Revista esprint Investigación

https://rei.esprint.tech
Vol. 5 N° 2, julio-diciembre 2026 (387-412)

ISSN: 2960-8317

Artículo de investigación

Benchmark del rendimiento en cinemática directa del UR10: matrices vs. cuaterniones duales con Julia y RoboDK

UR10 forward kinematics performance benchmark: matrices vs. dual quaternions with Julia and RoboDK

Marlon Stalyn Cargua Cando*
Universidad Nacional de Chimborazo
Riobamba – Ecuador
marlon.cargua@unach.edu.ec
https://orcid.org/0000-0002-1255-4858

Guillermo Edvin Machado Sotomayor
Universidad Nacional de Chimborazo
Riobamba – Ecuador
gmachado@unach.edu.ec
https://orcid.org/0000-0001-5226-468X

***Correspondencia:**
marlon.cargua@unach.edu.ec

Cómo citar este artículo:
Cargua, M., & Machado, G. (2026). Benchmark del rendimiento en cinemática directa del UR10: matrices vs. cuaterniones duales con Julia y RoboDK. *Esprint Investigación, 5*(2), 387-412. https://doi.org/10.61347/ei.v5i2.375

Recibido: *13 de agosto de 2026*
Aceptado: *14 de septiembre de 2026*
Publicado: *21 de septiembre de 2026*

Copyright: Derechos de autor 2026 Marlon Stalyn Cargua Cando, Guillermo Edvin Machado Sotomayor

Esta obra está bajo una licencia internacional Creative Commons Atribución-NoComercial 4.0.

Resumen: La simulación y el control de robots manipuladores requieren representaciones eficientes para la composición de transformaciones rígidas asociadas al cálculo de la cinemática directa (FK). El presente estudio evalúa comparativamente el desempeño computacional de matrices de transformación homogénea y cuaterniones duales unitarios mediante un benchmark aplicado al manipulador colaborativo UR10 de Universal Robots, con seis grados de libertad. La metodología se fundamentó en la convención Denavit-Hartenberg, utilizando datos articulares extraídos mediante la API de Python de RoboDK y procesados en Julia mediante las librerías Quaternions.jl, ForwardDiff.jl y BenchmarkTools.jl. El entorno experimental estuvo compuesto por un procesador AMD Ryzen 7 de 2.00 GHz, 8 GB de RAM y arquitectura de 64 bits. Ambas representaciones fueron comparadas bajo condiciones controladas considerando como métricas el tiempo de ejecución y el uso de memoria durante la composición sucesiva de seis transformaciones rígidas. La equivalencia matemática entre ambos formalismos se verificó mediante la reconstrucción de la transformación homogénea a partir del cuaternión dual compuesto, observándose diferencias asociadas únicamente al redondeo numérico de punto flotante. Los resultados proporcionan evidencia empírica sobre el comportamiento computacional de ambas implementaciones y contribuyen al análisis de alternativas algebraicas para la optimización de operaciones asociadas con la cinemática directa en manipuladores robóticos.

Palabras clave: Cinemática directa, cuaterniones duales, Denavit-Hartenberg, manipulador UR10, matrices de transformación homogéneas.

Abstract: *The simulation and control of robotic manipulators require efficient representations for the composition of rigid transformations associated with forward kinematics (FK) computation. This study comparatively evaluates the computational performance of homogeneous transformation matrices and unit dual quaternions through a benchmark applied to the UR10 collaborative robot from Universal Robots, a six-degree-of-freedom manipulator. The methodology was based on the Denavit–Hartenberg convention, using joint data extracted through the RoboDK Python API and processed in Julia using the Quaternions.jl, ForwardDiff.jl, and BenchmarkTools.jl libraries. The experimental environment consisted of an AMD Ryzen 7 2.00 GHz processor, 8 GB of RAM, and a 64-bit architecture. Both representations were compared under controlled conditions, considering execution time and memory usage as evaluation metrics during the successive composition of six rigid transformations. The mathematical equivalence between both formalisms was verified by reconstructing the homogeneous transformation from the composed dual quaternion, with the observed differences attributed exclusively to floating-point numerical rounding. The results provide empirical evidence regarding the computational behavior of both implementations and contribute to the analysis of algebraic alternatives for optimizing operations associated with forward kinematics in robotic manipulators.*

