IMPLEMENTACIÓN ENTORNO DE CONTROL BASADO EN PYTHON PARA SU APLICACIÓN EN BRAZOS ROBÓTICOS, UTILIZANDO EL QARM COMO CASO DE ESTUDIO

Universidad Nacional de Rafaela (UNRaf)

Licenciatura en Automatización y Robótica

Autor: Jonatan Bonetti

Tutor: Esp. Martin Visintini

Fecha de presentación:13/02/2026

AGRADECIMIENTOS

A Bety, porque sin todo lo que me enseñó no sería quien soy hoy.

A mi familia, que siempre, de una manera u otra, me apoya, aunque muchas veces no comprendan del todo mis proyectos.

A Carloncho, porque si tuviera que elegir un ejemplo de lo que quiero llegar a ser como persona en muchos aspectos, sin dudas sería él.

A Martín, quien, más allá de ser el director de esta tesis, fue un compañero de trabajo excepcional, con quien aprendí mucho, y sin cuya colaboración este proyecto no hubiera sido posible.

RESUMEN:

El presente trabajo tiene como objetivo implementar el lenguaje Python como plataforma de control para brazos robóticos, explorando su potencial para desarrollar soluciones flexibles, actuales y escalables tanto a nivel educativo como industrial. Para validar esta implementación se utilizó como caso de estudio el brazo robótico QArm de Quanser, disponible en la Universidad Nacional de Rafaela (UNRaf), cuyo uso hasta el momento se encontraba limitado al entorno MATLAB. Esta restricción dificultaba la incorporación de nuevas tecnologías y reducía el alcance de los proyectos posibles.

La migración hacia Python permitió establecer una estructura de control más accesible y adaptable, aprovechando la amplia disponibilidad de librerías abiertas orientadas a robótica, visión artificial y procesamiento de datos. Sobre esta base se desarrollaron tres aplicaciones principales: una interfaz gráfica capaz de trazar, almacenar y ejecutar rutas; un módulo introductorio de cinemática inversa; y un sistema de seguimiento visual mediante cámara, integrando Python con OpenCV y Mediapipe para controlar el movimiento del brazo en tiempo real.

A lo largo del desarrollo se incorporaron además herramientas de inteligencia artificial como apoyo metodológico, contribuyendo a reducir tiempos de depuración y optimizar procesos de diseño sin reemplazar el análisis técnico necesario. Los resultados obtenidos demuestran que Python amplía las capacidades del QArm al tiempo que sienta las bases para futuros proyectos con mayores niveles de automatización, integración sensorial y escalabilidad hacia entornos robóticos más complejos.

ABSTRACT:

This work aims to implement the Python programming language as a control platform for robotic arms, exploring its potential to develop flexible, modern, and scalable solutions applicable to both experimental and industrial environments. To validate this approach, the Quanser QArm robotic arm — available at the Universidad Nacional de Rafaela (UNRaf) — was used as a case study, replacing the previous MATLAB-based workflow that limited experimentation and hindered the integration of new technologies.

The transition to Python enabled a more accessible and adaptable control structure, taking advantage of the vast ecosystem of open-source libraries related to robotics, computer vision, and automation. Based on this environment, three main applications were developed: a graphical interface capable of drawing, storing, and executing trajectories; an introductory inverse kinematics module; and a vision-based tracking system using a camera, integrating Python with OpenCV and Mediapipe to control the arm's movement in real time.

Throughout the development process, artificial intelligence tools were incorporated as methodological support, helping reduce debugging time and optimize design iterations without replacing the required technical reasoning. The results demonstrate that Python significantly expands the capabilities of the QArm and provides a foundation for future projects involving higher levels of automation, sensory integration, and scalability toward more complex robotic systems.

Índice

Índice de Figuras

Índice de Tablas

CAPÍTULO I

CAPÍTULO I: Introducción

1.1. Contexto general del proyecto

La robótica ha experimentado un crecimiento exponencial en los últimos años, impulsado por la necesidad de automatizar procesos, mejorar la precisión operativa y ampliar las capacidades de interacción entre humanos y máquinas. En este escenario, los sistemas robóticos manipuladores ocupan un lugar central tanto en la industria como en entornos académicos destinados a la formación y la investigación. La integración de herramientas digitales y lenguajes de programación modernos constituye un elemento clave para potenciar el uso de estos dispositivos y permitir su adaptación a diferentes contextos de desarrollo (Mordor Intelligence, 2025).

En el ámbito universitario, el uso de brazos robóticos con fines experimentales depende en gran medida de la flexibilidad y accesibilidad de las herramientas de programación disponibles. El brazo QArm, incorporado por la UNRaf para actividades de docencia e investigación, ha sido utilizado exclusivamente mediante interfaces provistas para MATLAB. Si bien este entorno ofrece algunos ejemplos funcionales, su dependencia de licencias específicas y su menor adopción entre estudiantes orientados hacia herramientas de software libre ha limitado el aprovechamiento pleno del dispositivo. Como resultado, la comunidad académica no ha podido desarrollar aplicaciones más complejas, integraciones externas o prácticas avanzadas de control que un lenguaje ampliamente difundido como Python sí podría facilitar.

La motivación de este proyecto surge de la necesidad de ampliar las posibilidades de uso del QArm habilitando su control mediante Python, permitiendo así un entorno de trabajo más

accesible, flexible y escalable. Esta elección se alinea con tendencias tecnológicas actuales y favorece el desarrollo de competencias altamente demandadas en el sector industrial, donde Python se ha consolidado como una herramienta clave para automatización, análisis de datos, visión artificial e integración con sistemas de control (Subuyuj Juarez, 2025)

Además, la adopción de Python como alternativa de control abre la puerta a futuros desarrollos tanto en el ámbito académico como en aplicaciones reales. Si la metodología implementada para el QArm demuestra ser eficaz, podría extrapolarse a manipuladores industriales adaptando la lógica desarrollada a entornos más exigentes. De esta manera, el proyecto no solo busca mejorar el uso del brazo dentro de la Universidad, sino también establecer bases para una línea de trabajo sostenible que combine teoría, práctica y desarrollo tecnológico avanzado.

1.2. Contexto general del uso de Python en Robótica

El lenguaje Python se ha consolidado como una de las herramientas más influyentes dentro del campo de la robótica contemporánea debido a su simplicidad sintáctica, su ecosistema extensible y su compatibilidad con tecnologías emergentes. Su crecimiento sostenido en los últimos años se debe, en gran medida, a la disponibilidad de bibliotecas orientadas al cálculo científico, la visión por computadora, la inteligencia artificial y la comunicación con hardware, lo cual lo convierte en un lenguaje altamente adaptable a distintos entornos de automatización (Corke, 2017).

En el caso particular de los brazos robóticos, Python presenta características que pueden ser ventajas técnicas. Su integración con librerías como NumPy, SciPy, OpenCV o PyTorch permite implementar cálculos esenciales para la cinemática directa e inversa, la planificación de trayectorias, el control de la posición y la interpretación de datos provenientes de sensores o cámaras. Esta capacidad de combinar control, percepción y análisis dentro de un mismo entorno de desarrollo favorece la creación de arquitecturas modulares y escalables, con aplicaciones potenciales tanto en entornos experimentales como industriales.

Python también destaca por su facilidad para comunicarse con hardware a través de APIs y SDK desarrollados por fabricantes de robots o por comunidades de código abierto. Esto permite controlar actuadores, acceder a flujos de datos en tiempo real y desarrollar interfaces personalizadas sin depender de plataformas propietarias. Asimismo, su compatibilidad natural con frameworks como ROS (Robot Operating System) lo posiciona como un estándar actual para proyectos que buscan interoperabilidad, flexibilidad y opciones de expansión futura (Quigley et al., 2009).

1.3. Necesidad de migración desde entornos cerrados hacia entornos abiertos

El desarrollo de aplicaciones para control robótico ha estado históricamente condicionado por el uso de entornos propietarios, diseñados por los mismos fabricantes de hardware. Si bien estas plataformas suelen ofrecer estabilidad y un conjunto de herramientas prediseñadas, también presentan limitaciones importantes: escasa flexibilidad, restricciones en la modificación del código, poca compatibilidad con bibliotecas actuales y dificultades para integrar tecnologías emergentes como visión artificial, aprendizaje automático o interfaces personalizadas. Estas

restricciones se vuelven más evidentes cuando se busca extender la funcionalidad original del sistema o explorarlo más allá de los ejemplos predefinidos (Corke, 2017).

La tendencia global en investigación y automatización se orienta cada vez más hacia el uso de lenguajes y frameworks de código abierto que permiten mayor control sobre la arquitectura del sistema y posibilitan una integración más fluida entre distintos componentes. En este contexto, Python se convierte en una alternativa especialmente atractiva al ofrecer un entorno accesible, altamente documentado y con una comunidad activa que impulsa su crecimiento continuo. A diferencia de los entornos cerrados, Python permite combinar control robótico con algoritmos avanzados, simulación, análisis de datos y procesamiento de imágenes, sin depender de licencias o herramientas propietarias (Quigley et al., 2009).

Migrar desde entornos cerrados hacia plataformas abiertas también facilita la interoperabilidad entre distintos sistemas robóticos, una característica clave en las arquitecturas actuales de automatización. Los robots ya no funcionan como unidades aisladas, sino como parte de celdas o líneas de producción donde es necesario que múltiples dispositivos, sensores y softwares puedan comunicarse entre sí. En este sentido, el uso de lenguajes abiertos como Python contribuye a generar sistemas más flexibles y escalables, capaces de adaptarse a nuevas tecnologías o requerimientos sin depender de actualizaciones comerciales o cambios impuestos por terceros.

1.4. Alcances y limitaciones del proyecto

El presente proyecto se centra en la implementación del lenguaje Python como entorno principal para el control de brazos robóticos, tomando como caso de estudio el brazo QArm. El alcance del trabajo incluye la configuración completa del entorno de desarrollo, el establecimiento de la comunicación con el robot, la elaboración de scripts de control básicos y avanzados, y la creación de ejemplos aplicados que demuestran las posibilidades de esta migración hacia un entorno abierto. Entre estos ejemplos se encuentran una interfaz para trazado y ejecución de trayectorias, un módulo de cinemática inversa y un sistema de visión artificial basado en seguimiento de movimiento.

El proyecto también abarca el análisis comparativo entre el funcionamiento original del robot en MATLAB y su desempeño bajo Python, evaluando aspectos como flexibilidad, capacidad de integración, velocidad de desarrollo, escalabilidad y potencial de expansión hacia aplicaciones futuras. Si bien el brazo QArm es el dispositivo utilizado para validar la propuesta, los principios y técnicas implementados han sido diseñados con la intención de ser transferibles a otros manipuladores robóticos, incluyendo sistemas industriales con arquitecturas más complejas.

Sin embargo, el trabajo presenta ciertas limitaciones inherentes al hardware disponible y a las características mecánicas del QArm. La precisión en la ejecución de trayectorias, especialmente en tareas relacionadas con la cinemática inversa, se ve condicionada por la estructura del robot y por la ausencia de sensores adicionales que permitan retroalimentación en tiempo real. Del mismo modo, la capacidad de carga y el rango de movimiento restringen la

posibilidad de simular funciones industriales más exigentes. También se consideran como limitaciones el tiempo de desarrollo disponible y el alcance académico del proyecto, que prioriza la validez técnica y la demostración de concepto antes que la implementación de una solución industrial completa.

Aun con estas restricciones, el proyecto establece una base sólida para el uso de Python en el control de brazos robóticos y abre la puerta a desarrollos futuros que podrían extender sus capacidades hacia aplicaciones industriales, integración con ROS, incorporación de algoritmos de inteligencia artificial o el diseño de interfaces de operación más avanzadas.

1.5. Objetivos generales

Evaluar la viabilidad del uso de Python como entorno de control para brazos robóticos, utilizando el QArm como caso de estudio, con el fin de analizar las ventajas, posibilidades de expansión y capacidades de integración que ofrecen los entornos abiertos frente a plataformas propietarias.

1.5.1. Objetivos específicos:

Para alcanzar el objetivo general planteado, se establecen los siguientes objetivos específicos:

- Analizar la arquitectura de hardware y software del QArm, identificando sus capacidades, restricciones y requerimientos para su control desde entornos de programación basados en Python.

- Evaluar la viabilidad y robustez de una solución de control basada en Python para establecer comunicación, enviar comandos y gestionar el comportamiento del brazo robótico de manera modular y escalable.

- Analizar el alcance y las capacidades de las librerías y funciones desarrolladas en Python para el control del QArm, considerando movimientos básicos, trayectorias y operaciones sobre sus ejes, actuadores y sensores.

