificación del movimiento quedan integradas en un mismo entorno de procesamiento.

13.2.4. Generación de Trayectorias sincronizadas

La generación de trayectorias sincronizadas para el robot se divide en dos partes: la definición de la forma geométrica en el espacio cartesiano y la asignación de una ley de movimiento temporal que respete las limitaciones dinámicas de los motores Paso a Paso. Esta separación permite desacoplar el problema geométrico del dinámico, simplificando el diseño y análisis de la trayectoria.

La primer parte consiste en establecer un conjunto de puntos de control (*waypoints*) en el espacio cartesiano, que terminan definiendo la ruta que debe seguir el efector final. En este nivel, solo importa la geometría: el punto inicial *A*, el punto final *B* y los puntos de paso obligatorios que garantizan,

por ejemplo evitar obstáculos. A partir de los puntos definidos, se genera una curva paramétrica $p(s)$ suave en el espacio cartesiano interpolada utilizando B-Splines cúbicas:

$$p(s) = [x(s), y(s), z(s)], \quad s \in [0, 1] \quad (44)$$

Donde el parámetro s es geométrico y no representa tiempo físico. El uso de B-Splines cúbicas garantiza continuidad de segundo orden (C^2), es decir: $p(s)$, $\dot{p}(s)$ y $\ddot{p}(s)$ son continuas, lo que significa que no habrá saltos infinitos en velocidad o aceleración. Una propiedad relevante de las B-Splines es que permite generar curvas que pasen específicamente por los puntos asignados (se les asigna gran peso) o simplemente aproximarse para asegurar suavidad en la curvatura (se les asigna poco peso), para más información consultar [BSp, 2001]. Para este proyecto se asignó:

- Gran peso en los extremos, para garantizar pasar exactamente por las posiciones inicial y final
- Poco peso en los puntos intermedios, para evitar cambios bruscos en posición o velocidad

Para cada punto $p(s)$ se resuelve la cinemática inversa, obteniendo la trayectoria articular:

$$q(s) = [q_1(s), q_2(s), q_3(s)] \quad (45)$$

La evolución del parámetro s en el tiempo queda definida por un polinomio de grado 5. A continuación, se puede observar el polinomio y sus dos siguientes derivadas respecto al tiempo.

$$s(t) = a_5 \tau^5 + a_4 \tau^4 + a_3 \tau^3 + a_2 \tau^2 + a_1 \tau + a_0, \quad \tau = \frac{t}{T} \quad (46)$$

$$\dot{s}(t) = \frac{1}{T} \left(5a_5 \tau^4 + 4a_4 \tau^3 + 3a_3 \tau^2 + 2a_2 \tau + a_1 \right) \quad (47)$$

$$\ddot{s}(t) = \frac{1}{T^2} \left(20a_5 \tau^3 + 12a_4 \tau^2 + 6a_3 \tau + 2a_2 \right) \quad (48)$$

Siendo T el tiempo total de la trayectoria. Para encontrar los valores de los coeficientes es necesario establecer 6 condiciones de contorno:

$$\begin{cases} s(0) = 0, & s(T) = 1 \\ \dot{s}(0) = 0, & \dot{s}(T) = 0 \\ \ddot{s}(0) = 0, & \ddot{s}(T) = 0 \end{cases}$$

Al aplicar estas condiciones y resolviendo el sistema de ecuaciones, los coeficientes resultantes son: $a_0 = a_1 = a_2 = 0$ $a_3 = 10$ $a_4 = -15$ $a_5 = 6$. De esta manera, el polinomio $s(t)$ con sus respectivas derivadas queda de la siguiente manera:

$$s(t) = 6\tau^5 - 15\tau^4 + 10\tau^3 \quad (49)$$

$$\dot{s}(t) = \frac{1}{T} \left(30\tau^4 - 60\tau^3 + 30\tau^2 \right) \quad (50)$$

$$\ddot{s}(t) = \frac{1}{T^2} \left(120\tau^3 - 180\tau^2 + 60\tau \right) \quad (51)$$

Se puede observar la representación gráfica de estas ecuaciones en la Figura 23. El procedimiento mencionado desacopla la forma de la trayectoria de su evolución temporal, permitiendo posteriormente aplicar técnicas de *time-scaling*, que consiste en escalar la trayectoria paramétrica a un tiempo T para satisfacer los límites de velocidad y aceleración en la articulación más exigida:

$$|\dot{q}| \leq \dot{q}_{max}, \quad |\ddot{q}| \leq \ddot{q}_{max} \quad (52)$$

Luego se relacionan las derivadas temporales con el parámetro geométrico:

$$\dot{q}(t) = \frac{dq}{ds} \dot{s}(t) \quad (53)$$

$$\ddot{q}(t) = \frac{d^2q}{ds^2} \dot{s}^2(t) + \frac{dq}{ds} \ddot{s}(t) \quad (54)$$

Combinando la geometría $q(s)$ con la ley temporal $s(t)$ se obtienen: $q(t)$, $\dot{q}(t)$ y $\ddot{q}(t)$, que constituyen las trayectorias articulares físicamente realizables por los motores (ver Figura 24). La representación de la trayectoria interpolada en el espacio de trabajo con puntos de control se puede ver en la Figura 25.

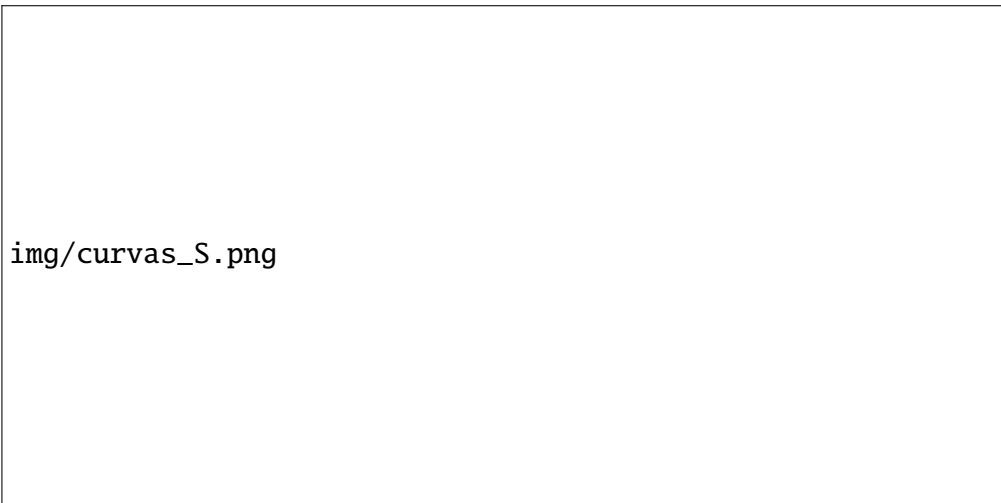


Figura 23: Posiciones, velocidades y aceleraciones del parámetro s

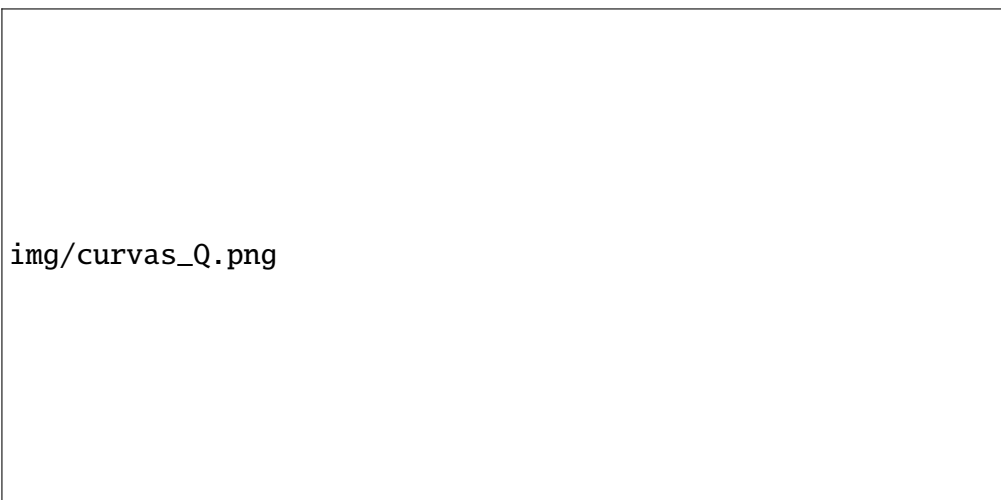


Figura 24: Posiciones, velocidades y aceleraciones articulares

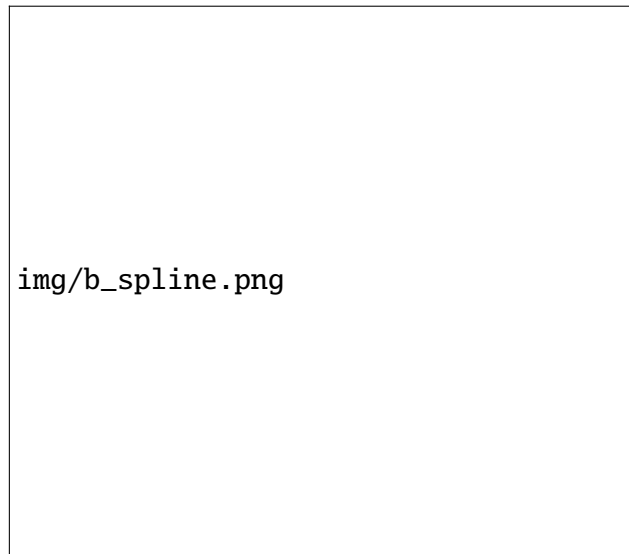


Figura 25: Trayectoria en el espacio de trabajo interpolada con B-Spline

14. Integración de ROS, microROS y FreeRTOS en Sistemas Embebidos

El objetivo de esta sección es describir de manera general la arquitectura de integración entre el nodo orquestador y el microcontrolador, enfatizando los roles de cada componente y su relación en el sistema robótico.

El control de alto nivel (control supervisor) se hace en la computadora, y este nodo envía las consignas de posición, homing, parada de emergencia. El sistema de mensajería de ROS permite disminuir el tiempo de desarrollo del envío de información entre los sistemas a través de modos que se comunican con un patrón *publisher/subscriber* anónimo. El microcontrolador ejecuta microROS, una variante de ROS que se ejecuta dentro de una *Task* de FreeRTOS. La misma está dedicada a la creación de *executors*, que manejan la comunicación entre los nodos y las publicaciones/suscripciones.