- Examinar el aporte de una interfaz de usuario al proceso de interacción con el QArm, evaluando su impacto en la facilidad de uso, la configuración de parámetros y la ejecución de acciones en contextos educativos y de laboratorio.

- Evaluar el desempeño del sistema propuesto mediante pruebas experimentales, considerando criterios de precisión, estabilidad y facilidad de uso en entornos educativos y de laboratorio.

- Analizar la importancia de la documentación y los lineamientos generados como soporte para el mantenimiento, la ampliación y la reutilización del sistema en futuras prácticas académicas y proyectos de investigación.

CAPÍTULO II

CAPÍTULO II: Marco conceptual

2.1. Introducción a los brazos robóticos y sus sistemas de control

Los brazos robóticos constituyen uno de los elementos centrales dentro de la robótica moderna. Su diseño se inspira en la anatomía del brazo humano, incorporando eslabones articulados y una cadena cinemática que permite ejecutar movimientos precisos y repetitivos. Estos dispositivos se utilizan en una amplia variedad de industrias, desde la manufactura y la logística hasta la medicina y la investigación científica (Craig, 2005).

En términos generales, un brazo robótico está compuesto por una estructura mecánica con varios grados de libertad, un sistema de actuadores (generalmente servomotores o motores eléctricos) y sensores que proporcionan información sobre posición, velocidad o fuerza. La combinación de estos elementos permite controlar el movimiento del efector final, ya sea para manipulación, ensamblaje, clasificación, transporte u otras tareas específicas.

El sistema de control es el componente que coordina estas funciones. A través de algoritmos de control cinemático y dinámico, se interpreta la información sensorial y se generan las señales necesarias para que el brazo realice movimientos seguros y eficientes. Estos sistemas pueden implementarse mediante plataformas cerradas, que limitan el acceso a parámetros internos, o mediante herramientas abiertas y programables, que permiten al usuario intervenir directamente en la lógica de control.

A medida que la robótica avanza, crece la demanda por sistemas de control más flexibles, transparentes y escalables, capaces de adaptarse a distintos contextos productivos. En este

escenario, lenguajes como Python adquieren un rol relevante al ofrecer un puente accesible entre el usuario y los algoritmos que gobiernan el movimiento del brazo robótico.

2.2. Restricciones operativas del QArm bajo entornos cerrados

Hasta el momento, el QArm ha sido utilizado en UNRaf principalmente mediante el entorno de programación MATLAB y su plataforma asociada Simulink, ambos desarrollados por la empresa MathWorks. Este enfoque ha permitido realizar pruebas básicas de control, calibración y movimiento del brazo, utilizando ejemplos preconfigurados provistos por la empresa Quanser.

La estructura de control basada en MATLAB se fundamenta en el uso de bloques funcionales dentro de Simulink, que representan las diferentes articulaciones del robot y sus señales de entrada y salida. Cada bloque comunica con el hardware físico del QArm a través de la interfaz QDrive, la cual gestiona la transmisión de datos entre el software y los servomotores.

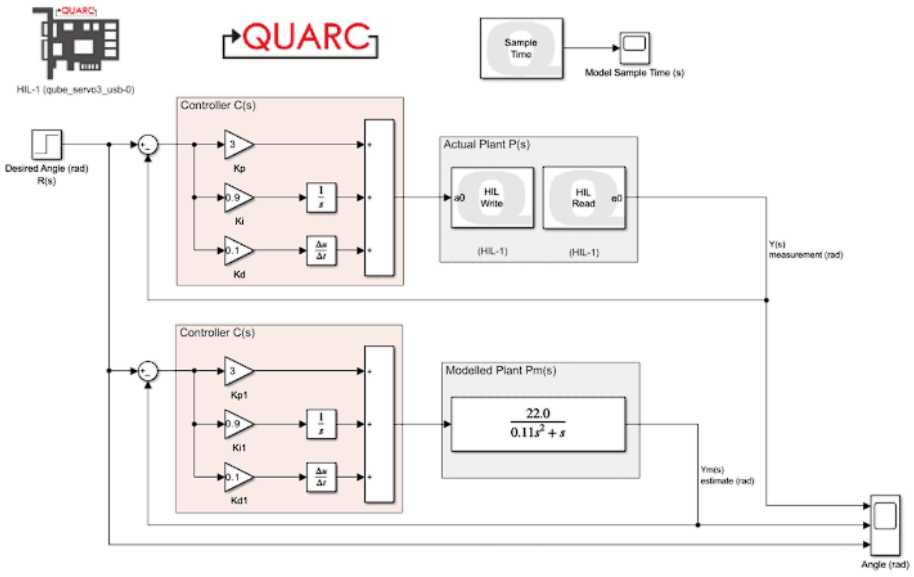

Figura 1. Recorte SimuLink - Fuente: Elaboración propia

Mediante esta arquitectura, es posible implementar rutinas de posicionamiento, secuencias de movimiento y control de la pinza, generalmente con fines didácticos o de demostración.

El uso de entornos propietarios como MATLAB y Simulink para el desarrollo de aplicaciones de control robótico presenta una serie de limitaciones que impactan tanto en el ámbito académico como en el experimental. En primer lugar, estos entornos dependen de licencias institucionales o individuales de costo elevado, lo que restringe su disponibilidad fuera de los laboratorios universitarios y limita la continuidad del aprendizaje por parte de los estudiantes en contextos extracurriculares.

Asimismo, si bien Simulink ofrece una estructura modular orientada al modelado gráfico, esta característica puede convertirse en una desventaja cuando se requiere un mayor grado de personalización. La modificación del código subyacente, la incorporación de nuevas funcionalidades o la integración de bibliotecas externas no siempre resulta directa, y en muchos casos exige un conocimiento profundo del entorno. A esto se suma que la documentación disponible suele centrarse en ejemplos específicos, lo que dificulta su adaptación a desarrollos no contemplados originalmente y aumenta la complejidad del proceso de aprendizaje.

Otro aspecto relevante es la curva de aprendizaje inicial, que puede resultar elevada para usuarios que recién se inician en la programación o que provienen de áreas ajenas al ecosistema MATLAB. La necesidad de comprender simultáneamente el lenguaje de programación y la

lógica del entorno gráfico de Simulink puede constituir una barrera adicional en etapas tempranas de formación.

Por otra parte, la integración con tecnologías modernas como inteligencia artificial, visión por computadora o aprendizaje automático no se presenta de forma natural dentro del entorno MATLAB, especialmente en comparación con lenguajes y frameworks de código abierto. Si bien existen toolboxes específicos, su uso suele estar condicionado por licencias adicionales y por una menor flexibilidad en la combinación con bibliotecas externas ampliamente utilizadas en la comunidad científica.

Estas limitaciones reducen el potencial del QArm como herramienta educativa y de investigación dentro de la UNRaf, al depender de un entorno cerrado y costoso. En consecuencia, surge la necesidad de incorporar Python como alternativa de programación, aprovechando su naturaleza abierta, su amplia comunidad y su compatibilidad con librerías modernas orientadas al control, la automatización y la robótica.

2.3. Librerías y ecosistemas Python aplicables (Quanser, OpenCV, NumPy, etc.)

El ecosistema de Python se ha consolidado como uno de los más amplios y versátiles dentro del ámbito del desarrollo tecnológico y científico. Su comunidad activa y la disponibilidad de paquetes especializados permiten abordar de manera integrada diversos aspectos del control robótico, desde procesamiento de señales hasta visión artificial y aprendizaje automático. En el caso particular de la manipulación robótica, esta diversidad de herramientas

representa una ventaja significativa frente a entornos más cerrados o con menos posibilidades de extensión.

Entre las librerías más relevantes para este proyecto se encuentra la API de Quanser para Python, que permite establecer comunicación directa con los módulos del QArm y acceder a funciones de bajo nivel para el control de articulaciones, lectura de sensores y ejecución de trayectorias. Esta capa de software constituye el punto de partida para reemplazar la dependencia exclusiva del entorno MATLAB/Simulink y habilitar un flujo de trabajo más flexible (Quanser Inc., 2024).

Por otro lado, Python ofrece un conjunto de herramientas científicas, entre las cuales se destacan NumPy y SciPy, fundamentales para el tratamiento eficiente de datos, la implementación de cálculos matriciales y la resolución de algoritmos asociados al control robótico. Estas bibliotecas permiten diseñar y evaluar cinemáticas, aplicar transformaciones geométricas, filtrar señales y simular movimientos con gran eficiencia computacional (Python Software Foundation, 2024).

Asimismo, el proyecto contempla el uso de librerías orientadas al procesamiento de imágenes, siendo OpenCV una de las más difundidas tanto en investigación como en la industria. Su incorporación abre la posibilidad de explorar tareas de visión artificial (como detección de objetos, segmentación o seguimiento) que pueden integrarse de manera nativa con el control del robot, ampliando así el alcance funcional del sistema.

El ecosistema Python incluye múltiples recursos adicionales, como Matplotlib para visualización, Pandas para el manejo estructurado de datos y librerías orientadas a la inteligencia artificial como TensorFlow o PyTorch, que pueden considerarse para futuras ampliaciones del proyecto. La interoperabilidad entre todos estos componentes pone de manifiesto la capacidad de Python para conformar un entorno de desarrollo integral, modular y escalable, adecuado tanto para aplicaciones educativas como para desafíos propios de la automatización industrial contemporánea.

2.4. Caso de estudio: descripción técnica del QArm

Figura 2. Brazo QArm - Fuente: Quanser

En la figura 2, se observa el brazo robòtico QArm, es un manipulador robótico desarrollado por la empresa Quanser, diseñado con fines educativos y de investigación en el

campo de la robótica, la automatización y el control. Forma parte de la línea Quanser Interactive Labs (QLabs), una plataforma que combina hardware físico y simulación virtual para el desarrollo de experiencias prácticas en programación y control de sistemas mecatrónicos (Quanser Inc., 2024).

El QArm es un brazo robótico de 4 grados de libertad (GDL), accionado mediante servomotores de precisión, que reproduce las características básicas de un manipulador industrial a escala de laboratorio. Está compuesto por una base rotatoria, un brazo inferior, un brazo superior y una muñeca con pinza, permitiendo movimientos coordinados en los ejes de rotación, elevación y extensión.

Entre sus principales características técnicas se destacan:

Parámetro	**Valor**
Peso del manipulador	8,25 kg
Carga útil	350 – 750 g
Alcance	750 mm
Repetibilidad	± 0,05 mm
Cámara	Intel® RealSense™ D415
Interfaz	USB (QFLEX 2)
Modos de control internos	Modo posición, modo corriente
Frecuencia de control externo	500 Hz

Frecuencia de control interno (mín.)	1000 Hz
Entradas/Salidas expandibles	PWM / Analógico / I²C / SPI / UART
Rango mínimo y máximo de articulaciones	Base: ± 170°Hombro: ± 85°Codo: −95° / +75°Muñeca: ± 160°
Velocidad máxima de las articulaciones	± 90°/s

Tabla 1. Características técnicas QArm

En cuanto a sus capacidades, el QArm permite la ejecución de tareas típicas de un manipulador robótico, tales como movimiento punto a punto, control de trayectorias, manipulación de objetos y calibración de posiciones. Estas funciones lo convierten en una herramienta sumamente útil para la enseñanza de conceptos de cinemática, dinámica, control y programación de robots.

Gracias a su diseño modular y a la compatibilidad con diferentes entornos de programación (como MATLAB, Simulink y ahora Python), el QArm se posiciona como una plataforma educativa versátil, ideal para la integración con tecnologías emergentes y el desarrollo de proyectos interdisciplinarios en robótica.

2.5. Comunicación hardware–software en controladores educativos

La comunicación entre el hardware del robot y el software de control es un aspecto fundamental en cualquier sistema robótico, especialmente en plataformas educativas, donde la arquitectura debe ser accesible, segura y comprensible para estudiantes y desarrolladores. Este tipo de entornos prioriza la facilidad de uso y la estabilidad, permitiendo que los usuarios puedan

interactuar con el robot sin requerir un conocimiento avanzado de protocolos industriales. Sin embargo, esta simplificación suele implicar ciertas restricciones que condicionan la flexibilidad y las posibilidades de expansión.

En los brazos robóticos educativos, como el QArm, el fabricante provee una interfaz de comunicación prediseñada que actúa como intermediaria entre el usuario y los actuadores del dispositivo. Esta capa de abstracción, implementada generalmente mediante APIs o SDK propietarios, traduce los comandos de alto nivel (como mover un eje, activar una pinza o ejecutar una trayectoria predefinida) en señales adecuadas para los motores, encoders y sensores del robot. Si bien este enfoque reduce la complejidad del control y evita errores críticos, también limita el acceso directo al hardware, restringiendo la personalización y la integración con otras tecnologías.