14.1. ROS

ROS2 provee distintos mecanismos de comunicación entre módulos, que son objetos llamados nodos. Los mecanismos de comunicación son interfaces, quienes permiten el intercambio de información entre nodos de manera estructurada. Las principales son mensajes, servicios y acciones, cada una diseñada para diferentes patrones de comunicación dentro del sistema.

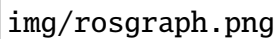
Los mensajes se transmiten a través de tópicos, siguiendo el patrón *publisher/subscriber*, como se mencionó anteriormente. Cualquier nodo que publique un mensaje en un tópico determinado puede ser recibido por todos los nodos suscritos a él. Este mecanismo es ideal para flujos de datos continuos o asincrónicos, como sensores o comandos periódicos. En este caso se envían las consignas de posición mediante tópicos con un tiempo periódico fijo.

Los servicios implementan un modelo *request/response*, permitiendo que un nodo solicite una operación a otro y reciba una respuesta inmediata. Son adecuados para interacciones puntuales que requieren confirmación o retorno de datos.

Las acciones están diseñadas para tareas que requieren más tiempo en completarse. Una acción está compuesta por dos servicios (uno para enviar el objetivo *-goal-* y otro para obtener el resultado

final) y un flujo de mensajes publicados en un t pico para proporcionar retroalimentaci n durante la ejecuci n. Otra diferencia entre acciones y servicios es que las primeras pueden ser canceladas.

En la Figura 26 se pueden ver los nodos correspondientes al orquestador ROS (llamado “minimal_publisher”) y los nodos correspondientes al m dulo de c lculo de trayectoria y al de la interfaz de usuario. El sentido de las flechas es desde el publicador hacia el suscriptor.



img/rosgraph.png

Figura 26: Gr fico de nodos ROS

La comunicaci n entre nodos viene dada por un protocolo basado en UDP, y se debe respetar la red en la que se comunican los nodos. Por ejemplo, se puede tener grupos de nodos con un prop sito en la misma red que otro grupo. Para esto se usa el **ROS_DOMAIN**, que en este caso es 0, lo que quiere decir que el rango de conexi n aceptado para las conexiones es entre los puertos 7400 y 7651 (m s detalles en 18.5).

Se eligieron tipos de mensajes *core* de ROS para ser enviados. Estos son del tipo **Point** y **Bool**. Tambi n se pueden enviar mensajes de tipo String y hacer una “traducci n” de una trama personalizada. En este caso se opt  por usar unos tipos de mensajes nativos, pero tambi n se pueden crear mensajes personalizados (formato .msg). A continuaci n se muestra un mensaje de tipo nativo. Con el siguiente comando se puede ver el tipo de mensaje, en este caso de tipo Point.

```
ros2 interface show geometry_msgs/msg/Point
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Estructura de mensaje Point

Se pueden crear mensajes personalizados, con el formato `<tipo de dato> <nombre_variable>` y ubicando el archivo en la carpeta `msg` en el proyecto. Por ejemplo, para el envío de comandos para variables articulares, se usó esta configuración:

```
float64[3] q      # Angulos de las articulaciones (posicion)
float64[3] qd     # Velocidades angulares
int32 t_total    # Tiempo total
int32 traj_state  # Estado de la trayectoria
```

Estructura de mensaje Point

Se podría, por ejemplo, agregar un header con el *timestamp* del mensaje, y algún flag. Los mensajes pueden ser compuestos por otros mensajes, por ejemplo:

```
ros2 interface show std_msgs/msg/Header

# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
# Two-integer timestamp that is expressed as seconds and nanoseconds.