El uso de estas interfaces cerradas suele ser adecuado para entornos académicos que buscan brindar una experiencia inicial en robótica, pero resulta menos conveniente cuando se pretende avanzar hacia estrategias más avanzadas, como control en lazo cerrado, algoritmos basados en percepción, o integración con sistemas externos. En muchos casos, las APIs disponibles se encuentran diseñadas para funcionar únicamente dentro de un ecosistema específico, como MATLAB o un entorno propietario, lo que dificulta su adaptación a lenguajes más extendidos como Python.

En este contexto, comprender la lógica de comunicación hardware–software resulta esencial para implementar soluciones alternativas. La migración hacia un entorno abierto, como

Python, implica desarrollar mecanismos propios de envío de comandos, lectura de estados y coordinación de tareas, respetando las restricciones del dispositivo y manteniendo la estabilidad del sistema. Este proceso no solo implica traducir funciones de una API a otra, sino también entender cómo se estructuran los mensajes, qué parámetros requiere el hardware, cómo se gestionan los tiempos de muestreo y qué limitaciones impone el controlador interno del robot.

La experiencia con controladores educativos ofrece una base sólida para proyectos más avanzados. Las mismas lógicas de comunicación implementadas en kits académicos pueden escalarse a manipuladores industriales, donde la interacción hardware–software adquiere una dimensión más compleja, pero sigue sosteniéndose en principios similares: envío de comandos estructurados, lectura de estados, sincronización de procesos y diseño de arquitecturas modulares. Por ello, el estudio de estos mecanismos en plataformas educativas representa un paso clave hacia la comprensión de sistemas robóticos de mayor complejidad y hacia el desarrollo de soluciones abiertas, flexibles y escalables (Craig, 2005).

2.6 Cinemática inversa (IK) y cinemática directa (FK)

La cinemática inversa (IK) es el proceso mediante el cual se determinan los valores articulares necesarios para que el actuador final de un robot alcance una posición y orientación deseada en el espacio.

Mientras que la cinemática directa (FK) parte de las articulaciones para obtener la posición cartesiana, la cinemática inversa resuelve el problema opuesto: encontrar los ángulos de cada articulación a partir de las coordenadas deseadas.

En manipuladores robóticos como el QArm, este proceso presenta dos desafíos principales:

- **No siempre existe una solución real:** ciertas posiciones están fuera del espacio alcanzable del robot.
- **Puede haber múltiples soluciones válidas**: distintas configuraciones articulares pueden llevar al mismo punto (por ejemplo, “codo arriba” o “codo abajo”).

2.6.1 Estructura cinemática del QArm

El QArm es un manipulador de 4 grados de libertad (DOF) más una quinta articulación correspondiente al gripper.

Sus articulaciones son:

- Base (rotación)
- Hombro (rotación)
- Codo (rotación)
- Muñeca (rotación – Gamma)
- Gripper (apertura/cierre)

Esta estructura corresponde a un manipulador serial típico, con cinemática relativamente compleja debido a su combinación de enlaces y restricciones mecánicas, especialmente en las articulaciones del hombro y el codo.

2.6.2 Relación entre posición cartesiana y configuración articular

Para controlar el robot utilizando coordenadas X, Y, Z y orientación γ, se necesita transformar el punto deseado en el espacio tridimensional a un conjunto de ángulos articulares.

En términos generales:

$$\phi = f^{-1}(X, Y, Z, \gamma)$$

donde:

ϕ = vector de posiciones articulares

f^{-1} = función de cinemática inversa

Este cálculo implica resolver un sistema de ecuaciones no lineales, ya que el movimiento de cada articulación afecta la posición final del efector.

2.6.3 Naturaleza del problema y estrategias de resolución

El problema de la cinemática inversa puede abordarse, en términos generales, mediante dos enfoques principales, cada uno con características, ventajas y limitaciones propias. Por un lado, se encuentran las soluciones analíticas, que se basan en la obtención de expresiones matemáticas cerradas para calcular directamente los valores de cada articulación a partir de una posición deseada del actuador final. Este enfoque presenta como principal ventaja su elevada rapidez de cálculo, ya que no requiere procesos iterativos ni búsquedas numéricas. Sin embargo, la derivación de estas ecuaciones suele ser compleja y altamente dependiente de la geometría del robot, lo que limita su aplicación a manipuladores con estructuras relativamente simples. Además, estas soluciones resultan poco flexibles ante modificaciones del sistema, como cambios en la configuración mecánica o la incorporación de nuevas restricciones.

Por otro lado, las soluciones numéricas abordan la cinemática inversa como un problema de optimización, en el cual se buscan los valores articulares que minimicen el error entre la posición deseada del efector final y la posición efectivamente alcanzada. Este enfoque se caracteriza por su gran flexibilidad, ya que puede aplicarse a robots con geometrías complejas y permite incorporar de manera relativamente sencilla restricciones mecánicas, límites articulares o criterios adicionales. Como contrapartida, los métodos numéricos requieren un mayor esfuerzo computacional y, en ciertos casos, pueden converger hacia soluciones no ideales, como mínimos locales, lo que exige un diseño cuidadoso del algoritmo y de las condiciones iniciales.

2.6.4 Enfoque utilizado en este proyecto

El QArm, mediante su librería oficial QArmUtilities, utiliza un método numérico optimizado, diseñado por Quanser específicamente para su arquitectura mecánica.

Entradas:

- Recibe la posición deseada (X,Y,Z)
- Recibe la orientación deseada
- Usa la posición articular actual como condición inicial (esto mejora la estabilidad y evita saltos bruscos)

Salidas:

- Genera todas las configuraciones posibles mediante qarm_inverse_kinematics()
- Selecciona la solución más adecuada y libre de colisiones internas
- Devuelve un vector articular listo para ejecutar en el robot

Esto permite controlar al QArm desde Python sin necesidad de derivar ecuaciones analíticas complejas, logrando movimientos fluidos y precisos incluso durante cambios continuos en la GUI.

2.6.5 Importancia de la cinemática directa en el proceso

La cinemática directa (FK) se utiliza internamente para verificar que la solución obtenida por IK realmente alcanza la posición deseada, muestra en la interfaz los valores cartesianos reales y ayudar a depurar errores o inconsistencias

El uso conjunto de IK y FK asegura coherencia entre los movimientos planeados y los realmente ejecutados por el robot.

La cinemática inversa permite controlar el QArm de forma intuitiva usando posiciones en lugar de ángulos. Este proyecto implementa dicho control aprovechando la API oficial de Quanser, la cual utiliza algoritmos numéricos robustos optimizados para el hardware.

Gracias a esto, fue posible desarrollar una interfaz gráfica que mueve el robot en tiempo real sin requerir derivaciones matemáticas complejas, manteniendo precisión y estabilidad operativa.

2.7 Antecedentes y proyectos similares (Estado del arte)

En la última década Python ha pasado de ser una herramienta de scripting a una plataforma central para el desarrollo robótico: su ecosistema facilita el control, la percepción y la

integración con frameworks como ROS, lo que lo hace apto tanto para prototipado académico como para despliegues en entornos industriales cuando se combina con las capas de software correctas (Macenski et al., 2022).

2.7.1 Plataformas y APIs

Vendedores de hardware y proveedores académicos han publicado APIs y SDKs en Python para facilitar la adopción. Quanser, por ejemplo, ofrece documentación y APIs oficiales en Python para sus dispositivos (incluyendo soporte para QLabs/QArm), lo cual simplifica el acceso al hardware desde scripts y aplicaciones Python. De forma paralela, existen SDKs y paquetes en GitHub que permiten a desarrolladores integrar controladores y drivers en Windows/Linux usando Python. Estas iniciativas convierten a Python en una vía válida y soportada por proveedores para interactuar con manipuladores reales (Quanser Consulting Inc., 2025).

2.7.2 ROS + Python: puente hacia aplicaciones industriales

ROS (y ROS 2) es el principal middleware para robótica que admite ampliamente Python (rospy/ros2 Python APIs). Numerosos tutoriales y cursos muestran cómo controlar robots reales con scripts Python, y muchas integraciones industriales emplean Python en la capa de orquestación, visión y planificación, mientras que conservan componentes de tiempo real en C/C++ cuando es necesario. Esto convierte a Python en la “capa de integración” en arquitecturas heterogéneas (Macenski et al., 2022).

2.7.3 Visión artificial + control: proyectos representativos

Hay abundantes trabajos académicos y proyectos prácticos que usan OpenCV + Python para dotar a brazos robóticos de capacidades de percepción (detección, seguimiento, pick-and-place). Estos proyectos cubren desde prototipos con cámaras simples y microcontroladores hasta implementaciones con cámaras RGB-D y redes de ML para reconocimiento. Son especialmente frecuentes en tesis y trabajos finales que integran visión con control de trayectoria o agarre (Sharma & Mital, 2021).

2.7.4 Cinemática e/inversa: librerías y herramientas en Python

Existen paquetes robustos para cinemática y control en Python, como la Robotics Toolbox for Python (implementaciones de cinemática inversa o también conocida como IK , dinámicas y utilidades para manipuladores) y diversas bibliotecas en GitHub que generan o resuelven IK para manipuladores reales. Estas herramientas facilitan experimentar con IK y resolver problemas prácticos sin desarrollar todo desde cero (Haviland & Corke, 2023).

2.7.5 Proyectos de integración práctica (casos de uso)

En el ámbito de los proyectos de integración práctica pueden encontrarse numerosos casos de uso en los que Python se emplea como lenguaje principal para el control y la extensión de sistemas robóticos. En plataformas educativas, diversos trabajos académicos y tesis de grado documentan la integración de brazos robóticos con Python para el desarrollo de aplicaciones que combinan control de trayectoria, seguimiento visual y creación de interfaces gráficas de usuario. En este tipo de proyectos, la utilización conjunta de Python y librerías como OpenCV permite

incorporar capacidades de percepción visual que enriquecen las prácticas formativas y amplían el alcance de los experimentos más allá de los ejemplos predefinidos por los fabricantes.

De manera paralela, en entornos industriales y en el campo de la robótica colaborativa, se observan experiencias exitosas donde los fabricantes ponen a disposición APIs basadas en Python para facilitar la interacción con cobots y manipuladores modernos. Estas interfaces permiten implementar tareas de supervisión, visión artificial y automatización de procesos mediante scripts, reduciendo significativamente los tiempos de desarrollo y validación. En muchos casos, Python se utiliza como herramienta de prototipado y prueba piloto, posibilitando evaluar algoritmos y flujos de trabajo antes de su migración a capas de control en tiempo real o a lenguajes de más bajo nivel, manteniendo así un equilibrio entre flexibilidad y rendimiento (Poncelas Bodelón, 2014).

2.7.6 Limitaciones y brechas observadas

A pesar de las ventajas que ofrece Python en el desarrollo de aplicaciones robóticas, existen ciertas limitaciones que deben considerarse al momento de su adopción, especialmente en contextos industriales o de control avanzado. Una de las principales brechas está relacionada con el determinismo y el tiempo real. Python no está diseñado para cumplir requisitos estrictos de hard real-time, por lo que en aplicaciones críticas suele emplearse únicamente en capas de alto nivel. En estos casos, el control de lazo rápido y las tareas con requerimientos temporales estrictos se delegan a implementaciones en C/C++ o a controladores industriales como PLCs, mientras que Python se utiliza para planificación, percepción, supervisión y orquestación del sistema.

Otra limitación relevante es la fragmentación existente en las interfaces de programación disponibles. Actualmente no existe un estándar único que define una "API Python para brazos robóticos", lo que deriva en soluciones heterogéneas desarrolladas por distintos fabricantes o comunidades. Esta diversidad de enfoques implica diferencias en protocolos, estructuras de datos y capacidades, obligando en muchos casos a desarrollar adaptadores específicos o capas intermedias que permitan unificar el acceso al hardware dentro de un mismo proyecto (Macenski et al., 2022).

2.7.7 Oportunidades y tendencias relevantes

En el contexto actual, se identifican oportunidades claras que refuerzan la pertinencia del enfoque adoptado en esta tesis. En primer lugar, el hecho de que Quanser proporcione herramientas oficiales y documentación para el uso de Python facilita significativamente el proceso de integración con el QArm y legitima este enfoque como una alternativa práctica, replicable y alineada con las tendencias actuales en entornos educativos y de investigación. La disponibilidad de APIs compatibles con Python reduce la dependencia de plataformas propietarias y habilita flujos de trabajo más flexibles y extensibles.

Por otro lado, el ecosistema de Python ofrece un soporte sólido para el desarrollo de aplicaciones que integran control robótico con visión artificial e inteligencia artificial. El uso de librerías consolidadas como OpenCV, junto con frameworks de aprendizaje automático como PyTorch o TensorFlow, permite incorporar tareas de percepción, seguimiento y reconocimiento de manera directa en el control del brazo robótico. Este tipo de integraciones aparece con frecuencia en publicaciones recientes y proyectos académicos, lo que evidencia una tendencia

consolidada hacia sistemas robóticos que combinan percepción y acción dentro de un mismo entorno de desarrollo.

Además, la disponibilidad de paquetes especializados para cinemática y control en Python representa una ventaja significativa para el desarrollo del proyecto. Bibliotecas como Robotics Toolbox for Python, junto con diversos desarrollos de código abierto orientados a la resolución de cinemática inversa y control de manipuladores, permiten acelerar la implementación de algoritmos y apoyarse en soluciones previamente validadas por la comunidad. Este enfoque contribuye a reducir el tiempo de desarrollo y a mejorar la robustez técnica de las soluciones propuestas.

2.7.8 Antecedentes

2.7.8.1 "Automatización de un sistema identificador y posicionador de objetos a través de un Brazo robótico mediante Visión Artificial con Lenguaje Python"

Esta tesis desarrolla un sistema educativo de laboratorio en el que un brazo robótico es controlado con Python y visión artificial para detectar y mover objetos según su color. Se implementa programación en Python, lógica de cinemática, control de trayectoria y clasificación visual integrada con el manipulador para pruebas experimentales (Llerena Buenaño & Salazar Villamar, 2022).

2.7.8.2 “Desarrollo de un módulo educativo para la clasificación de objetos utilizando un brazo robótico y visión artificial”

Este trabajo propone un módulo educativo donde se controla un brazo robótico integrando Python en una Raspberry Pi para realizar clasificación de objetos por forma y color, combinando algoritmos de visión artificial y control de movimiento con una interfaz intuitiva (Cantos Párraga & Guerrero Flores, 2024).

2.7.8.3 “Desarrollo del software de control de un brazo robotizado mediante Python”

La tesis presenta el desarrollo de software de control para un brazo robótico industrial empleando Python como lenguaje principal, incluyendo comunicación con el hardware, generación de trayectorias y control dinámico desde una aplicación desarrollada en Python, demostrando la viabilidad de este enfoque (Poncelas Bodelón, 2014).

2.7.8.4 “Desarrollo de sistema para el control de un brazo robótico impreso en 3D mediante api de programación e interfaz gráfica”

Este trabajo describe la creación de un sistema de control accesible para un brazo robótico impreso en 3D usando APIs en Python junto con una interfaz web, permitiendo movimientos precisos, programación de secuencias y funciones de manipulación mediante software de alto nivel (Abarzúa Poblete, 2023).

CAPÍTULO III

CAPÍTULO III: Metodología

3.1 Enfoque general del proyecto

El enfoque general de este proyecto se basa en el desarrollo de una solución integral que permita controlar un brazo robótico mediante el lenguaje Python, en nuestro caso QArm de Quanser, priorizando la flexibilidad, la modularidad y la posibilidad de expansión hacia aplicaciones más avanzadas. Para ello, se adopta una metodología orientada al diseño iterativo, en la cual cada etapa del desarrollo (desde la comunicación básica con el hardware hasta la implementación de interfaces de usuario y módulos de cinemática) se valida progresivamente a través de pruebas experimentales.

El proyecto se estructura bajo una lógica de construcción por capas, comenzando por la comprensión del funcionamiento interno del dispositivo y el análisis de su entorno original de operación. Sobre esa base, se desarrolla una arquitectura de software en Python que reemplaza las restricciones del entorno cerrado previamente utilizado por un ecosistema abierto, accesible y capaz de integrarse con herramientas de visión artificial, planificación de movimiento y análisis de datos.

Este enfoque combina tanto elementos teóricos como prácticos. Por un lado, incluye el estudio de bibliotecas, protocolos de comunicación y principios de control necesarios para garantizar un funcionamiento estable del sistema. Por otro, prioriza la experimentación directa con el robot, lo cual permite identificar limitaciones reales del hardware, validar los modelos implementados y ajustar los algoritmos de acuerdo con el comportamiento observado.

La propuesta también contempla la escalabilidad como eje central. Si bien el QArm constituye el caso de estudio principal, el diseño del sistema se realiza de manera que pueda adaptarse con mínimas modificaciones a otros manipuladores, incluidos brazos industriales. Este criterio orienta la organización del código, la separación de funciones y la definición de interfaces de control claras, permitiendo extender el trabajo en futuros proyectos o integrarlo con sistemas externos.

En conjunto, el enfoque adoptado busca garantizar que la implementación en Python sea robusta, replicable y técnicamente fundamentada, contribuyendo no solo al aprovechamiento académico del QArm, sino también a la construcción de una base metodológica aplicable a contextos de automatización más exigentes.

3.2 Entorno de desarrollo en Python

El desarrollo del sistema de control se llevó a cabo utilizando un entorno de trabajo basado íntegramente en Python. El entorno se configuró con versiones estables de las principales dependencias del proyecto, asegurando compatibilidad tanto con la simulación como con el hardware real del robot.

Para la ejecución del software se utilizó Python 3.12, junto con un conjunto de paquetes que conforman la base del desarrollo:

- **NumPy (2.2.6)**: Empleado para cálculos matriciales, operaciones vectoriales y manipulación de posiciones articulares.

- **OpenCV (4.11.0):** Utilizado en la parte del proyecto destinada a visión y seguimiento, aportando herramientas de captura, procesamiento y análisis de imágenes.

- **Tkinter (8.6):** Biblioteca estándar de Python para la creación de la interfaz gráfica del usuario (GUI), integrada posteriormente en el módulo Graphic_interface.py.

Quanser SDK y drivers HIL, utilizados por **QArm_lib.py** para establecer comunicación directa con la tarjeta del robot durante las pruebas en hardware real.

A nivel organizativo, el entorno de desarrollo se estructuró mediante un proyecto modular, en el cual cada componente cumple un rol específico dentro del flujo general de ejecución. Esta modularidad facilitó el mantenimiento del código, la depuración y la posibilidad de extender funcionalidades sin afectar el funcionamiento global.

Se trabajó utilizando un editor de desarrollo habitual como lo es Visual Studio Code, configurado con herramientas de apoyo tales como inspección de sintaxis, entornos virtuales y control de versiones. Esta configuración permitió mantener un entorno limpio y reproducible para cada etapa de prueba.

La integración de las distintas librerías se realiza exclusivamente desde el archivo principal **TODO.py** para el ejemplo del GUI, el cual actúa como punto de entrada del sistema.

Desde allí se inicializan los módulos internos, se cargan las configuraciones del robot y se establece la lógica operativa que gobierna la interacción entre el hardware, el controlador y la interfaz gráfica. Esto permite ejecutar el proyecto completo desde un único archivo, simplificando su uso en contextos educativos y experimentales. Luego para los ejemplos de visión artificial y cinemática inversa se utiliza solo 2 códigos, la librería **QArm_lib.py** y solo un código más simple que el del GUI, ya que este es el más largo debido a todas sus funcionalidades.

3.3 Integración de Python con el brazo robótico

La integración de Python con el QArm se desarrolló mediante una arquitectura modular que permite gestionar de forma clara y eficiente la comunicación entre el software y el hardware del robot. A diferencia del entorno original basado en MATLAB/Simulink, Python ofrece un ecosistema más flexible y abierto, lo que facilita la implementación de funciones personalizadas, la creación de interfaces y la expansión hacia comportamientos avanzados.

El núcleo de esta integración se construye en torno a la librería **QArm_lib.py**, encargada de establecer el enlace directo con la plataforma HIL del robot. Este módulo encapsula las funciones esenciales de comunicación —lectura de estados, envío de consignas y control del actuador del gripper— permitiendo que el resto del sistema opere sobre una capa de abstracción sencilla y consistente. La comunicación se realiza principalmente en Position Mode, que ofrece estabilidad y un comportamiento predecible para aplicaciones educativas y experimentales.

Sobre esta base se desarrolló el módulo **Qarm_controller.py**, que actúa como una capa intermedia entre la lógica del proyecto y el hardware. Este módulo incorpora validación de límites, zonas muertas, control seguro del gripper, funciones de inicialización y mecanismos de parada inmediata. Su función principal es transformar comandos de nivel alto en instrucciones compatibles con QArm_lib.py, asegurando un funcionamiento seguro y evitando la necesidad de que la interfaz gráfica interactúe directamente con el hardware.

La integración con Python también permite conectar el brazo con otros módulos del proyecto, como la interfaz gráfica creada con tkinter o los componentes de visión por computadora implementados con OpenCV. Esta modularidad garantiza que las distintas partes del sistema se comuniquen de manera controlada y que nuevas funciones puedan añadirse sin modificar la estructura principal. De esta forma, la arquitectura resulta escalable y adaptable, tanto para extender el proyecto como para utilizarlo con otros manipuladores robóticos compatibles. En conjunto, la integración realizada demuestra que Python es una alternativa viable y robusta para el control del QArm, permitiendo superar las limitaciones del entorno original y habilitando un desarrollo más flexible, personalizable y orientado a la experimentación.

3.4 Diseño de scripts para el control básico del QArm

El control de movimientos del ejemplo GUI con QArm se estructuró a partir de una arquitectura modular en Python, cuyo funcionamiento se organiza alrededor del archivo principal **TODO.py**. Este script actúa como punto de entrada del sistema y es el encargado de inicializar

todas las librerías necesarias, gestionar el ciclo de lectura–escritura y coordinar la interacción entre la interfaz gráfica, el controlador lógico y el hardware del robot.

En esta arquitectura, la librería **Qarm_controller.py** cumple el rol de módulo central para la lógica de control. Este archivo integra internamente la librería **QArm_lib.py**, ya que requiere acceder a las funciones de lectura de estado, envío de consignas y administración del modo de operación del robot. De este modo, **Qarm_controller.py** funciona como una capa intermedia que encapsula la interacción con el hardware (o la simulación) y expone funciones de mayor nivel para ser utilizadas por la interfaz gráfica o por scripts externos.

La librería **QArm_lib.py** mantiene las funciones esenciales para operar el QArm, especialmente en modo Position Mode, que es el modo recomendado para aplicaciones educativas o experimentales orientadas al control directo de articulaciones. Entre los métodos principales se encuentran:

Método	Descripción	Aplicación
read_std()	Permite obtener en un único ciclo las posiciones articulares, corrientes, velocidades estimadas, temperatura y la señal PWM aplicada a cada articulación.	Se utiliza para implementar zonas muertas, verificaciones de límites, retroalimentación en tiempo real y análisis del desempeño del sistema.
write_position(phiCMD, gprCMD)	Envía consignas articulares en modo posición, permitiendo realizar movimientos precisos sin modificar configuraciones internas del sistema HIL.	Fue el método empleado en la mayor parte del control por consignas de posición del brazo robótico.
read_write_std(phiCMD, gprCMD, baseLED)	Realiza operaciones de lectura y escritura en un mismo ciclo de ejecución, incluyendo el control del LED de la base.	Se emplea en bucles de control de alto rendimiento, donde es necesario enviar comandos y recibir retroalimentación en la misma iteración.

Tabla 2. Métodos principales (QArm_lib.py)

Sobre estas funciones se construye **Qarm_controller.py**, que incorpora lógica adicional de seguridad, validación de límites, aplicación de zonas muertas, control del gripper y funciones auxiliares que solo son necesarias para la implementación de la GUI. Además, este módulo

gestiona el estado interno del sistema y ofrece funciones de más alto nivel, lo que permite separar la manipulación directa del hardware de la lógica de control y de la interfaz.

Por su parte, la librería **Graphic_interface.py** implementa la interfaz gráfica del proyecto, actuando como la capa de presentación (frontend). Allí se definen las ventanas, controles, botones, entradas, bindings y los elementos visuales que permiten al usuario interactuar con el robot sin necesidad de manipular directamente el código. La GUI se comunica únicamente con **Qarm_controller.py**, manteniendo una estructura limpia y modular.

Finalmente, **TODO.py** se encarga de orquestar todos estos componentes. En él se inicializan el hardware o la simulación, el controlador, la interfaz gráfica y el bucle principal del sistema. Gracias a esta organización, cualquier modificación realizada en las librerías internas se integra automáticamente en la ejecución global sin necesidad de alterar el archivo principal. Por razones de seguridad operativa, el sistema incorpora además el método **stop_immediate()**, que fuerza la detención del robot enviando repetidamente la posición HOME para cancelar perfiles activos y evitar comportamientos inesperados durante las pruebas iniciales y finales. Esta estructura modular garantiza escalabilidad, mantenimiento sencillo y un comportamiento consistente tanto en modo real como simulado.