builtin_interfaces/Time stamp
  int32 sec
  uint32 nanosec

# Transform frame with which this data is associated.
string frame_id
```

Ejemplo de estructura compuesta

Para extender ROS a microcontroladores, se utiliza *microROS* (ver 14.2). La arquitectura típica sigue el siguiente esquema:

1. **ROS2 en un sistema host:** nodos más complejos, planificación y coordinación.
2. **microROS en un microcontrolador:** tareas de interpretación de consignas y adquisición de sensores.
3. **FreeRTOS:** como RTOS subyacente.
4. **microROS Agent:** como puente entre DDS y DDS-XRCE.

Esta estructura permite que un microcontrolador se comporte como un nodo ROS nativo desde la perspectiva del resto del sistema. En la Figura 27 se puede apreciar un detalle.

Como mención, se puede agregar que las mediciones de las variables articulares pueden ser guardadas en una cola RTOS y ser enviadas a un publisher mediante un tópico.

Otro dato no menos relevante es que se pueden elegir diversas estrategias para enviar y recibir consignas, mensajes de ACK/NAK, mensajes de debug o valores de sensores. En el caso de estudio actual se optó por distintos *topics*, por una cuestión de facilidad de desarrollo y prototipado. Otra opción disponible era hacer uso de *actions*. Por ejemplo, se podría enviar la consigna de posición deseada (goal), obtener el resultado y estar suscrito a un tópico de feedback que tendría las posiciones actuales de los motores.

Cabe destacar que ROS tiene una estructura interna que se debe seguir, porque en ella se indican los nodos, a que clases corresponden, las dependencias y demás. Además de esto cuenta con herramientas que ya facilitan este tipo de configuraciones.

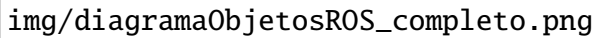
img/diagramaObjetosROS_completo.png

Figura 27: Mapa conceptual de las conexiones entre el orquestador y el esclavo

14.2. microROS

El stack microROS es una versión reducida de ROS2 diseñada específicamente para microcontroladores con recursos limitados de RAM, CPU y energía. Se desarrolla como proyecto *open source* desde 2018. Hace uso del protocolo DDS-XRCE, una variante ligera del DDS utilizada por ROS2 que permite la serialización y transporte eficiente de mensajes. El objetivo de este software stack es que el código de ROS2 sea fácilmente portable a microcontroladores [Belsare et al., 2023]. El stack microROS utiliza la arquitectura en capas de ROS y reutiliza la mayor cantidad de paquetes de ROS2.

Entre las principales características, se puede mencionar que, además de contar con APIs similares a ROS2 (optimizadas para baja memoria), se cuenta con soporte para distintos RTOS y para diferentes canales de comunicación. En la capa de Middleware, usa una implementación del standard DDS-XRCE de eProsima, Micro XRCE-DDS (DDS For Extremely Resource Constrained Environments). Para poder comunicar el microcontrolador con el orquestador, se necesita un agente que hace de puente entre DDS y DDS-XRCE.

La configuración de los nodos y del *executor* se hacen en una tarea de FreeRTOS. Las suscripciones y publicaciones a los distintos tópicos (considerando tipos de datos), los timers (si los hay) y el enlace con los callbacks asociados a cada tópico) son manejadas por el *executor*.

El flujo se puede resumir de la siguiente manera: se inicia el transporte (UART en este caso) y el middleware, luego se especifican los publishers/subscribers o timers si los hubiera, se define el *executor* asociado y se mantiene sincronizado con el agente. Se considera la liberación de recursos en caso de que se finalice la comunicación.

14.3. FreeRTOS

FreeRTOS es un sistema operativo de tiempo real ampliamente utilizado en sistemas embebidos y está diseñado para microcontroladores. Provee un modelo simple de multitarea con prioridades explícitas, colas, semáforos y temporizadores. En este proyecto se ha optado por usar FreeRTOS. Cabe destacar al haber una tarea dedicada a microROS no se bloquean tareas críticas dentro del sistema.

El nodo microROS necesita al menos 10 kB de memoria en el stack para poder ejecutarse, y la comunicación serial se hace mediante UART con DMA. Se tienen distintos tipos de transports, se elige DMA porque no usa tiempo de procesamiento en la CPU, lo cual evita sobrecargas. Es

importante que el canal Rx esté configurado como circular ². Otro detalle a mencionar es que para la programación en el microcontrolador se usó la API de CMSIS que viene integrada en los microcontroladores ST. Esta misma sirve como capa de abstracción porque se pueden usar sus funciones y las mismas pueden servir para otro RTOS subyacente, como por ejemplo Zephyr.

FreeRTOS administra los recursos temporales y de concurrencia, mientras que microROS gestiona la comunicación a nivel de middleware.

Entre las funcionalidades principales de FreeRTOS se encuentran:

- **Tasks:** hilos ligeros con prioridad configurada. Como se ve en la gráfica presentada anteriormente, hay tareas para el control de los motores, para el cálculo de la cinemática y para el modo homing.
- **Queues:** se usan para que haya comunicación segura entre tareas. Se pueden hacer colas y arrays de colas. Este último caso sirve para `mid_JointQueue`, así se separa una cola por motor.
- **Timers:** ejecución periódica controlada por el RTOS.
- **APIs seguras para ISR:** integración con interrupciones externas. Se pueden configurar en el caso de parada de emergencia.
- **Gestión de memoria configurable:** importante dado que microROS favorece esquemas estáticos. Por cuestiones de funcionamiento, la tarea asignada a microROS debería tener al menos 10 kB.

La selección adecuada de prioridades entre tareas de control y tareas asociadas a microROS es crítica para cumplir *deadlines*.

14.4. Integración: ROS2 – microROS – FreeRTOS

El ecosistema general puede describirse mediante tres niveles:

1. **Nivel de aplicación (ROS2):** nodos de planificación, control de alto nivel e interfaces de usuario.
2. **Nivel de middleware:** ROS2 utiliza DDS completo; microROS emplea DDS-XRCE con un agente como intermediario.
3. **Nivel embebido (microcontrolador):** FreeRTOS ejecuta las tareas del sistema y el *rlc_executor* procesa los callbacks de microROS.

Teniendo en cuenta estos puntos, se presenta un gráfico (Figura 28) desde el punto de vista de ROS, con la información de los nodos correspondientes al orquestador y al agente del microcontrolador.

Este enfoque desacopla el control de alto nivel del control en tiempo real, permitiendo un diseño robusto y escalable. Entrando más en detalle, se pueden diferenciar las distintas partes del sistema. En la computadora, el sistema está compuesto por 4 nodos principales:

² Así el DMA escribe de manera continua los datos entrantes en un búfer sin requerir reinicio manual ni intervención constante de la CPU. Una vez que se reciben todos los bytes de datos, se reinicia el contador y el DMA continúa recibiendo datos



img/rosgraph_micro.png

Figura 28: Diagrama de conexiones entre nodos

- **robot_ui:** Interfaz de usuario que permite enviar comandos al robot, carga de trayectorias CSV y comunicación con servidores externos.
- **trajectory_planner:** Nodo encargado de la planificación de trayectorias. Recibe posiciones cartesianas y genera trayectorias articulares interpoladas, publicando mensajes del tipo extra_interfaces/Trama (posiciones q , velocidades \dot{q} , tiempo y estado de trayectoria).
- **publisher:** es el encargado de comunicar la interfaz de usuario con el microcontrolador, así como con el planificador de trayectoria.
- **microROS Agent:** actúa como puente de comunicación (*bridge*), traduciendo los mensajes DDS de ROS2 al protocolo XRCE-DDS utilizado por el microcontrolador a través de la interfaz UART.

En la Tabla 3 se muestra una breve descripción de los tópicos utilizados, el tipo de mensaje utilizado, el sentido de la comunicación y un detalle funcional.

El firmware del STM32 implementa un sistema multitarea basado en FreeRTOS cuyas tareas principales son:

- **StartDefaultTask:** Tarea de inicialización que configura el transporte UART, crea el nodo microros_pubsub_rclc y ejecuta el bucle del executor para despachar los callbacks de las subscripciones.
- **StartMotorControlTask:** Tarea consumidora responsable de ejecutar los movimientos de cada motor. Inicializa los métodos relacionados a la inicialización de las consignas PWM.

Tópico	Tipo de mensaje	Dirección	Descripción
/microROS/homing	std_msgs/Bool	ROS \Rightarrow STM32	Comando de homing
/microROS/cmd	extra_interfaces/Trama	ROS \Rightarrow STM32	Trayectoria articular
/microROS/inverse	geometry_msgs/Point	ROS \Rightarrow STM32	Posición cartesiana (cinemática inversa)
/microROS/string_publisher	std_msgs/String	STM32 \Rightarrow ROS	Realimentación de posición

Tabla 3: Descripción de tópicos más relevantes

- **Callbacks de subscripción:** Incluyen `cmd_callback` (trayectorias), `homing_callback` (referenciado) e `inverse_kinematics_callback` (coordenadas cartesianas). El microcontrolador es capaz de realizar los cálculos de cinemática directa e inversa internamente.

El flujo de datos sigue una secuencia definida para garantizar la integridad de los comandos:

1. **Fase de Recepción:** Al recibir un mensaje, el executor invoca el `cmd_callback`, que convierte el mensaje a una estructura interna `JOINT_POS_t` y la deposita en la cola `mid_JointQueue` mediante `osMessageQueuePut()`.
2. **Sincronización:** La cola funciona como un mailbox (tamaño unitario); si llega un comando nuevo, el anterior es sobrescrito para evitar datos obsoletos.
3. **Fase de Ejecución:** la tarea lee la cola, e inicia el movimiento mediante Timers de hardware en modo PWM. La rutina de interrupción `HAL_TIM_PeriodElapsedCallback()` actualiza la posición en tiempo real.
4. **Fase de Retroalimentación:** la posición se guarda en `mid_FeedbackQueue` al completar el movimiento. Un temporizador de software (500 ms) consulta esta cola y publica el estado en el tópico `/microROS/string_publisher` utilizando `rcl_publish()`.

14.5. Flujo de información en trayectorias

El flujo de información más completo es el de generación de trayectorias de múltiples objetivos. Para realizar una determinada trayectoria, se detallan a continuación los pasos que se aplican desde la publicación de los comandos de posición hasta la ejecución en el microcontrolador:

1. **robot_ui:** el nodo de interfaz gráfica recibe la ruta con el archivo que contiene los objetos solicitados, simulando así un determinado pedido. Publica en el tópico `/robot_ui/load_objects`, donde es consumido por el nodo `publisher`.
2. **publisher:** este nodo se encarga de la comunicación con el servidor, donde recibe la posición `xyz` del objetivo por `socket`, la decodifica y publica en el tópico `/ROS/trajectory_planning`, donde es consumido por el nodo `trajectory_planner`.

3. **trajectory_planner**: este nodo se encarga de la generación de las trayectorias considerando el interpolador B-Spline y la ley de tiempo de polinomio de grado 5 (Ecuación 46). La trayectoria sincronizada es determinada por los puntos de inicio, fin y posibles puntos intermedios. Se la discretiza con un paso de tiempo fijo, que para este proyecto se estableció en $\Delta t = 45ms$, el cual es un valor de compromiso entre rendimiento y movimientos suaves en el robot. A continuación, se publican las posiciones objetivo en el tópic `/planner/path` y son consumidas por el publisher, quien las envía al microcontrolador.
4. **startMotorControlTask**: el callback de microROS subscripto a `/microROS/cmd` se encarga de realizar una adaptación del mensaje y colocarlo en cola (como se menciona en la sección 14.4). Cada punto de la trayectoria se categoriza en uno de los siguientes estados: inicio, durante, finalización. El estado es guardado en la variable `traj_state` de la estructura *Trama* y su valor se traducirá en el control de trayectoria mediante **trajectoryControl**, inicio y parada de los timers PWM.
5. **trajectoryControl**: es la función encargada del control de trayectoria. Toma como parámetros la posición $q(i)$, la velocidad $q_d(i)$ y qué motor se desea controlar. Realiza un control de posición corrigiendo la velocidad con un control proporcional. La variación de velocidad se traduce de $^\circ/s$ a $steps/s$ con la correspondiente relación de transmisión por motor y los micropasos por revolución. Cuando se desea detener los timers, se realiza una compensación de posición hacia la posición objetivo hasta situarse dentro de un error de tolerancia de $0,2^\circ$.³

El flujo mencionado anteriormente se puede ver en la Figura 29.



img/flujoDatoConsigna.png

Figura 29: Flujo de datos en trayectorias

³Ver función de control en 18.6

15. Ensayos realizados

15.1. Ensayo de carga electroimán

Con el fin de determinar las capacidades reales de sujeción del efector final, se realizaron diversos ensayos experimentales sobre el electroimán instalado en el robot. Estos ensayos permitieron caracterizar el comportamiento del sistema ante variaciones en la masa, la geometría y la distancia de las piezas a manipular.

En primer lugar, se evaluó la carga máxima que el electroimán puede sostener de manera confiable. Se comprobó que el límite práctico se alcanza cuando se sujetan piezas de 135 g. La sujeción fue realizada respecto del centro de gravedad de las piezas, ya que más allá de este punto, el peso realiza momento que aumenta la carga efectiva, lo que reduce la capacidad de sujeción y puede comprometer la estabilidad durante el traslado. Por esta razón, se considera que la manipulación debe realizarse preferentemente tomando la pieza lo más cerca posible de su centro de gravedad y se debe tener en cuenta este valor de peso máximo por pieza.

Otro aspecto analizado fue la máxima distancia admisible entre la superficie del electroimán y la pieza para que la sujeción sea efectiva. Los ensayos realizados permitieron establecer que, para el electroimán utilizado, la distancia límite es de aproximadamente 4 mm. Esto implica que, siempre que el robot se posicione por debajo de ese valor, la atracción magnética será suficiente para asegurar la captura del objeto. Este valor deja un buen margen sobre el cual puede posicionarse el electroimán al intentar sujetar piezas.

Por último, se verificó la presencia de magnetismo remanente una vez que el electroimán es desenergizado. Este efecto provoca que objetos de muy baja masa (menores a 20 g) permanezcan adheridos al electroimán aún sin corriente. Si bien este fenómeno no afecta significativamente la manipulación de piezas más pesadas, es relevante tenerlo en cuenta en aplicaciones donde la liberación precisa del objeto sea un requisito fundamental. Finalmente se descubrió que agregando un fino aislante a la base del electroimán, los objetos de baja masa caen por su propio peso.

15.2. Ensayo de calibración de sensores de ángulo

La calibración de los sensores magnéticos AS5600 se llevó a cabo mediante un procedimiento experimental destinado a obtener una referencia angular absoluta confiable. Para esto, se usó un sensor inercial MPU6050, el cual se colocó directamente sobre los eslabones accionados por los motores correspondientes, de modo que proporcionara una medida del ángulo real respecto al eje X del robot.

El proceso consistió en posicionar cada eslabón en una orientación conocida y fácilmente identificable: 0° para el eslabón 1, 90° para el eslabón 2 y 0° para el eslabón 3. Una vez fijada cada posición y registrado el ángulo mediante el MPU6050, se procedió a leer el valor entregado por el sensor AS5600. Debido a que la medición del AS5600 depende directamente de la orientación del imán de neodimio solidario al eje, fue necesario realizar ajustes iterativos en la posición del imán hasta que la lectura reportada coincidiera con el ángulo de referencia establecido por el MPU6050.

Este método de calibración fue imprescindible porque el AS5600 no mide ángulos absolutos en un sistema de coordenadas físico predefinido, sino la rotación relativa del campo magnético generado por el imán. De esta manera, la alineación precisa entre el campo magnético y el sistema de coordenadas del robot fue esencial para garantizar coherencia entre el ángulo medido y la orientación real del eslabón.

15.3. Ensayos cinemáticos

En la etapa inicial del proyecto, el cálculo de la cinemática inversa del robot fue desarrollado y evaluado en *MATLAB*. Una vez verificada la validez del modelo, el algoritmo fue trasladado al *STM32* haciendo uso de su *FPU*. Aun así, las consignas de movimiento para cada articulación se realizan en *Python* con el objetivo de simplificar la verificación dentro del área de trabajo.

Las pruebas realizadas confirmaron que el efector final se desplaza con precisión hacia los puntos definidos por la cinemática inversa, demostrando la correcta implementación del algoritmo y su adecuada integración con el sistema de control del robot.

Mediante ensayos experimentales se verificó que los motores Paso a Paso, operando bajo carga, pueden alcanzar una velocidad máxima de aproximadamente 75 rpm (equivalente a 450°/s). Sin embargo, con el fin de garantizar un funcionamiento seguro y permitir una observación más clara del movimiento durante las pruebas y la operación normal, se decidió limitar la velocidad máxima de trabajo a 37.5 rpm (225°/s). Este valor proporciona un margen adecuado para evitar pérdidas de pasos y asegurar desplazamientos suaves y controlados en todas las articulaciones.

15.4. Comunicación ROS2/microROS

En esta etapa se ensayó la comunicación entre el orquestador (la computadora en este caso) y el robot (el microcontrolador STM32). Como se mencionó anteriormente, se debieron configurar múltiples cosas. De manera algorítmica, se puede listar un paso a paso para poder conectar el agente microROS a la computadora:

- 1. Preparación del entorno en el host:** En la computadora que ejecutará ROS2 se utiliza Docker para desplegar el entorno del microROS Agent. Este contenedor incluye todas las dependencias necesarias (DDS, transportes, compiladores y herramientas) y evita configuraciones manuales en el sistema operativo.
- 2. Obtención de las herramientas de microROS para CubeMX:** En el entorno de desarrollo del firmware se clonan los repositorios de `micro_ros_setup` (con soluciones listas) o `micro_ros_stm32cubeux_utils` (para el caso de una placa genérica ST). Estas utilidades permiten generar automáticamente el código de integración entre el microcontrolador, FreeRTOS y el cliente microROS (`rcrc`).
- 3. Generación del proyecto STM32:** se puede usar cualquier IDE o editor, en el caso de este proyecto se usó STMCube IDE, por lo tanto se adaptó con Cube MX. Se genera el proyecto con los módulos necesarios: soporte UART, inicialización del RTOS, asignación de memoria y creación de la tarea encargada de ejecutar el cliente microROS.
- 4. Configuración del transporte microROS:** Se configura UART con DMA circular y se registran las funciones de apertura, cierre, lectura y escritura con `rmw_uos_set_custom_transport`. Esta capa adapta la comunicación ROS2 al hardware embebido.
- 5. Inicialización del cliente microROS:** En la tarea principal de FreeRTOS se configuran el asignador de memoria, el soporte `rcrc`, el nodo microROS, y los publicadores y suscriptores. Además, se crea un executor que administra la ejecución de callbacks cuando llegan mensajes desde el agente.

6. **Lanzamiento del microROS Agent en Docker:** Se descarga con Docker el entorno de compilación preconfigurado. En el host se ejecuta el contenedor Docker del microROS Agent, que actúa como puente entre ROS2 y el microcontrolador. El agente traduce mensajes DDS/RTSPS a un transporte serial o UDP, y los reenvía al cliente microROS. El agente se inicia de la siguiente manera:

```
1  ros2 run micro_ros_agent micro_ros_agent serial --dev <device> -v6
2
```

Inicialización de Agente (device: dispositivo USB por ejemplo)

7. **Conexión y operación:** Una vez que el microcontrolador inicia su tarea microROS, establece comunicación con el agente. A partir de ahora y gracias al Agente, microROS se empieza a ver como un nodo.

Un detalle que vale aclarar es que durante estas pruebas hubo errores con el manejo de colas en los callbacks que son ejecutados por, valga la redundancia, el *executor*. Para esto, se debe asegurar un correcto limpiado de los elementos que son consumidos, y se debe prestar especial atención al tipo de datos que se cargan, ya que en la creación de las mismas se define el tamaño de cada elemento. Otro detalle importante es que la interfaz serial (UART) encargada de la comunicación ROS/microROS, esté configurada con DMA y el canal Rx debe ser **Circular**.

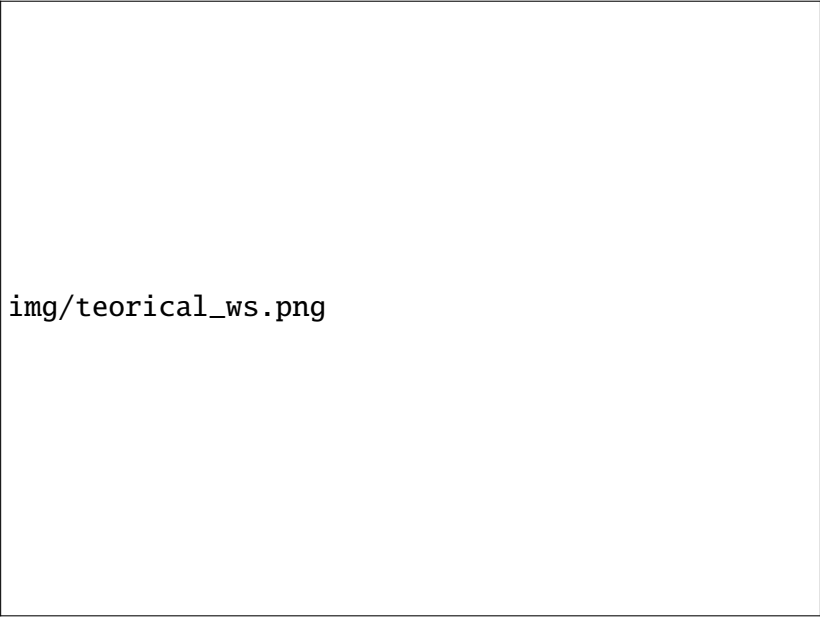
La solución que se brindó fue la siguiente: antes de insertar el dato, verificar si la cola está llena; en ese caso, eliminar el elemento más antiguo para asegurar que el valor actualizado pueda almacenarse. Luego, intentar colocar el nuevo mensaje en la cola y reportar un error si la operación no pudo completarse. De esta manera, la función garantiza que siempre exista un *setpoint* vigente sin bloquear la ejecución del sistema.