3.5 Validación y pruebas experimentales

El proceso de validación del sistema se llevó a cabo de manera progresiva, siguiendo una estrategia que permitió evaluar cada componente del proyecto en condiciones controladas antes de avanzar hacia pruebas más complejas. Para ello, en los tres ejemplos desarrollados se

incorporó una interacción inicial que permite seleccionar el modo de operación (virtual o real) facilitando tanto el trabajo remoto como las pruebas presenciales. Esta funcionalidad resultó fundamental, ya que permitió desarrollar y depurar gran parte del código desde el hogar utilizando el entorno simulado de QLabs, garantizando la continuidad del proyecto incluso sin acceso directo al hardware.

Las primeras pruebas se centraron en verificar la correcta inicialización del sistema, la comunicación con la tarjeta HIL y la lectura estable de datos por parte del QArm. En esta etapa se validaron las funciones básicas de la API, como la obtención de posiciones articulares y el envío de consignas simples mediante movimientos individuales de cada eje. Estas validaciones iniciales confirmaron la operación correcta tanto en simulación como en el hardware real, permitiendo asegurar que la estructura base del módulo **QArm_lib.py** funcionaba correctamente. Este proceso resultó especialmente desafiante debido a la escasez de documentación oficial disponible sobre la comunicación directa entre Python y el hardware del QArm, lo que implicó un enfoque intensivo de prueba y error hasta lograr resultados estables.

Una vez superada esta instancia, se avanzó hacia pruebas más estructuradas. En cada ejemplo sucesivo se incorporaron funcionalidades adicionales, lo que permitió evaluar la evolución del sistema. Así, se pasó de scripts orientados únicamente al control elemental de articulaciones a implementaciones más completas que integraban zonas muertas, validación de límites, lógica de seguridad y optimización del envío de consignas. Este enfoque incremental garantizó la detección temprana de errores y posibilitó corregirlos antes de complejizar el sistema.

En la etapa siguiente, las pruebas se extendieron al uso del controlador intermedio (**Qarm_controller.py**) y la interfaz gráfica (**Graphic_interface.py**). Estas evaluaciones permitieron verificar la interacción entre usuario, lógica de control y hardware, asegurando un funcionamiento fluido de los controles, botones, deslizadores y mecanismos de parada segura. La integración de la selección de modo (real/virtual) dentro de la GUI también fue validada, confirmando su estabilidad y precisión en ambos entornos.

Finalmente, se llevaron a cabo pruebas prolongadas y secuenciales para evaluar la robustez del sistema, incluyendo ciclos continuos de movimiento, operación simultánea de varios ejes y análisis de corrientes motoras. En esta etapa, el método **stop_immediate()** fue probado repetidamente para garantizar su eficacia como mecanismo de seguridad ante movimientos inesperados o errores en el flujo de comandos.

El enfoque progresivo de validación —comenzando desde lo más simple, avanzando hacia estructuras más completas y utilizando tanto simulación como hardware real— permitió verificar que el sistema desarrollado es estable, escalable y adecuado para el control del QArm. Esta secuencia de pruebas constituye evidencia sólida de que la migración hacia Python no solo es viable, sino ventajosa frente a entornos propietarios más restrictivos.

3.6 Criterios de evaluación del desempeño

La evaluación del desempeño del sistema de control desarrollado para el Q-Arm se llevó a cabo mediante un conjunto de criterios cuantitativos y cualitativos que permiten determinar su eficacia, robustez y utilidad práctica en el entorno académico. Estos criterios se seleccionaron en

función de los objetivos planteados en el proyecto y de las características propias de un brazo robótico didáctico controlado mediante Python.

La precisión posicional del sistema se evaluó analizando la capacidad del efector final para alcanzar posiciones objetivo dentro del espacio de trabajo del robot. Para ello, se midió el error absoluto promedio entre la posición deseada y la posición efectivamente alcanzada, considerando tanto ejecuciones repetitivas de un mismo movimiento como desplazamientos aislados hacia distintos puntos. Este análisis permitió cuantificar el nivel de exactitud alcanzado por el sistema de control implementado.

La repetibilidad se analizó a partir de la variación observada entre ejecuciones consecutivas de una misma consigna de movimiento. Este criterio resulta clave para evaluar la consistencia del algoritmo de control y la estabilidad de la comunicación entre el software y el hardware del brazo robótico. Una baja dispersión entre resultados sucesivos indica un comportamiento predecible y confiable del sistema.

El tiempo de respuesta se determinó midiendo el intervalo transcurrido entre el envío de un comando desde el entorno Python y el inicio de la ejecución efectiva en los actuadores del robot. Este parámetro permitió evaluar la eficiencia del modelo de comunicación adoptado e identificar posibles retardos asociados al procesamiento del software, la transmisión de datos o la interfaz con el hardware.

La suavidad de movimiento se evaluó mediante un análisis combinado, tanto cualitativo como cuantitativo, de la continuidad en el desplazamiento de las articulaciones. Se observaron posibles aceleraciones abruptas, vibraciones o discontinuidades durante los movimientos, ya que este tipo de comportamientos puede afectar la calidad del control y, en el largo plazo, comprometer la integridad mecánica del sistema.

La robustez ante fallos se examinó considerando el comportamiento del sistema frente a situaciones anómalas, como desconexiones breves, pérdidas parciales de datos o errores en la comunicación. En este análisis se verificó la correcta activación de los mecanismos de seguridad implementados, tales como la detención automática del movimiento o el retorno a posiciones seguras, con el objetivo de preservar la seguridad del equipo y del entorno de trabajo.

La flexibilidad del software desarrollado se evaluó analizando su capacidad para incorporar nuevas rutinas de movimiento o comportamientos personalizados sin necesidad de realizar modificaciones estructurales en el código. Este aspecto es fundamental para garantizar la escalabilidad del sistema y su reutilización en futuros desarrollos o aplicaciones educativas.

La experiencia del usuario se evaluó considerando la claridad de la interfaz gráfica, la facilidad de operación del sistema, la organización de los parámetros configurables y la curva de aprendizaje asociada. Este análisis se orientó especialmente a usuarios sin experiencia previa en control robótico, con el fin de determinar si la herramienta resulta accesible y adecuada para su uso en contextos educativos.

La confiabilidad de las mediciones se garantizó mediante la repetición sistemática de las pruebas experimentales, observando la consistencia de los resultados obtenidos ante la ejecución reiterada de las mismas consignas de control. La estabilidad del comportamiento del sistema frente a múltiples ciclos de operación permitió verificar la reproducibilidad de las respuestas del robot bajo condiciones similares.

La validez del estudio se sustenta en la coherencia entre los objetivos planteados, los criterios de evaluación seleccionados y el contexto académico en el cual se aplica el sistema desarrollado. Las métricas empleadas (precisión, repetibilidad, tiempo de respuesta, suavidad de movimiento, robustez y experiencia del usuario) se encuentran directamente vinculadas con los requerimientos de uso educativo y experimental del QArm, asegurando que los resultados obtenidos reflejen adecuadamente el desempeño real del sistema.

3.7 Uso de Inteligencia Artificial como herramienta de apoyo en el desarrollo

A lo largo del desarrollo del presente proyecto se utilizaron herramientas basadas en Inteligencia Artificial como recurso de apoyo en distintas etapas del proceso de diseño, implementación y documentación del sistema de control del Q-Arm. En particular, se empleó el modelo conversacional ChatGPT (versión GPT-5.2) como asistente técnico y conceptual. Su utilización tuvo un carácter complementario y no sustituyó en ningún momento el trabajo de análisis, programación, validación experimental ni la toma de decisiones propias del autor.

En una primera etapa, la herramienta de IA se utilizó como apoyo para la exploración de alternativas de diseño y para la consulta de conceptos teóricos vinculados con cinemática de

manipuladores, arquitecturas de control, comunicación hardware–software y buenas prácticas de programación en Python. Este uso permitió contrastar enfoques posibles, aclarar conceptos y reforzar el sustento teórico de las decisiones adoptadas en fases tempranas del proyecto, reduciendo la incertidumbre asociada al diseño inicial.

Posteriormente, la IA fue empleada como asistencia durante tareas de depuración y reorganización del código, especialmente en el proceso de integración entre el módulo de control del brazo robótico, la interfaz gráfica y el sistema de comunicación con el Q-Arm. En este contexto, la herramienta contribuyó a la detección de errores lógicos, a la mejora de la estructura modular y a la optimización de la legibilidad del código, siempre bajo supervisión y validación manual, asegurando que la funcionalidad final respondiera a los objetivos planteados.

Además, la herramienta actuó como un entorno de consulta permanente durante la resolución de problemas puntuales surgidos en la implementación, proporcionando ejemplos, aclaraciones conceptuales y sugerencias técnicas. Su uso contribuyó a un proceso de desarrollo más ordenado y eficiente, manteniendo en todo momento la responsabilidad técnica, el análisis crítico y la validación experimental bajo control directo del autor.

CAPÍTULO IV

CAPÍTULO IV: Desarrollo de las Implementaciones Prácticas

4.1 Interfaz de control y ejecución de trayectorias

La implementación de una interfaz gráfica para la programación y ejecución de trayectorias constituye uno de los desarrollos centrales del proyecto, ya que esto mismo simulará una interfaz industrial basada en los principios de los software más utilizados. Su propósito fue proporcionar una herramienta intuitiva que permita al usuario construir secuencias de consigna angular, almacenarlas, editarlas y ejecutarlas sobre el QArm en modo de simulación (QLabs) o en hardware real, todo ello sin necesidad de modificar código fuente.

4.1.1 Objetivos del desarrollo de la interfaz

- Facilitar la generación y edición de puntos articulares (consignas) mediante controles manuales (sliders y botones).

- Proveer mecanismos seguros de ejecución (ciclos, tiempo entre puntos) y parada de emergencia para minimizar riesgos durante las pruebas.

- Permitir persistencia de rutas en disco en formato JSON para reutilización y documentación de experimentos.

- Integrar de forma transparente la selección de modo (simulación/físico) para poder depurar en QLabs y, posteriormente, validar en hardware real.

- Mantener separación de responsabilidades: la GUI actúa como frontend y delega la lógica de control y seguridad en el controlador intermedio.

4.1.2 Estructura del sistema

La interfaz forma parte de una arquitectura modular cuya orquestación se realiza desde TODO.py. La estructura funcional relevante para la GUI es:

Archivo / Módulo	**Función principal**	**Descripción**
TODO.py (main)	Inicialización del sistema	Actúa como punto de entrada del proyecto. Presenta una ventana inicial para seleccionar el modo de operación (simulación o hardware físico) e instancia la clase “QArmWrapper”, desde donde se coordina el funcionamiento general del sistema.
QArmWrapper (Qarm_controller.py)	Capa de abstracción y normalización	Implementa un wrapper que normaliza las consignas de movimiento (conversión entre radianes y grados), aplica límites articulares de seguridad y centraliza las llamadas a la librería de bajo nivel “QArm_lib” para la lectura y escritura de estados del robot.

QArm_lib.py	Comunicación de bajo nivel	Gestiona la comunicación directa con el entorno HIL/QLabs y el hardware del QArm. Se encarga del manejo de buffers, la transmisión de comandos y la ejecución de perfiles de movimiento, actuando como interfaz directa con el dispositivo.
Graphic_interface.py (QArmGUI)	Interfaz gráfica de usuario	Contiene los componentes visuales y la lógica de la interfaz gráfica. Permite al usuario enviar consignas al sistema a través de "QArmWrapper" y gestiona el ciclo de ejecución de trayectorias y rutinas definidas.

Tabla 3. Estructura funcional GUI

Esta separación preserva la seguridad (la GUI no escribe directamente en la HIL) y facilita el mantenimiento y la reutilización en otros proyectos.

4.1.3 Interfaz gráfica (GUI)

La Figura 3 muestra la interfaz gráfica desarrollada para el control del QArm, correspondiente al módulo QArmGUI implementado, utilizando la biblioteca tkinter y diseñada para operar en una pantalla fija con una resolución de 980×760 px. La disposición de los elementos busca priorizar la claridad, la facilidad de uso y la seguridad durante la interacción con el robot.

Como se muestra en la Figura 3, el sector izquierdo corresponde al Control Manual, desde donde el usuario puede accionar directamente cada una de las articulaciones del brazo. Las juntas J1 a J4 se controlan mediante deslizadores que permiten ajustar la posición angular en grados, complementados con botones de incremento y decremento fino (±1°) para realizar ajustes precisos. Junto a cada control se visualiza el valor angular actual, lo que brinda una referencia inmediata del estado del sistema.