```
1 // Poner message type Point en queue
2 osStatus_t xStatus; // container para Overwrite wrapper
3 CARTESIAN_POS_t dummy;
4
5 if (osMessageQueueGetSpace(mid_PositionQueue) == 0) {
6     osMessageQueueGet(mid_PositionQueue, &dummy, NULL, 0); // eliminar
7     el mas antiguo
8 }
9 xStatus = osMessageQueuePut(mid_PositionQueue, &xSetpointToSend,
10 osPriorityNormal, 10);
11 if (xStatus != osOK) {
12     printf( "Could not send to the queue.\r\n" );
13 }
14 }
```

Fragmento de escritura en cola

15.5. Ensayo de Área de trabajo del robot


Se calculó el espacio de trabajo físico del robot en MATLAB en un plano XZ, donde solo las articulaciones q_2 y q_3 pueden moverse. Esto se realizó en un cálculo iterativo de la cinemática directa del robot, sabiendo los ángulos máximos y mínimos dados por el rango articular. la Figura 30 representa este espacio de trabajo, que si se puede observar, se lo limitó a puntos con $z \geq 0$ para tener como límite mínimo de z al plano base del robot ($z = 0$).



img/teorical_ws.png

Figura 30: Espacio de trabajo teórico

1005 Este espacio de trabajo es teórico, es decir que no representa en su totalidad el movimiento del
1006 robot, ya que el mismo tiene limitaciones físicas que impiden lograr ciertos ángulos. Se realizaron
1007 ensayos empíricos que demuestran que no todo el rango de ángulos de las articulaciones q_2 y q_3
1008 es alcanzable, sino que existen colisiones dadas por la geometría de los eslabones. Se generó una
1009 tabla (Figura 31) que combina q_2 y q_3 donde se colorean las celdas de verde si esa combinación es
1010 alcanzable dentro del espacio de trabajo, sino de color rojo.



img/ws_table_modified.png

Figura 31: Espacio de trabajo articular con restricción $z = 0$

1011 La Figura 32 presenta un zoom de la esquina superior derecha de la tabla, donde se pueden
1012 observar los pares de ángulos válidos dentro del espacio de trabajo. Con esta información, se realizó

1013 un nuevo cálculo de los puntos en el espacio teniendo en cuenta las limitaciones físicas, que se puede
1014 observar en la Figura 33.

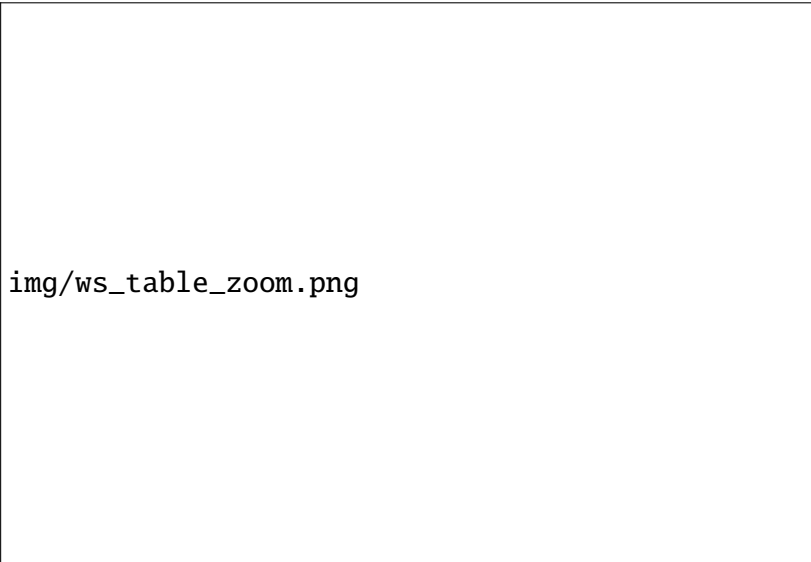


Figura 32: Espacio de trabajo articular (Zoom)

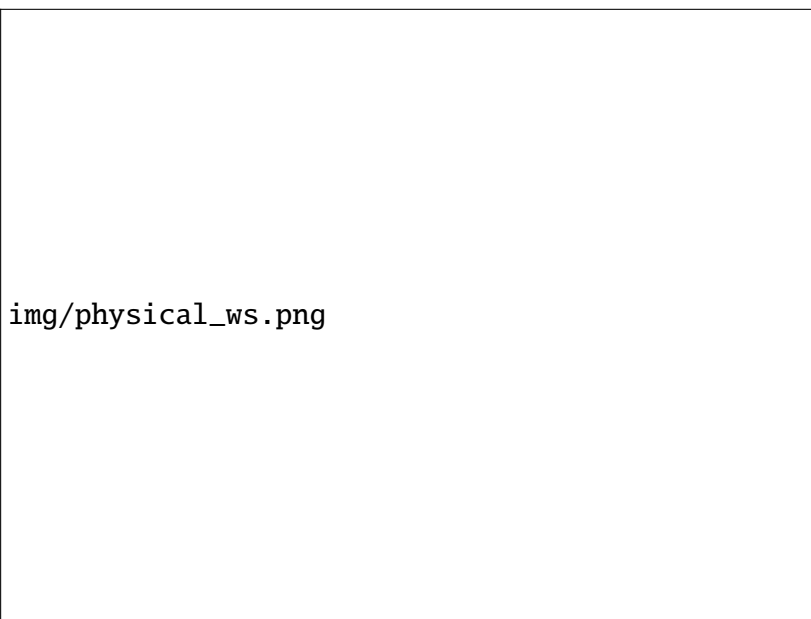


Figura 33: Espacio de trabajo real, plano XZ

1015 Se observó que existe un reducido rango de posiciones alcanzables en el plano $z = 0$, mientras
1016 que a mayor altura se iba incrementando. Se optó por “elevar” la base de posicionamiento de piezas
1017 a $40mm$ de la base del robot para tener mayor rango en este plano.

1018 Este ensayo permitió encontrar experimentalmente los puntos del espacio de trabajo tanto carte-
1019 siano como articular. Se guardó la tabla en formato .csv y durante la planificación de una trayectoria,
1020 cada punto es verificado consultando este archivo con el objetivo de comprobar que la posición so-
1021 licitada pertenece al espacio de trabajo. De esta manera, se evita intentar alcanzar posiciones no

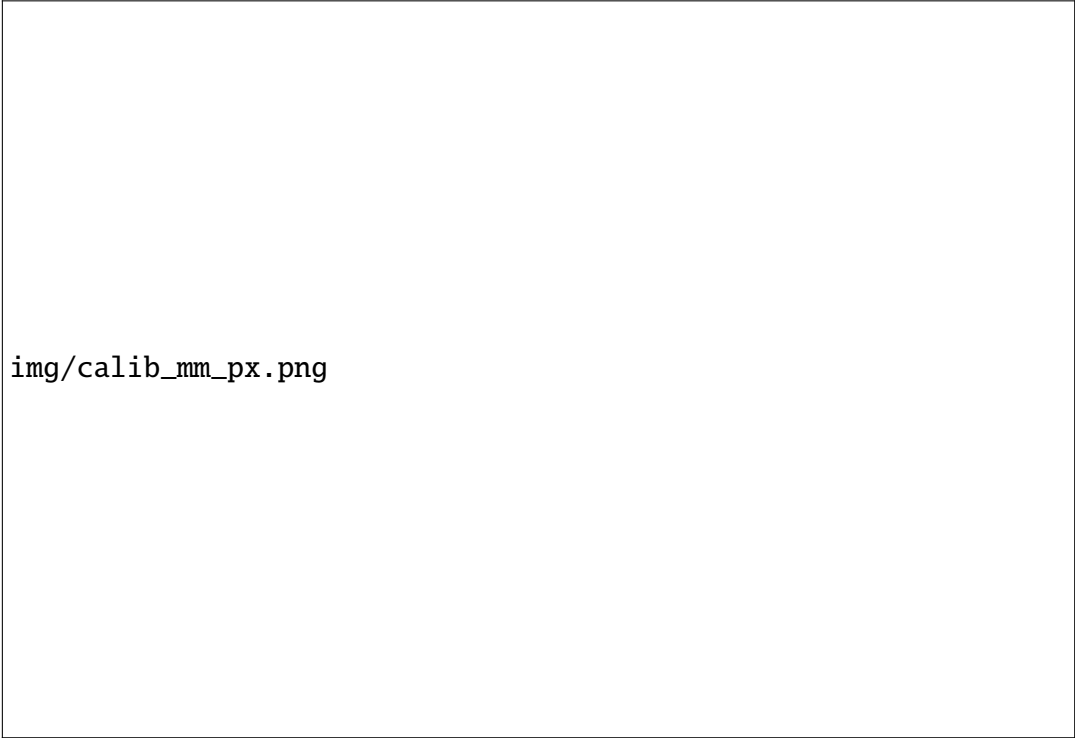
alcanzables. En caso que algún punto se encuentre fuera del mismo, la trayectoria no es generada. Como mejora futura, se propone implementar un algoritmo de planificación que re programe automáticamente los puntos intermedios que queden fuera del espacio de trabajo para ajustarlos a posiciones alcanzables; en cambio, si el punto inicial no pudiera ser alcanzado, el sistema debería notificarlo mediante un mensaje de error claro y específico.

15.6. Ensayo de visión artificial

En esta etapa se realizaron dos ensayos complementarios de visión artificial: uno basado en técnicas clásicas con *OpenCV* y otro utilizando un modelo de detección por *deep learning*. El objetivo principal fue evaluar la capacidad del sistema para identificar correctamente objetos dentro del área de trabajo y obtener su posición en coordenadas reales mediante la relación *mm/px*.

15.6.1. Detección del patrón y cálculo de la relación *mm/px*

Se desarrolló un código en Python utilizando *OpenCV* para detectar una figura rectangular o cuadrada cuyas dimensiones son conocidas. Esta figura puede ser desde una tarjeta de débito/crédito, una placa impresa en 3D o hasta un cuadrado impreso (ver Figura 34). De esta manera, se utilizó esta forma como patrón que sirvió como referencia geométrica para calcular la relación *mm/px*, necesaria para transformar coordenadas de imagen en coordenadas reales del robot.



img/calib_mm_px.png

Figura 34: Ensayo de calibración para obtener relación *mm/px*

15.6.2. Ensayo de detección mediante YOLOv11n

Además del procesamiento clásico, se evaluó el modelo entrenado con *YOLOv11n* para la detección de herramientas pertenecientes al dataset propio del proyecto. El modelo demostró capacidad

para identificar correctamente cada una de las clases consideradas, incluso ante variaciones moderadas de iluminación, con exceso de brillo, con piezas superpuestas entre sí y con piezas recortadas en los límites de la imagen.

Una vez detectado un objeto y conocido el centro de su *bounding-box*, fue posible determinar su posición real respecto del origen del robot utilizando la relación mm/px previamente calibrada. Para esto se emplea la Ecuación 38, que permite transformar coordenadas de imagen al sistema de referencia del robot.

Es importante remarcar que, para garantizar la validez de esta conversión, fue necesario fijar tanto la posición del robot como la de la cámara, evitando desplazamientos relativos que pudieran modificar la calibración. La precisión obtenida depende directamente de la estabilidad mecánica del montaje y de una correcta determinación del factor mm/px .

Como se puede apreciar en la Figura 35, se realizó la correcta detección de la herramienta y fue posible encontrar el centro de la *bounding-box*, donde se realizó la conversión a mm y se referenció este punto respecto al origen del robot.

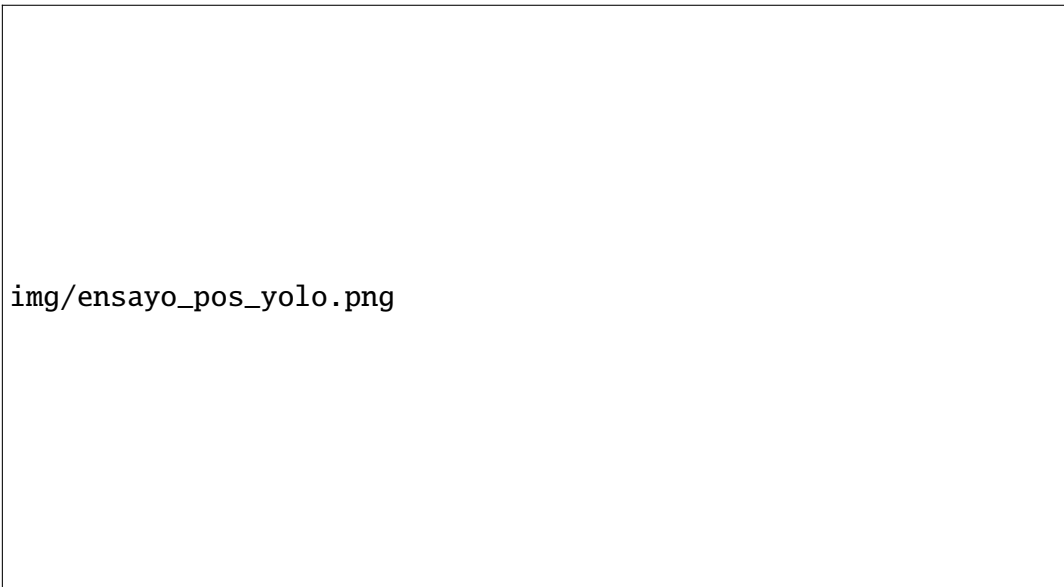


Figura 35: Ensayo de detección de herramientas y posicionamiento respecto origen del robot

Es importante realizar una aclaración respecto a la metodología empleada en este ensayo. Si bien el modelo de detección fue entrenado utilizando múltiples entidades de cada herramienta, algunas de ellas presentan variaciones de altura dentro de su misma categoría. Esta condición no resulta compatible con la suposición utilizada durante la planificación del movimiento, donde se considera una única altura representativa por clase de objeto. De esta manera, el uso de diversas entidades en el entrenamiento tuvo como objetivo principal mejorar la robustez del modelo de visión, mientras que en la práctica del proceso de manipulación, se requiere que los objetos de una misma categoría mantengan la misma altura. Si se aceptaran distintas alturas por cada objeto, se requerirían subcategorías de cada herramienta, lo cual se considera trabajo a futuro como mejora del proyecto.

15.7. Ensayo de trayectoria

Con el objetivo de evaluar el desempeño del sistema de control, se realizó un ensayo de seguimiento de trayectoria, en el cual el robot debía recorrer un perfil previamente planificado, de forma

que se registró en todo momento el error de posición de cada articulación. La trayectoria realizada es la de la Figura 24. El análisis se centró en comparar la referencia generada por el planificador con la respuesta real medida mediante los sensores angulares integrados en cada eje.

En una primera aproximación, el control se planteó como un seguimiento punto a punto de la posición y la velocidad de referencia. Sin embargo, se observó que el uso exclusivo de la velocidad calculada por la trayectoria no resultaba suficiente para garantizar un seguimiento preciso. Esto se debe a que el modelo cinemático no contempla ciertos efectos presentes en el sistema real: por un lado, los motores Paso a Paso poseen una velocidad mínima efectiva para producir movimiento, por debajo de la cual no se generan pasos; por otro, aparecen fenómenos como vibraciones, pequeñas holguras mecánicas y no idealidades en la transmisión. De esta manera, asumir que cada articulación reproducirá fielmente velocidades arbitrariamente bajas conduce a errores acumulativos durante el movimiento.

Para mitigar estos efectos, se cerró el lazo utilizando la medición de posición de cada articulación. Se implementó un control proporcional sobre la velocidad, tomando como referencia el error de posición instantáneo. El esquema adoptado se describe mediante las siguientes ecuaciones:

$$q_{error} = q_{ref} - current_angle \quad (55)$$

$$v_{new} = v_{ref} + K_p \cdot q_{error} \quad (56)$$

Siendo q_{ref} y v_{ref} la posición y velocidad de referencia en cada instante de la trayectoria. Este planteo combina un término *feedforward* de velocidad (v_{ref}), que impone la trayectoria deseada, con un término proporcional de realimentación ($K_p \cdot q_{error}$), encargado de corregir desvíos. Cuando el error de posición se reduce, el comportamiento está dominado por la velocidad planificada, mientras que ante errores significativos, el término proporcional adquiere mayor peso, incrementando o reduciendo la velocidad según el eje se encuentre atrasado o adelantado respecto de la referencia de posición. De esta forma, el sistema compensa desviaciones en tiempo real sin modificar la estructura general del plan de movimiento.

El parámetro K_p fue ajustado experimentalmente, buscando un compromiso entre rapidez de corrección y estabilidad del movimiento. Valores bajos producían un seguimiento no muy preciso, mientras que valores muy elevados generaban oscilaciones y mayor sensibilidad al ruido. Como se muestra en las Figuras 36, 37 y la Tabla 4, el valor $K_p = 5,0$ resultó ser el más adecuado para el sistema implementado, permitiendo un seguimiento cercano a la trayectoria teórica sin introducir inestabilidades apreciables. A partir de este valor, no se apreciaron aportes significativos a la reducción de error de posición. Se pudo observar que aproximadamente a mitad de la trayectoria de la articulación 1 (cuando alcanza velocidad máxima), existe un error de posición que no logró ser compensado con el aumento de K_p , por lo tanto, una posible mejora podría ser un control proporcional-derivativo.

	K_p					
	0.0	1.0	2.0	3.0	4.0	5.0
q_{error1}	19.88	9.77	8.54	7.92	7.53	7.44
q_{error2}	19.09	3.65	1.89	1.07	1.07	1.07
q_{error3}	12.05	5.92	3.76	2.76	2.15	1.75

Tabla 4: Comparación de máximos valores absolutos de errores en la trayectoria respecto a K_p



Figura 36: Comparación de trayectorias real y teórica con sus respectivas ganancias



Figura 37: Errores de posición con sus respectivas ganancias

15.8. Ensayo completo

El funcionamiento del sistema de control distribuido depende de una secuencia de inicialización para la comunicación mediante el protocolo XRCE-DDS entre el orquestador (ROS2) y el nodo embebido (microROS).

Prerrequisitos del Sistema: antes de iniciar el despliegue de los nodos, se deben verificar las siguientes condiciones operativas:

- **Servidor de Visión:** el servidor de visión artificial debe estar iniciado y en estado de escucha para el procesamiento de imágenes en tiempo real.
- **Compilación del Espacio de Trabajo:** el workspace de ROS2 debe estar correctamente construido mediante la herramienta colcon build.
- **Hardware:** el microcontrolador STM32F446RE debe estar conectado por UART/USB, con los permisos de puerto configurados (e.g., `sudo chmod 666 /dev/ttyACM0`).
- **Firmware:** el *firmware* basado en FreeRTOS y microROS debe estar cargado y en ejecución en el *target* embebido.

Para iniciar el servidor de visión se debe ejecutar el siguiente comando de ejemplo:

```
python3 yolo_with_socket.py --model best.pt --source /dev/video2 --thresh 0.75
--resolution 1280x720
```

Los argumentos usados son:

- **model:** archivo con los mejores pesos del entrenamiento
- **source:** fuente de las imágenes, en este caso la cámara. En el caso de estar en un sistema operativo Linux, para encontrar el dispositivo se puede ejecutar el comando:
v4l2-ctl --list-devices
- **thresh:** es el umbral a partir del cual detectará objetos, representado como flotante entre 0 y 1. Valores bajos pueden involucrar que el sistema detecte ruido, detectando así objetos que no formaron parte del dataset. Por defecto tiene el valor de 0,75
- **resolution:** resolución de la cámara. Se debe llegar a una solución de compromiso entre resolución y detección precisa de objetos del dataset (baja resolución implica más FPS pero puede bajar la precisión de la detección). Por defecto, para la cámara del proyecto se utiliza 1280×720
- **mm_px:** relación de mm_px calculada previamente. Se ha fijado un valor por defecto de 0,236 que corresponde a que los objetos detectados se encuentran a 40mm respecto de la base del robot, pero hay que tener en cuenta que una variación de la resolución de la cámara o distancia cámara/objetos modificará este valor

Secuencia de Ejecución de Nodos: Para que el sistema funcione correctamente, se debe seguir el orden de ejecución detallado en la Tabla 5. Es crítico que el agente y el nodo de visión estén operativos antes de lanzar el planificador.

Orden	Componente	Comando de ejecución
1	Agente microROS	ros2 run micro_ros_agent micro_ros_agent serial --dev [puerto] -v6
2	Talker	ros2 run robot_control talker
3	Trajectory Planner	ros2 run robot_control trajectory_planner
4	Interfaz de Usuario	ros2 run robot_control robot_ui

Tabla 5: Protocolo de inicio para el sistema distribuido ROS2 / microROS

Funcionalidades del Nodo de Interfaz: El nodo robot_ui (Figura 38) actúa como el punto de entrada para el operador, permitiendo la interacción con el brazo robótico mediante las siguientes funciones principales:

1. **Control Directo y Gestión de Estado (Opciones 1, 2, 4, 5 y 7):** se envían comandos directos al robot. Esto incluye rutinas de *homing*, consignas de posición directa (*Cinemática directa*) y el control del electroimán. Incluye funciones de seguridad, como la gestión de la parada de emergencia (*E-Stop*), y permite la solicitud de telemetría para visualizar la posición articular real reportada por el microcontrolador a través del tópico de retroalimentación.

2. **Cinemática Inversa (Opción 3):** permite el envío de coordenadas cartesianas (x, y, z) mediante mensajes de tipo `geometry_msgs/Point`. El sistema realiza una validación previa de los datos de entrada contra los límites físicos del espacio de trabajo antes de publicar la consigna para el cálculo de los ángulos articulares. Actualmente, esta opción es solo para validación, ya que el planificador de trayectoria genera consignas articulares directamente.
3. **Configuración de Rangos de Validación (Opción 6):** habilita la modificación de los límites locales de la interfaz para variables articulares ($^{\circ}$) y cartesianas (metros). Esta función opera exclusivamente de forma interna en el nodo de UI para filtrar comandos fuera de rango, por lo que no genera tráfico de comunicación con el resto de los nodos del sistema.
4. **Comunicación Externa (Opción 9):** implementa un cliente de *sockets* para la solicitud de pedidos a un servidor externo. Esta función requiere la existencia previa de un archivo `.csv` que contenga los objetos a solicitar.

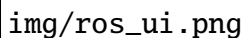


Figura 38: Interfaz de usuario