En este mismo bloque se incluye el control del gripper, cuyo deslizador permite regular la apertura de la pinza dentro de un rango definido, lo cual es importante ya que depende el elemento que se quiera sujetar se deberá calibrar el un cierre personalizado del grimper, ya que hay que lograr agarre en la pieza deseada, pero no puede quedar haciendo demasiada fuerza, porque esto elevará el consumo energético umbral de seguridad del brazo y se activará la parada de emergencia. Además, el botón “Volver a HOME” permite retornar el robot a una configuración inicial segura y conocida, facilitando el reinicio de rutinas o la recuperación del sistema ante posiciones no deseadas.

El la Figura 3 también destaca la presencia de un botón de Parada de Emergencia, claramente identificado y accesible. Este control permite detener de forma inmediata cualquier movimiento del robot y lo envía a la posición segura “home”, cabe aclarar que el mecanismo de seguridad principal ya lo tiene integrado el robot con respecto al consumo energético, por lo que esta parada de emergencia es un extra manual. Los botones “Reiniciar Robot” y “Cerrar conexión y salir” complementan este esquema, permitiendo restablecer el sistema tras una

detención o finalizar correctamente la sesión de trabajo, asegurando una desconexión ordenada del hardware o del entorno de simulación.

El sector derecho de la Figura 3 corresponde a la Gestión de Ruta, orientada a la programación y ejecución de trayectorias compuestas por múltiples puntos. Desde este panel, el usuario puede guardar la posición actual como un punto de la ruta, editar puntos existentes, eliminarlos o crear nuevas secuencias. Asimismo, se incluyen opciones para guardar rutas en archivos y cargarlas posteriormente, lo que facilita la reutilización de trayectorias entre distintas sesiones de trabajo.

La ejecución de rutas se controla mediante la configuración del tiempo entre puntos y la cantidad de ciclos, permitiendo evaluar tanto movimientos aislados como secuencias repetitivas. La lista de puntos guardados muestra cada posición junto con su retardo asociado, brindando una visualización clara del orden y contenido de la trayectoria antes de su ejecución.

Desde el punto de vista funcional, la GUI actual envía consignas por punto, sin realizar interpolación interna. Cada punto de la trayectoria se transmite al robot mediante la función write_position, y posteriormente la interfaz espera el tiempo de demora configurado antes de proceder al siguiente punto. La suavidad del movimiento entre consignas queda delegada a los perfiles definidos en la capa HIL, implementada en QArm_lib, donde se establecen las velocidades y aceleraciones de las articulaciones. Este enfoque simplifica la lógica de la interfaz gráfica y separa claramente las responsabilidades entre la capa de control de alto nivel y la gestión de perfiles de movimiento del hardware.

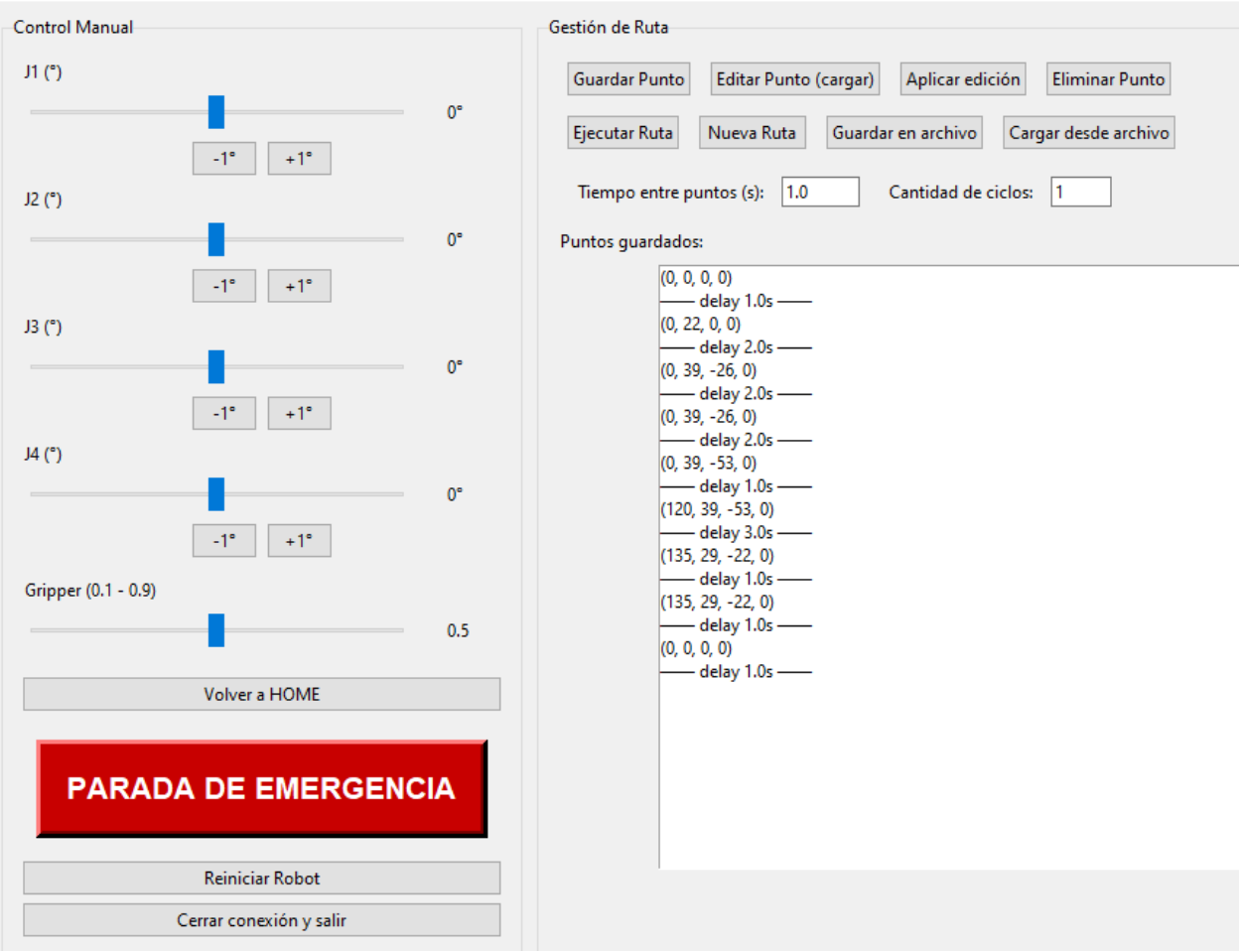

Figura 3. Recorte interfaz GUI - Fuente: Elaboración propia

4.1.4 Sistema de almacenamiento de trayectorias

Las rutas se guardan y recuperan en formato JSON con la estructura mínima necesaria para su reproducción:

[

{ "pos": [J1_deg, J2_deg, J3_deg, J4_deg], "gripper": g_value, "tiempo": delay_s },

...]

- **pos:** Lista de cuatro valores en grados (coherente con la interfaz del usuario).
- **gripper:** Valor entre 0.1 y 0.9.
- **tiempo:** Tiempo en segundos que la GUI espera después de enviar la consigna antes de continuar.

El guardado y la carga se efectúan mediante diálogos estándar de tkinter (filedialog.asksaveasfilename y askopenfilename), lo que facilita distribuir y documentar rutinas experimentales.

4.1.5 Resultados del sistema

La interfaz desarrollada permitió cumplir de manera satisfactoria los objetivos planteados para el proyecto, tanto en términos de funcionalidad como de usabilidad. La posibilidad de ejecutar el sistema en modo simulación mediante QLabs resultó especialmente valiosa durante las primeras etapas, ya que permitió desarrollar y depurar la mayor parte de la lógica de la GUI desde un entorno doméstico. Esto facilitó la detección temprana de errores lógicos, aceleró los ciclos de prueba y permitió iterar sobre el diseño de la interfaz sin necesidad de acceso permanente al hardware físico.

Una vez validado el funcionamiento en simulación, las mismas rutas y mecanismos de control pudieron ejecutarse directamente sobre el QArm real sin requerir modificaciones en la interfaz gráfica. La selección del modo de operación se realiza al inicio de la ejecución, desde el archivo principal, lo que permitió una transición transparente entre simulación y hardware físico. Este

comportamiento evidencia una correcta separación entre la lógica de la GUI y la capa de comunicación con el robot, favoreciendo la portabilidad y la reutilización del sistema.

Desde el punto de vista de la seguridad, la integración del botón de parada de emergencia dentro de la interfaz, en conjunto con la función “stop_immediate()” provista por QArm_lib, demostró ser efectiva para detener movimientos en curso y minimizar riesgos durante las pruebas. Esta función fuerza la cancelación del perfil activo mediante el envío reiterado de la posición HOME, evitando desplazamientos bruscos o comportamientos no deseados. Es importante destacar que el QArm cuenta además con un sistema de seguridad nativo basado en protección por sobrecorriente, que desenergiza automáticamente los motores ante consumos excesivos. En este sentido, la parada de emergencia implementada a nivel de software no reemplaza al mecanismo original, sino que actúa como una capa adicional orientada a prevenir incidentes durante el desarrollo y la experimentación.

En cuanto a la fiabilidad, las rutas almacenadas en formato JSON se reprodujeron de manera consistente entre ejecuciones. La estrategia de envío de consignas por puntos, combinada con los perfiles de movimiento definidos en la capa HIL, resultó adecuada para la facil comprensión y su posterior edición..

No obstante, durante las pruebas también se identificaron algunas limitaciones inherentes al enfoque adoptado. La ejecución por puntos responde a una lógica similar a un movimiento tipo *moveJ*, basada en consignas discretas con tiempos de espera intermedios. Si los puntos consecutivos se encuentran muy separados en el espacio articular, pueden producirse cambios rápidos en la trayectoria. Este efecto puede mitigarse ajustando los perfiles de velocidad y

aceleración en QArm_lib y aumentando los tiempos de espera entre puntos, con una correcta configuración de esta espera entre puntos se logra mitigar casi por completo el inconveniente.

Asimismo, la GUI no implementa interpolación posicional propia, por lo que cualquier mejora en la suavidad del movimiento debería abordarse incorporando interpolación en el controlador intermedio o directamente en la capa de la interfaz gráfica. Finalmente, la precisión alcanzable depende en gran medida de las limitaciones mecánicas del QArm y de su controlador interno. La interfaz se limita a enviar consignas angulares, sin implementar lazos de control adicionales a nivel de software, los cuales quedan supeditados al funcionamiento del HIL y a las características físicas del manipulador.

4.2 Implementación de cinemática inversa

En esta sección se presentan los fundamentos conceptuales de la cinemática inversa aplicada al QArm y la metodología utilizada para resolverla en Python mediante las funciones provistas por Quanser.

4.2.1 Implementación en Python

La IK del QArm se implementó en Python utilizando las librerías oficiales de Quanser (pal y hal), que incluyen funciones optimizadas para el cálculo de cinemática directa e inversa en este manipulador. El objetivo fue desarrollar una interfaz gráfica que permitiera controlar de forma directa la posición cartesiana del efector final (X, Y y Z), su orientación (γ) y el estado del gripper, de manera segura y con respuesta en tiempo real.

El programa comienza con una ventana de selección del modo de operación, donde el usuario puede elegir entre simulación en QLabs o ejecución sobre el hardware real. Este esquema permitió desarrollar y probar el sistema en simulación y luego validar el mismo código sobre el robot físico sin realizar modificaciones. A continuación, se inicializan los objetos principales: una instancia de QArm, utilizada para el envío de consignas, y una instancia de "QArmUtilities", que concentra las rutinas de cinemática directa e inversa.

El cálculo central se realiza a partir de los valores ingresados en la GUI, los cuales actualizan de manera continua la posición deseada del efector final. El sistema lee el estado actual de las articulaciones y utiliza la función de cinemática inversa para obtener la configuración articular adecuada. La solución seleccionada se valida mediante un cálculo de cinemática directa y luego se envía al robot a través de la función estándar de lectura y escritura, incorporando una breve espera para estabilizar la actualización.

La interfaz gráfica se apoya en sliders y controles de ajuste fino que permiten modificar las consignas de forma intuitiva. Cada cambio en los controles dispara automáticamente la actualización del sistema, generando un movimiento fluido y eficiente del brazo.

4.2.2 Integración con el robot real y simulación

La misma base de código funciona tanto en simulación como en hardware real.

Esto fue posible porque el sistema de Quanser utiliza la misma API en ambos modos, y porque se evitó manipular comandos de bajo nivel.

En la práctica, esto permitió desarrollar y depurar desde casa en QLabs, además aplicar sin cambios sobre el brazo físico y validar la precisión del IK contra movimientos reales

La única diferencia operativa es el parámetro:

QArm(hardware=0) → Simulación

QArm(hardware=1) → Físico

4.2.3 Consideraciones de seguridad

Aunque esta implementación utiliza las funciones oficiales de Quanser, se tomaron precauciones adicionales:

- Se definieron límites espaciales razonables para los sliders, evitando posiciones inalcanzables para el robot.
- El robot QArm posee un sistema de protección por sobrecorriente que lo desenergiza automáticamente ante esfuerzos excesivos.
- Aun así, la interfaz integra un botón de apagado que ejecuta: **myArm.terminate()** para asegurar un cierre seguro del hardware.