```
# Ejecucion del agente en Ubuntu 22.04
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyACM0 -b 115200
```

Ejemplo de ejecución del agente microROS

En el esquema propuesto se define el punto A como la posición en el espacio asociada a cada objeto detectado en el área de trabajo, mientras que el punto B corresponde a una ubicación fija y conocida que representa el contenedor final (caja, pallet o depósito). El procedimiento completo se estructura como un ciclo repetitivo de manipulación que parte siempre desde la posición de referencia o *homíng*, garantizando condiciones iniciales conocidas para cada operación.

El ensayo comienza con la lectura de un archivo `.csv` que contiene la lista de objetos solicitados en el pedido. Este archivo es cargado a través de la interfaz gráfica (Opción 9), lo que permite inicializar la secuencia de tareas. Una vez cargada la lista, el nodo *Talker*, que opera como cliente *socket*, establece comunicación con el servidor de visión artificial. Para cada elemento de la lista, el cliente envía una solicitud con el nombre del objeto (por ejemplo, “tuerca”).

El servidor procesa la imagen adquirida por la cámara, ejecuta el modelo de detección y responde con la posición tridimensional del objeto en un formato codificado como cadena de caracteres: $Xx.xYy.yZz.z$ donde cada valor representa las coordenadas cartesianas del punto A en el sistema de referencia del robot. En caso de que el objeto no sea detectado dentro del campo visual, el servidor devuelve $X0.0Y0.0Z0.0$, lo cual actúa como condición de error o ausencia.

El nodo cliente decodifica la cadena recibida, separando los valores de X , Y y Z , y construye así el punto objetivo A . Con esta información, el planificador de trayectorias genera la primera trayectoria en línea recta $Homing \Rightarrow A$. Una vez alcanzado este punto, se activa el electroimán para efectuar la sujeción de la pieza. Posteriormente, se calcula y ejecuta la segunda trayectoria: $A \Rightarrow B$ donde pasa por un punto intermedio y se genera una trayectoria como la de la Figura 25. Al llegar al punto B , el electroimán se desactiva para liberar el objeto en el contenedor. Finalmente se ejecuta la tercer trayectoria en línea recta: $B \Rightarrow Homing$ restableciendo la condición inicial del robot.

Este ciclo completo se repite secuencialmente para cada elemento contenido en el archivo `.csv`, hasta procesar la totalidad del pedido. De esta manera, el sistema implementa un flujo automatizado de tipo *pick-and-place*, integrando percepción, planificación de trayectoria y control del efector final dentro de una arquitectura distribuida basada en comunicación por *sockets*. En la Figura 39 se puede ver el sistema completo.

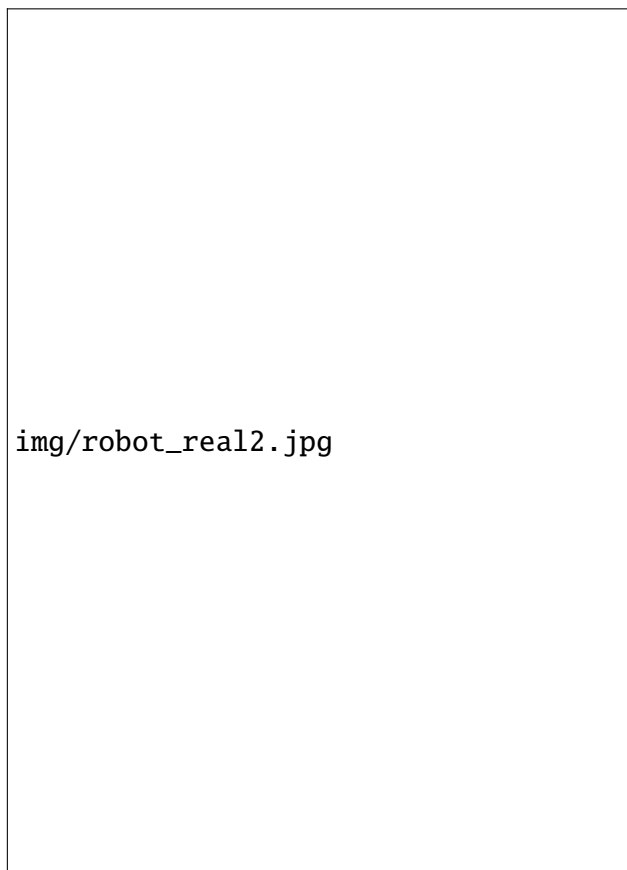


Figura 39: Foto real del robot realizando *pick-and-place*

16. Futuras mejoras

Si bien el sistema robótico cumplió con las expectativas, es necesario considerar que existen múltiples mejoras a implementar a futuro que mejoran el rendimiento, la robustez y permiten acercarse más a implementaciones de robots industriales:

1. **Servomotor en efector final:** para que el efector final pueda realizar rotaciones. Esto permitiría ampliar la versatilidad del manipulador, facilitando tareas que requieran orientación precisa de la pieza, como ensamblado o posicionamiento angular específico.
2. **Gripper en efector final:** para que tenga la posibilidad de sujetar piezas de distintos materiales y geometrías. De este modo, el sistema podría adaptarse a aplicaciones más variadas.
3. **Control PD:** para mitigar aún más los errores durante la trayectoria. La incorporación de una acción derivativa contribuiría a mejorar la respuesta dinámica del sistema, reduciendo vibraciones y tiempos de establecimiento.
4. **Sistema en Tiempo Real:** debería asegurarse determinismo a la hora de enviar las consignas desde el orquestador al robot. Esto relacionado al envío de consignas desde el orquestador al microcontrolador (requiere construir el kernel en el sistema operativo -Linux por ejemplo)
5. **Interfaz de usuario:** necesaria para generar consignas y para mover el robot en modo *Jog*. Se podría tener una interfaz más amigable para el usuario, como así automatizar la inicialización de los nodos que forman parte del sistema.
6. **Generación de trayectoria:** realizar trayectorias de manera más intuitiva y asegurar en todo momento encontrarse dentro del espacio de trabajo, en caso que se intente alcanzar un punto fuera de éste, recalcularlo. Otra opción compatible con *ROS* es usar la librería *moveIt* para enviar las consignas en alto nivel. Además, se podría implementar una trayectoria vertical a velocidad reducida al acercarse al objetivo de carga/descarga, tal como realizan los autómatas con función *pick-and-place*.
7. **Ampliar dataset:** agregando imágenes del entorno normal de operación para evitar falsa detección de objetos y agregar objetos similares a cada categoría para evitar *overfitting*. Además, que existan subcategorías por cada una de las ya existentes para ampliar la base de datos de alturas.
8. **Visión artificial:** reemplazar la cámara actual con una estereoscópica para realizar un cálculo automático de la altura del objeto y que de esta manera se calcule la relación *px/mm* de manera dinámica.
9. **Evitar colisiones:** usar la cámara ya existente para poder detectar elementos extraños y lanzar algún mensaje de error/alarma y frenar el movimiento, en el caso de que el robot esté moviéndose.
10. **Generación de algoritmos de búsqueda:** tal como A^* , para evitar colisiones de objetos en el espacio de trabajo y evitar llegar a límites articulares. Mas detalles y más soluciones pueden ser encontradas en [Lynch, 2017].
11. **Automatización de pedidos:** que exista un algoritmo de Inteligencia Artificial capaz de encontrar las mejores trayectorias en función de múltiples pedidos.

1228
1229
1230
1231

12. **Mejoras de ROS:** mejoras de bajo costo de implementación, que contribuyen a la simulación y al análisis del comportamiento del robot. Por ejemplo, el uso de *ROS bag* para hacer testing y debug, y también Gazebo para la creación de entornos virtuales, o incluso el desarrollo de un *digital twin* del sistema.

17. Conclusiones

Durante la ejecución del proyecto, aparecieron numerosos retos relacionados con limitaciones en la práctica y alteraciones de diseño no contempladas desde el principio. Específicamente, los problemas de precisión y la necesidad de un rediseño electromecánico alargaron el tiempo dedicado a la programación, ya que fue preciso incluir nuevos sensores junto con su lógica de control y las adaptaciones electrónicas necesarias.

Se concluye del proceso que para optimizar los recursos y reducir correcciones, es importante una planificación adecuada y un modelado estricto desde el principio. Esto se considera relevante en cada etapa del proyecto, desde la idea inicial hasta la implementación final.

En este desarrollo, implementar *topics* se consideró suficiente para cumplir con los requerimientos definidos. Si se define un comportamiento similar para los *callbacks* asociados a los mismos, puede llegar a haber algún conflicto. Esto se solventó definiendo una máquina de estados y manejo de colas *FIFO* en el RTOS que aseguró que los llamados sean no simultáneos.

Por limitaciones de tiempo, no se pudo terminar todas las funcionalidades proyectadas. Se decidió descartar el módulo MQTT, ya que al analizarlo se vio que el aporte de este a la eficiencia general del sistema era escaso en comparación con el esfuerzo adicional que suponía su incorporación.

El desarrollo del sistema robótico paralelo pick-and-place implementa tecnologías que se están usando en la industria e hizo posible comprobar la unión exitosa de cuatro campos esenciales: planificación de movimiento, visión artificial, control de motores Paso a Paso y gestión de sistema operativo en tiempo real (*FreeRTOS* + *microROS*). Durante el desarrollo del proyecto, se verificó que la arquitectura sugerida tiene la capacidad de identificar objetos en el área de trabajo, calcular las trayectorias requeridas y llevar a cabo los movimientos del manipulador con una precisión adecuada para realizar su función logística simulada.

Se lograron mediciones exactas gracias a la calibración de los sensores AS5600 y a la implementación de un proceso sistemático para referenciar los ángulos articulares, lo cual fue fundamental para el adecuado desempeño de la cinemática inversa. Además, los ensayos con el electroimán posibilitaron la determinación experimental de los límites operativos del sistema de sujeción.

En cuanto a la visión artificial, los ensayos realizados con métodos tradicionales de OpenCV y modelos de inferencia basados en YOLOv11n demostraron que se puede identificar y localizar objetos con buena exactitud, siempre que el sistema tenga una calibración espacial apropiada y una referencia constante de posición entre la cámara y el robot. Esto hizo posible obtener con precisión la ubicación de los objetos y convertirla al sistema de coordenadas del manipulador para llevar a cabo el ciclo completo de *pick-and-place*.

La cinemática inversa, que se calculó y validó en MATLAB al principio, demostró un correcto funcionamiento cuando fue trasladada al STM32, desde donde las instrucciones articulares se produjeron correctamente. Esta migración probó que fue factible implementar los algoritmos directamente en el hardware de control sin poner en riesgo la velocidad de respuesta del sistema. Aún así, cabe destacar que para tener mayor facilidad de validación en la generación de trayectorias, se optó por realizar la cinemática inversa en el orquestador (ROS2 en la computadora).

El conjunto de ensayos realizados (mecánicos, electrónicos, de comunicación, de control y de visión) hizo posible verificar que el robot funciona conforme al modelo teórico y que las resoluciones de diseño tomadas se convierten en un proceso estable. El sistema resultante ofrece una plataforma firme para futuros trabajos que se enfoquen en optimizar la velocidad, la exactitud del posicionamiento, la robustez del sistema de visión y la autonomía general del manipulador.

Referencias

- [BSp, 2001] (2001). B-splines and their properties. <https://pages.cs.wisc.edu/~deboor/toast/>. Accessed: 2026-10-15.
- [YOL, 2024] (2024). Ros 2 interfaces overview. <https://docs.ultralytics.com/models/yolo11/>. Accessed: 2025-06-15.
- [ros, 2025] (2025). Ros 2 interfaces overview. <https://docs.ros.org/en/humble/Concepts/Basic/About-Interfaces.html>. Accessed: 2025-02-20.
- [Barrientos et al., 2007] Barrientos, A., Peñín, L. F., Belaguer, C., and Aracil, R. (2007). *Fundamentos de Robótica*. McGraw Hill.
- [Barry, 2016] Barry, R. (2016). *Mastering the FreeRTOS Real Timer Kernel: A Hands-On Tutorial Guide*.
- [Belsare et al., 2023] Belsare, K., Rodriguez, A. C., Sánchez, P. G., Hierro, J., Ko lcon, T., Lange, R., Lütkebohle, I., Malki, A., Losa, J. M., Melendez, F., Rodriguez, M. M., Nordmann, A., Staschulat, J., , and von Mendel, J. (2023). *Micro-ROS*, pages 3–55. Springer.
- [Biagiotti and Melchiorri, 2008] Biagiotti, L. and Melchiorri, C. (2008). *Trajectory Planning for Automatic Machines and Robots*. Springer Publishing Company.
- [Corke, 2017] Corke, P. (2017). Robotics, vision and control: Fundamental algorithms in matlab® second, completely revised, extended and updated edition. *Springer Tracts in Advanced Robotics* №118.
- [Craig, 2006] Craig, J. J. (2006). *Robótica*. Pearson Educación.
- [Lynch, 2017] Lynch, F. C. P. K. M. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press.
- [Mark W. Spong, 2005] Mark W. Spong, Seth Hutchinson, M. V. (2005). *Robot Modeling and Control*. Wiley.
- [Pieper, 1968] Pieper, D. L. (1968). The kinematics of manipulators under computer control.
- [Torres et al., 2002] Torres, F., Pomares, J., Gil, P., Puente, S. T., and Aracil, R. (2002). *Robots y Sistemas Sensoriales*. Pearson Educación, S. A.
- [Zhou et al., 2020] Zhou, Z., Zhang, X., Gao, Y., Wu, D., and Dong, W. (2020). Design and kinematics analysis of a percutaneous needle insertion robotic system. *IOP Conference Series: Materials Science and Engineering*, 825(1):012009.

18. Anexo

18.1. Teorema del coseno

Ampliamente usado en geometría, es muy útil para realizar desarrollos de cinemática inversa geométrica de robots serie, como el caso de estudio. El mismo postula:

Sea un triángulo con lados a , b y c , opuestos a los ángulos A , B y C respectivamente, entonces:

$$c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos(\alpha)$$

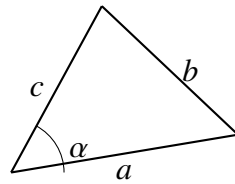


Figura 40: Triángulo general donde se aplica el Teorema del Coseno.

18.2. Colas FreeRTOS

Para poder comunicar entre distintas tareas o entre “callbacks” se usan colas de tipo FIFO (*First In, First Out*) el funcionamiento está explicado en detalle en [Barry, 2016]. Los elementos a almacenar en las colas pueden ser de distinto tamaño, por eso al crearlas se define el tamaño del dato a almacenar en cada elemento y la cantidad de elementos a almacenar.

El acceso a escribir datos en las colas se da según la prioridad de la tarea, y al llenarse el *buffer* se mantiene la prioridad. Una vez liberado el primer elemento, se hace un desplazamiento de elementos y la tarea con más prioridad asigna el elemento a la última posición de la cola. También existe el caso de que el elemento provenga de una interrupción (ISR), entonces este elemento pasa a tener mayor prioridad y se ubica en la primer posición de la cola. En la imagen 41 (presentada en [Barry, 2016]) se puede ver una secuencia básica de lectura y escritura en colas.

A continuación se muestra un ejemplo, en pseudocódigo, para la creación de colas en el hilo principal, antes de que el Scheduler tome el control:

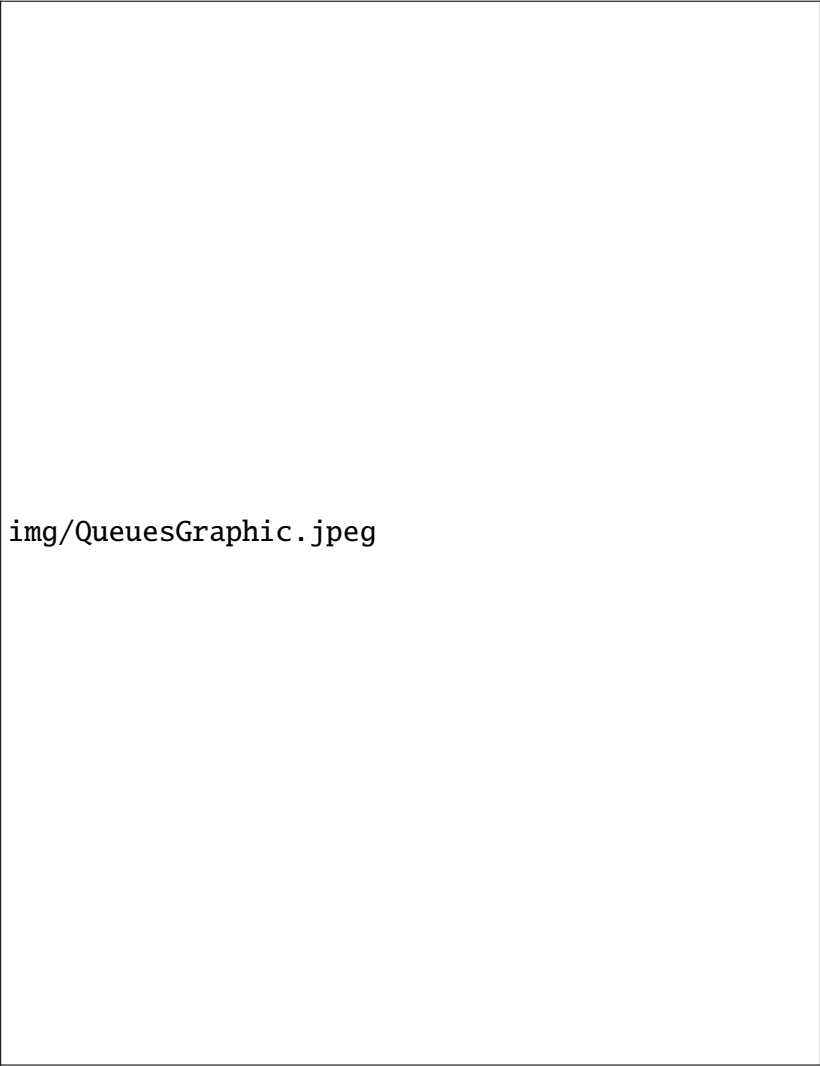
```