4.3 Visión artificial

4.3.1 Uso de cámara + OpenCV + Mediapipe

Para implementar el control visual del QArm se utilizó una cámara web estándar en conjunto con las bibliotecas OpenCV y Mediapipe. OpenCV permite gestionar el acceso al dispositivo de captura, realizar operaciones básicas de preprocesamiento y mostrar las imágenes procesadas en tiempo real. Sobre estas imágenes, Mediapipe proporcionó un marco robusto para la detección y el seguimiento de la mano del usuario, aprovechando modelos optimizados para ejecución eficiente en CPU.

El flujo de trabajo comienza con la captura de cada fotograma a través de OpenCV, que se convierte al espacio de color requerido por Mediapipe para su análisis. Una vez procesada la imagen, el módulo Hands de Mediapipe detecta los puntos clave de la mano (landmarks), calculando su posición relativa dentro del fotograma. Estas coordenadas son esenciales para determinar la ubicación y orientación de la mano, permitiendo traducir gestos en comandos para el movimiento del robot.

El uso combinado de OpenCV y Mediapipe ofrece un procesamiento fluido, con baja latencia y suficiente precisión para aplicaciones educativas y experimentales. Además, la integración de ambas bibliotecas facilita la rápida implementación de interfaces visuales interactivas, permitiendo al usuario tener retroalimentación inmediata durante el control del manipulador.

4.3.2 Integración con el brazo robótico

La integración entre el sistema de visión artificial y el QArm se realizó estableciendo un vínculo directo entre las coordenadas detectadas por Mediapipe y los comandos de movimiento enviados al robot. Una vez obtenidos los puntos clave de la mano en el fotograma, se seleccionó un landmark representativo —en este caso, la posición del puño cerrado— como referencia principal para el control. Dichas coordenadas se normalizan y se traducen a valores proporcionales al rango operativo de las articulaciones del QArm.

La comunicación con el brazo se llevó a cabo mediante la biblioteca QArm_lib, que permite enviar instrucciones de posición angular o velocidades articulares. El sistema procesa

cada fotograma en tiempo real, calcula la posición relativa de la mano y determina el movimiento correspondiente del robot. Para evitar oscilaciones o movimientos involuntarios se implementaron zonas muertas y filtros básicos, reduciendo la sensibilidad en áreas críticas y mejorando la estabilidad del control.

El resultado es un esquema de teleoperación visual en el cual los movimientos del usuario se reflejan de forma inmediata en el manipulador. Aunque no se trata de un control de precisión (debido a limitaciones mecánicas, velocidad de procesamiento y variabilidad en la detección) el sistema demuestra ser efectivo para tareas experimentales, demostraciones educativas y validación conceptual de interacción hombre-robot mediante visión artificial.

4.3.3 Lógica de seguimiento del puño

Para implementar un control intuitivo basado en visión artificial, se definió una lógica específica para el seguimiento del puño del usuario. El sistema utiliza Mediapipe Hands para detectar en cada fotograma la presencia de la mano y sus puntos clave (landmarks). Entre ellos, se emplea el punto asociado al wrist o al centro geométrico del puño como referencia primaria para determinar la posición del brazo robótico.

Una vez identificada la mano, el algoritmo evalúa dos aspectos fundamentales:

- **Posición del puño:**

Se toman las coordenadas normalizadas entregadas por Mediapipe (en un rango de 0 a 1 para X e Y). Estas coordenadas se mapean al rango operativo del manipulador, permitiendo que el desplazamiento del puño en la imagen se traduzca en movimientos articulares. Para suavizar la

respuesta se aplican filtros básicos y una zona muerta, evitando que pequeñas variaciones por ruido provoquen correcciones innecesarias del robot.

- **Estado de la mano (abierta o cerrada):**

Para determinar si el usuario desea accionar la garra del QArm, se calcula la distancia relativa entre los dedos y la palma. Si la distancia promedio es baja, se interpreta como puño cerrado; si es alta, como mano abierta. Esta lógica permite que el robot cierre o abra su garra sin necesidad de botones adicionales.

Combinando ambas señales, el sistema obtiene un control simultáneo de posición y agarre mediante gestos simples. La lógica resultante permite una teleoperación fluida y natural: mover el puño desplaza el extremo del brazo robótico, mientras que cerrar o abrir la mano genera la acción correspondiente en la garra. Si bien el método depende de la calidad de detección y presenta limitaciones inherentes al uso de cámaras RGB, demuestra ser eficaz para propósitos demostrativos, académicos y exploratorios.

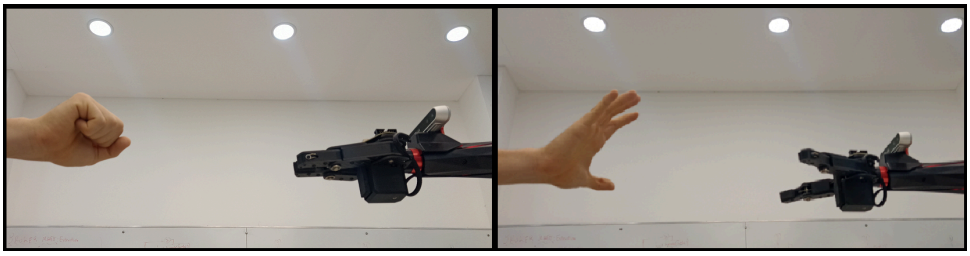

Figura 4. Apertura/Cierre Grimpper - Fuente: Elaboración propia

CAPÍTULO V

CAPÍTULO V: Resultados

5.1 Desempeño general de Python controlando un brazo robótico

El uso de Python como entorno principal para controlar el QArm demostró ser una alternativa flexible, accesible y suficientemente robusta para el desarrollo de aplicaciones de manipulación robótica a nivel académico. Su capacidad para integrar librerías externas, comunicarse con el hardware y ejecutar algoritmos de control en tiempo real permitió construir un sistema funcional sin recurrir a entornos más complejos o restrictivos.

En términos de desempeño, Python mostró un comportamiento estable durante la mayor parte de las pruebas. Las rutinas de comunicación con el brazo (gestionadas internamente por las librerías oficiales de Quanser) respondieron con baja latencia y mantuvieron una sincronización adecuada entre los comandos enviados y la retroalimentación del robot. Esto permitió ejecutar movimientos continuos y controlar tanto trayectorias como posiciones cartesianas desde una interfaz gráfica sin interrupciones significativas.

Si bien Python no es un lenguaje orientado a aplicaciones de control crítico en tiempo real, su rendimiento resultó suficiente para las exigencias del QArm, gracias a que el procesamiento más delicado (como la cinemática, la lectura de sensores y la protección interna) se encuentra encapsulado dentro del propio firmware y las librerías de Quanser. Esto reduce la carga computacional del lado de Python y evita que pequeñas demoras del intérprete afecten gravemente la ejecución.

Por otra parte, Python aportó una ventaja fundamental: su ecosistema. La integración natural con bibliotecas como Tkinter para interfaces, NumPy para cálculos numéricos y OpenCV/Mediapipe para visión artificial permitió construir un entorno completo de control robótico sin necesidad de cambiar de plataforma ni fragmentar el desarrollo.

En conjunto, el desempeño general del sistema confirma que Python es una herramienta adecuada para el control de robots educativos como el QArm. Ofrece un balance sólido entre facilidad de desarrollo, capacidad de integración y estabilidad operacional, lo cual lo convierte en una opción recomendable tanto para docencia como para investigación introductoria en robótica.

5.2 Evaluación del rendimiento de los algoritmos implementados

La evaluación del rendimiento de los algoritmos desarrollados durante el proyecto permitió identificar su grado de eficiencia, estabilidad y adecuación al trabajo con un manipulador físico real. Cada uno de los componentes (cinemática inversa, control de movimiento, filtrado de señales de visión artificial e integración general) fue analizado considerando su comportamiento tanto en simulación como en el brazo QArm real.

En términos generales, los algoritmos presentaron un desempeño sólido dentro de las limitaciones mecánicas y sensoriales del robot. El cálculo de cinemática inversa, soportado por las funciones internas de Quanser, demostró buena estabilidad y tiempos de respuesta reducidos, permitiendo actualizaciones continuas desde la interfaz gráfica sin pérdidas de sincronización. No obstante, la precisión final obtenida en el efector depende fuertemente de la capacidad del

robot para ejecutar físicamente los valores articulares propuestos por el algoritmo, presentando pequeñas desviaciones propias de la naturaleza del QArm.

Los algoritmos de control de movimiento, incluyendo la normalización del gripper, el recorte por límites articulares y la lógica de envío de consignas en tiempo real, mostraron un funcionamiento estable. El sistema respondió sin bloqueos ni saturaciones apreciables incluso bajo variaciones rápidas en los comandos de entrada. El diseño modular del código contribuyó a reducir errores, mejorar la legibilidad y facilitar la depuración.

En cuanto a visión artificial, los algoritmos basados en OpenCV y Mediapipe lograron una detección de mano consistente y con baja latencia, adecuada para tareas de seguimiento o control gestual básico. El filtrado implementado para reducir ruido en el cálculo del punto de control (valor promedio y zona muerta programada) contribuyó significativamente a la estabilidad del movimiento. Sin embargo, su rendimiento depende del entorno de iluminación y de la calidad de la cámara utilizada.

Finalmente, la integración de todos los algoritmos en un sistema único confirmó que la arquitectura general es eficiente y escalable. La combinación de cálculos numéricos, control de hardware, GUI en tiempo real y visión artificial se ejecutó sin conflictos mayores, mostrando que el flujo de datos entre módulos es consistente y entrega una experiencia de uso continua.

En conjunto, los algoritmos implementados alcanzaron un nivel de rendimiento adecuado para los objetivos del proyecto. Si bien ciertos aspectos podrían optimizarse (principalmente la

precisión del movimiento físico y la estabilidad del seguimiento visual en entornos complejos) los resultados obtenidos reflejan una solución funcional y confiable para un brazo robótico educativo.

5.3 Estabilidad del sistema durante la ejecución

La estabilidad del sistema es un factor crítico cuando se integra software de control, algoritmos numéricos, interfaces gráficas y hardware físico. En este proyecto, la estabilidad fue evaluada tanto desde la perspectiva del comportamiento del robot como desde la robustez del software frente a variaciones en la entrada, cambios de modo (simulación/físico) y posibles fallos durante la operación.

En líneas generales, el sistema demostró un funcionamiento estable incluso bajo condiciones dinámicas, como desplazamientos continuos del efector, cambios bruscos de consignas o movimientos derivados del seguimiento por visión artificial. La arquitectura modular desarrollada contribuyó de forma significativa a esta estabilidad, permitiendo aislar componentes críticos (como el controlador del robot, los algoritmos de cinemática o el procesamiento de vídeo) y evitar que un error local afectará el funcionamiento completo.

Desde el punto de vista del hardware, la estabilidad estuvo condicionada por las limitaciones del QArm. Sus motores presentan una respuesta adecuada para movimientos suaves, pero pueden perder precisión cuando se realizan cambios abruptos o cuando se trabaja cerca de los límites articulares. Aun así, los mecanismos de protección del propio robot (incluyendo la

desenergización por sobrecorriente) evitaron situaciones de riesgo y complementaron el control de software.

En cuanto al software, no se observaron bloqueos críticos durante las pruebas. El bucle principal de actualización de la GUI mantuvo una frecuencia constante, permitiendo una respuesta fluida sin afectar el rendimiento del procesamiento de visión ni el envío de consignas. La función de emergencia, implementada como un mecanismo de detención inmediata, respondió correctamente tanto en simulación como en hardware, garantizando un cierre seguro en caso de error o situación inesperada.

La estabilidad del sistema también se vio favorecida por el uso de retardos mínimos y controlados (por ejemplo, delay de ~0.05 s en movimientos IK), evitando saturaciones en la comunicación con el robot. Esto permitió que los comandos se enviarán de manera ordenada y a un ritmo compatible con las capacidades del QArm.

5.4 Comparación entre simulación y hardware físico

La implementación del sistema se llevó a cabo de manera paralela en dos entornos: el simulador oficial del Q-Arm y el hardware físico disponible en el laboratorio. La comparación entre ambos escenarios resulta fundamental para evaluar el comportamiento real del sistema, identificar limitaciones propias del entorno virtual y validar la transferencia de los algoritmos desarrollados en Python hacia el robot físico.

En el entorno de simulación, las pruebas iniciales permitieron verificar el correcto funcionamiento de las librerías de control, la comunicación con el modelo 3D del brazo y la ejecución de movimientos básicos. Este entorno ofrecía condiciones controladas, repetibilidad y ausencia de ruido externo, facilitando la detección temprana de errores lógicos, así como el ajuste de parámetros de cinemática y temporización.

En contraste, las pruebas realizadas sobre el hardware físico introdujeron variables adicionales propias del entorno real, como tolerancias mecánicas, latencias en la comunicación serie, no linealidades en los servomotores y pequeñas diferencias entre los valores teóricos y los desplazamientos efectivos. Estas diferencias hicieron necesario ajustar los límites articulares, calibrar los tiempos de respuesta y considerar zonas muertas en los controles, particularmente al detectar gestos mediante visión por computadora.

Asimismo, se observó que ciertos movimientos que resultaban fluidos en el simulador requerían compensaciones adicionales en el hardware, especialmente en trayectorias rápidas o combinadas. A pesar de ello, la continuidad funcional entre ambos entornos fue satisfactoria: los scripts diseñados en Python pudieron ejecutarse sin modificaciones estructurales, requiriendo únicamente ajustes paramétricos menores.

5.5 Recomendaciones técnicas para mejoras futuras

A partir de los resultados obtenidos durante el desarrollo y las pruebas del sistema, se identificaron diversas oportunidades de mejora que podrían potenciar el desempeño, la estabilidad y la escalabilidad del control del Q-Arm mediante Python. Las siguientes

recomendaciones se presentan como líneas de trabajo futuras, orientadas tanto a la optimización del software como a la integración más robusta con el hardware.

En primer lugar, se sugiere avanzar hacia una arquitectura de software modular más extensa, incorporando clases y módulos específicos para el manejo de cinemática, comunicación, visión artificial y registro de datos. Esto permitiría una mayor mantenibilidad del código, facilita su reutilización en nuevos proyectos y permitiría aislar fallas o cuellos de botella de manera más eficiente.

En cuanto al procesamiento de video y la interacción basada en gestos, sería beneficioso integrar modelos de detección más modernos y optimizados, como soluciones basadas en redes neuronales ligeras (por ejemplo, variantes de MediaPipe o modelos ONNX optimizados). Dichas herramientas podrían mejorar la precisión del seguimiento de la mano y reducir el impacto del ruido ambiental, especialmente en escenarios con cambios de iluminación.

CAPÍTULO VI

CAPÍTULO VI: Conclusiones

6.1 Cumplimiento de objetivos

El proyecto logró cumplir de manera satisfactoria los objetivos generales y específicos propuestos al inicio del trabajo.

Se consiguió desarrollar una solución funcional en Python para controlar el brazo robótico QArm, integrando tanto módulos propios como las bibliotecas oficiales provistas por Quanser. Asimismo, se alcanzó el objetivo de extender las capacidades del QArm más allá de su uso tradicional en MATLAB, demostrando que Python constituye una alternativa válida, flexible y escalable para su operación.

También se cumplió el propósito de incorporar visión artificial como método intuitivo de interacción, permitiendo controlar el robot mediante gestos capturados por cámara, lo cual constituye un aporte innovador para prácticas docentes y proyectos de investigación dentro de la institución.

Por último, se desarrollaron herramientas de prueba, un sistema de seguridad complementario, una interfaz gráfica funcional y diversos casos de validación que consolidan una base técnica sólida para trabajos futuros.

El avance logrado respecto del estado inicial del QArm es significativo: pasó de ser un dispositivo cuya operación dependía casi exclusivamente de conocimientos específicos en MATLAB (y limitado en cuanto a accesibilidad y flexibilidad) a convertirse en una herramienta ampliamente aprovechable por estudiantes y docentes de la Tecnicatura y la Licenciatura.

Gracias a la migración a Python y a la incorporación de nuevos módulos de control y visión artificial, el QArm adquiere ahora un entorno de uso más intuitivo, accesible y eficiente, lo que amplía su valor didáctico y su potencial para actividades académicas e institucionales.

6.2 Conclusiones técnicas

Desde el punto de vista técnico, el proyecto permitió comprobar que Python es una herramienta válida y eficiente para el control del QArm, aunque presenta ciertas limitaciones asociadas principalmente a la latencia de comunicación y a las restricciones mecánicas propias del manipulador. A lo largo de los ensayos se observó que la cinemática inversa, si bien se implementa correctamente a nivel teórico mediante las librerías disponibles en Python, no siempre se traduce en movimientos completamente estables. Esto se debe a factores como tolerancias mecánicas, holguras estructurales, precisión limitada de los actuadores y variaciones en la respuesta entre el entorno de simulación y el hardware real.

El control basado en visión artificial mostró resultados satisfactorios, particularmente en las pruebas de seguimiento del puño, evidenciando que Python y sus librerías permiten integrar percepción y control de manera efectiva. No obstante, este enfoque se mostró sensible a condiciones externas como la iluminación, la distancia a la cámara y la presencia de oclusiones, lo que impacta directamente en la robustez del sistema. Asimismo, la comunicación entre el software y el hardware presentó una latencia moderada, que no impidió el funcionamiento general, pero sí condicionó la ejecución de movimientos rápidos o correcciones continuas de pequeña magnitud.

Por otra parte, la incorporación de una función de parada inmediata y de un sistema de seguridad complementario aportó una mejora significativa en la confiabilidad durante las pruebas con el robot físico, reduciendo riesgos y permitiendo un entorno de experimentación más controlado. En conjunto, el sistema desarrollado logró un funcionamiento estable y consistente, evidenciando una integración adecuada entre los módulos de control, percepción y hardware, y validando el uso de Python como plataforma central para este tipo de aplicaciones en entornos educativos y experimentales.

6.3 Conclusiones académicas e institucionales

El desarrollo e implementación de Python como herramienta de control para el QArm representa un avance significativo desde el punto de vista educativo e institucional. La disponibilidad de una interfaz intuitiva y de rutinas funcionales permite que los estudiantes de la tecnicatura y de la licenciatura en automatización y robótica interactúen directamente con el brazo robótico sin necesidad de conocimientos avanzados en MATLAB o programación de bajo nivel.

Esta accesibilidad facilita la comprensión de conceptos fundamentales como cinemática inversa, cinemática directa, control de actuadores y seguimiento visual, integrando teoría y práctica de manera inmediata.

Asimismo, la institución se beneficia al contar con una plataforma más versátil y segura, que puede ser utilizada en laboratorios de docencia, trabajos prácticos y proyectos de investigación, ampliando las posibilidades de experimentación y aprendizaje. La estandarización

de la API en Python también permite la reutilización del código en simulaciones, proyectos de extensión y futuras integraciones con otros sistemas educativos o de investigación.

La implementación desarrollada no solo mejora la experiencia de los estudiantes, sino que fortalece las capacidades institucionales para ofrecer formación práctica, innovadora y segura en el área de la robótica aplicada.

6.4 Recomendaciones y líneas futuras

A partir del desarrollo e implementación del control en Python para el QArm, se identificaron diversas oportunidades de mejora y expansión que podrían potenciar tanto su uso educativo como su aplicación en contextos de investigación. Una de las líneas más relevantes consiste en la integración con ROS (Robot Operating System), lo que permitiría acceder a un entorno de desarrollo más avanzado y estandarizado. La incorporación de ROS facilita la comunicación entre distintos nodos del sistema, la planificación de trayectorias complejas y la integración de sensores adicionales, alineando el trabajo con prácticas ampliamente utilizadas en la investigación robótica actual.

Otra línea de trabajo futura está relacionada con la implementación de algoritmos de inteligencia artificial y aprendizaje automático. La incorporación de estas técnicas podría habilitar el aprendizaje de trayectorias óptimas, la predicción de errores o la adaptación automática del control frente a variaciones de carga o desviaciones mecánicas, incrementando la autonomía y la precisión del manipulador. Este enfoque permitiría explorar aplicaciones más avanzadas y acercar el uso del QArm a problemáticas reales de la automatización moderna.

Asimismo, resulta pertinente profundizar en el desarrollo de entornos de simulación más avanzados. La utilización de simulaciones 3D con mayor fidelidad visual y física permitiría planificar y evaluar trayectorias complejas sin riesgo para el hardware, además de probar algoritmos de control y escenarios de interacción con objetos virtuales antes de su ejecución en el robot real. Esto contribuiría a un proceso de desarrollo más seguro y eficiente.

En el ámbito del control, se propone optimizar los algoritmos de cinemática inversa mediante el estudio de métodos numéricos más robustos o enfoques híbridos que combinen soluciones analíticas y numéricas. Estas mejoras podrían incrementar la precisión y la estabilidad del sistema, especialmente en regiones cercanas a singularidades o a los límites articulares del robot. Paralelamente, la expansión de la interfaz gráfica con nuevas funcionalidades, como la grabación y reproducción de trayectorias, el análisis de errores o la visualización de métricas de rendimiento en tiempo real, permitiría enriquecer la experiencia educativa y facilitar el análisis del comportamiento del sistema.

El desarrollo de una documentación más completa, acompañada de guías, tutoriales y ejemplos de código claros, favorece la adopción del sistema por parte de nuevos usuarios y su integración en cursos de laboratorio y proyectos académicos.

En conjunto, estas recomendaciones no solo apuntan a mejorar el desempeño técnico del QArm, sino también a ampliar su valor pedagógico e investigativo, consolidándose como una plataforma versátil, segura y alineada con las tendencias actuales en robótica educativa y aplicada.

REFERENCIAS BIBLIOGRÁFICAS

Referencias bibliográficas:

Abarzúa Poblete, A. (2023). *Desarrollo de sistema para el control de un brazo robótico impreso en 3D mediante API de programación e interfaz gráfica* (Trabajo de título). Universidad de Chile. https://repositorio.uchile.cl/xmlui/handle/2250/199877

Cantos Párraga, D. I., & Guerrero Flores, G. E. (2024). *Desarrollo de un módulo educativo para la clasificación de objetos utilizando un brazo robótico y visión artificial* (Trabajo de titulación). Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/28136

Corke, P. (2017). *Robotics, vision and control: Fundamental algorithms in MATLAB® and Python* (2nd ed.). Springer.

Craig, J. J. (2005). *Introduction to robotics: Mechanics and control* (3rd ed.). Pearson Education.

Haviland, J., & Corke, P. (2023). *Robotics toolbox for Python.* https://petercorke.github.io/robotics-toolbox-python/

Llerena Buenaño, A. S., & Salazar Villamar, M. J. (2022). A*utomatización de un sistema identificador y posicionador de objetos a través de un brazo robótico mediante visión artificial con lenguaje Python* (Trabajo de titulación). Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/22832

Macenski, S., Foote, T., Gerkey, B., Lalancette, F., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *IEEE Robotics & Automation Magazine*, 29(2), 42–55.

Mordor Intelligence. (2025). *Análisis del tamaño y la cuota de mercado de la robótica*. https://www.mordorintelligence.ar/industry-reports/robotics-market

Poncelas Bodelón, R. (2014). *Desarrollo del software de control de un brazo robotizado mediante Python* (Trabajo de Fin de Grado). Universidad de Valladolid. https://uvadoc.uva.es/handle/10324/11853

Python Software Foundation. (2024). *Python documentation.* https://docs.python.org/3/

Quanser Consulting Inc. (2025). *QUARC Python documentation*. https://docs.quanser.com/quarc/documentation/python/index.html

Quanser Inc. (2024). *Quanser*. https://www.quanser.com/

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: An open-source robot operating system. En *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop*.

Sharma, A., & Mital, M. (2021). Computer vision enabled robotics using Python & OpenCV. *International Journal of Computer Applications,* 975.

Subuyuj Juarez, R. E. (2025). *Diseño de un sistema de control y monitoreo en tiempo real para microcontroladores como herramienta didáctica y de aprendizaje en el laboratorio de robótica* (Tesis de licenciatura). Facultad de Ingeniería, Universidad de San Carlos de Guatemala. https://biblio.ingenieria.usac.edu.gt/tesis25/T16997.pdf

ANEXOS

Anexos:

Anexo A: Código

Enlace: https://github.com/Jonatan180/QArm-Python

Anexo B: Material audiovisual

Video B.1 – Programación punto a punto del QArm en Python (timelapse)

Enlace: https://www.youtube.com/watch?v=REl6ruUV72A

Video B.2 – Programación punto a punto del QArm en Python (tiempo real)

Enlace: https://www.youtube.com/watch?v=RGDOY5QbVYk

Video B.3 – Control por cinemática inversa del QArm en Python

Enlace: https://www.youtube.com/watch?v=XAjjPOm35gA

Video B.4 – Control asistido por visión artificial del QArm en Python

Enlace: https://www.youtube.com/watch?v=S1acEPbbCow