INICIAR creacion de colas de comunicacion
// Crear cola para posiciones cartesianas
crear cola PositionQueue con capacidad 1 elemento de tipo CARTESIAN_POS_t
si la cola no pudo crearse:
    registrar error "No se pudo crear PositionQueue"
// Crear colas individuales para cada motor
para i desde 0 hasta N_MOTORS - 1:
    crear cola JointQueue[i] con capacidad 1 elemento tipo float
    si la cola i no pudo crearse:
        registrar error "No se pudo crear JointQueue[i]"
FINALIZAR creacion de colas

```

Pseudocodigo: manejo de colas

Se debe especificar el tamaño de cada elemento y la cantidad de elementos que tiene cada cola.



img/QueuesGraphic.jpeg

Figura 41: Funcionamiento de colas

18.3. Interfaces internas de ROS

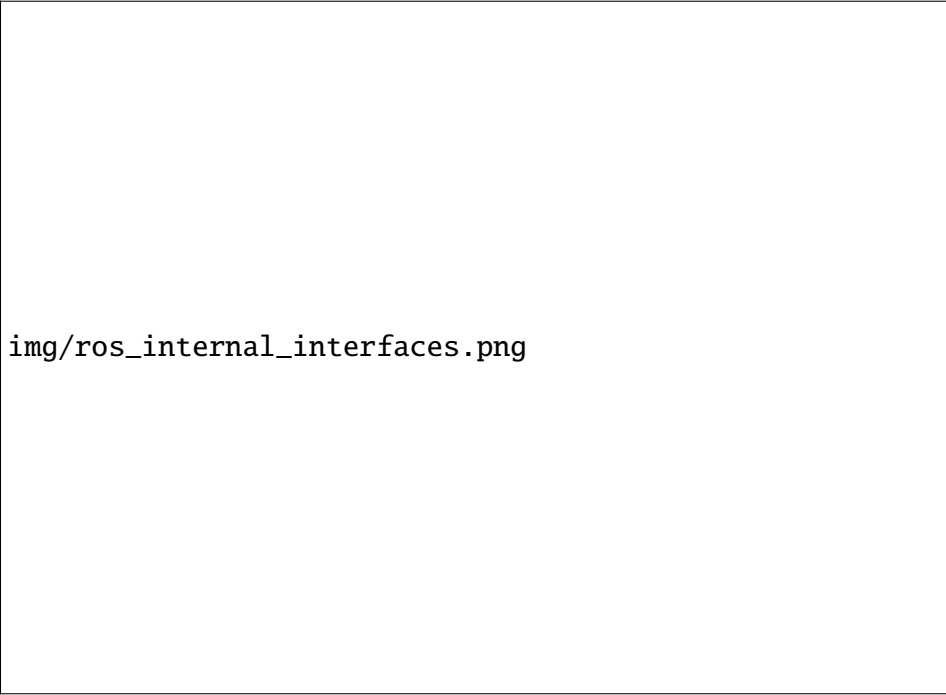
A nivel de capas de abstracción, ROS cuenta con dos que son las principales: la interfaz de librería de cliente, que es `rcl` (ROS Client Library), que brinda soporte a las librerías de cliente (la capa superior) tales como `rclcpp` y `rclpy`; por otro lado está la interfaz de *Middleware*, que abstrae a `rcl` de las capas de implementación de middleware específicas (Fast-DDS, Cyclone, etc)⁴.

18.4. Task de microROS

En el firmware, el RTOS ejecuta una tarea específica, en la cual está el núcleo de microROS. En primer lugar, se configura un transporte personalizado para microROS. En este caso, se emplea una interfaz UART y se registran explícitamente las funciones de apertura, cierre, lectura y escritura del canal físico. Posteriormente, se redefine el asignador de memoria por defecto de ROS2 mediante un conjunto de funciones compatibles con FreeRTOS.

A continuación, se inicializa la estructura `rclc_support`, se crea el nodo microROS y se definen

⁴DDS: Data Distribution Service. Es un middleware estandarizado



img/ros_internal_interfaces.png

Figura 42: Interfaces internas de ROS

los mensajes utilizados por los distintos canales de comunicación. Se instancian suscriptores y publicadores asociados a tópicos específicos, cada uno enlazado a un tipo de mensaje en particular. Finalmente, se configura un executor, el cual gestiona la ejecución de las funciones de callback cuando se reciben nuevos datos. El bucle principal de la tarea ejecuta periódicamente dicho executor, asegurando el procesamiento continuo de mensajes entrantes desde el agente ROS2, mientras que la interacción con el planificador del sistema operativo se controla mediante retardos explícitos. las funciones de finalización que se encuentran al final del bucle para realizar una limpieza de memoria y evitar fugas; en el funcionamiento normal del microcontrolador, no se llega hasta este punto. El código correspondiente se encuentra en el repositorio Proyecto Final de Estudios.

18.5. ROS DOMAIN

El **ROS_DOMAIN_ID** es una variable de entorno de ROS2 que permite segmentar una red física en múltiples redes lógicas independientes. Su función principal es evitar la interferencia entre diferentes grupos de robots o grupos de nodos que operan en la misma infraestructura, ya que solo los nodos que comparten el mismo identificador de dominio pueden descubrirse y comunicarse entre sí.

Para establecer la conexión, el middleware DDS utiliza este ID para calcular automáticamente los puertos **UDP** necesarios para el intercambio de datos y el descubrimiento de nodos. Específicamente, cada dominio abre puertos de **multicast** que permiten a los nodos “anunciarse” y encontrarse dentro de un grupo, además de puertos de **unicast** dedicados para la comunicación directa entre procesos. Al basarse en UDP, se tiene un descubrimiento dinámico y rápido en la red.

18.6. Función de control de trayectoria

Se muestra a continuación la función `trajectoryControl` en pseudocódigo, que ha sido implementada en el STM32 y es encargada de controlar el seguimiento preciso de la trayectoria. La función realiza las siguientes tareas:

1. Calcula el error de posición
2. Genera un comando de velocidad con feedforward y control proporcional
3. Convierte la velocidad a steps/s
4. Define la dirección según la velocidad
5. Escribe la dirección al pin digital
6. Establece la velocidad del PWM por motor

```

funcion trajectoryControl(q_ref, qd_ref, m):
    // Calcular error de posicion
    q_err = q_ref - m.currentAngle

    // Calcular comando de velocidad (feedforward + feedback proporcional)
    v_cmd = qd_ref + KP_POS * q_err

    // Convertir velocidad a Hz (pasos por segundo)
    v_hz = valor_absoluto(v_cmd) * DEG_TO_STEPS * m.i

    // Limitar velocidad maxima y minima
    si v_hz > 1000 entonces
        v_hz = 1000
    fin si

    si v_hz < 1 entonces
        v_hz = 1 // para evitar division por cero
    fin si

    // Determinar direccion del motor
    si v_cmd >= 0 entonces
        m.dir = ENCENDIDO // GPIO_PIN_SET
    si no
        m.dir = APAGADO // GPIO_PIN_RESET
    fin si

    // Actualizar direccion en el pin correspondiente
    EscribirPin(m.DIR_PORT, m.DIR_PIN, m.dir)

    // Configurar velocidad del motor paso a paso
    Stepper_SetSpeed(m.timer, m.timerChannel, v_hz)
fin funcion

```

Pseudocodigo de trajectoryControl