Keywords*: Denavit–Hartenberg, dual quaternions, forward kinematics, homogeneous transformation matrices, UR10 manipulator.*

1. Introducción

La cinemática directa constituye un componente fundamental del modelado, control y simulación de robots manipuladores, debido a que permite determinar la posición y orientación del efector final a partir de una configuración articular determinada. Su formulación resulta indispensable para la planificación del movimiento, el seguimiento de trayectorias y la ejecución de tareas de manipulación en sistemas robóticos industriales y colaborativos (Liu et al., 2018; Menon et al., 2019). En este contexto, el UR10 de Universal Robots corresponde a un manipulador colaborativo de seis grados de libertad empleado en aplicaciones que requieren precisión, repetibilidad y coordinación espacial (Petrone et al., 2025). La resolución de su cinemática directa puede abordarse mediante diferentes formalismos matemáticos destinados a representar las relaciones geométricas existentes entre sus articulaciones y la pose final del robot.

Entre los procedimientos más utilizados para estructurar estas relaciones se encuentra la convención de Denavit-Hartenberg, ampliamente aplicada en el modelado cinemático de manipuladores de cadena abierta (Liu et al., 2018; Živković et al., 2022). A partir de sus parámetros es posible construir transformaciones rígidas sucesivas que describen la posición y orientación relativa entre los diferentes sistemas de referencia asociados a las articulaciones. Estas transformaciones pueden representarse mediante matrices homogéneas de dimensión 4 × 4, las cuales integran rotación y traslación dentro de una misma estructura algebraica. Su utilización se mantiene extendida en robótica debido a su interpretación geométrica directa, su compatibilidad con herramientas de simulación y su aplicación en algoritmos de cinemática y control.

Los cuaterniones duales constituyen una alternativa algebraica para representar movimientos rígidos mediante una formulación compacta que combina simultáneamente rotación y traslación. Esta representación ha sido empleada en cinemática directa e inversa, control de orientación, modelado dinámico, transformación de coordenadas y coordinación de sistemas multirrobot (AlAttar & Kormushev, 2020; Dantam, 2021; Giribet et al., 2025; Zeng et al., 2024). Su estructura permite describir transformaciones espaciales mediante ocho componentes sujetas a restricciones algebraicas, evitando parte de la redundancia presente en las matrices homogéneas (Adorno & Marinho, 2021; Qi et al., 2021). En consecuencia, los cuaterniones duales han adquirido relevancia como herramienta matemática para representar movimientos rígidos dentro del grupo especial euclidiano SE(3).

La equivalencia entre matrices homogéneas y cuaterniones duales permite describir una misma transformación rígida mediante formalismos algebraicos diferentes. Diversos trabajos han demostrado que ambas representaciones pueden emplearse para modelar movimientos de cuerpos rígidos, preservar relaciones geométricas y establecer correspondencias entre rotaciones y traslaciones tridimensionales (Cohen & Shoham, 2020; Condurache & Popa, 2023; Yaqub & Schröcker, 2025). Esta relación también ha sido estudiada en problemas de estimación, sincronización, calibración y control de robots, donde la formulación mediante cuaterniones o cuaterniones duales permite desarrollar representaciones compactas de la pose (Huang et al., 2026; Solà et al., 2018; Zhao et al., 2026). Por tanto, la elección entre ambos formalismos no depende exclusivamente de su validez matemática, sino también de las características de implementación y del contexto computacional en el que se utilizan.

Desde una perspectiva computacional, los cuaterniones duales presentan propiedades que pueden resultar favorables en aplicaciones que requieren la ejecución repetida de transformaciones rígidas. Su estructura de tamaño fijo, menor redundancia paramétrica y capacidad para unificar rotación y traslación han motivado su incorporación en bibliotecas y algoritmos orientados al modelado y control robótico (Adorno & Marinho, 2021; Dantam, 2021). Investigaciones recientes también han incorporado

matrices homogéneas y cuaterniones duales en modelos de aprendizaje y políticas cinemáticas, evidenciando que ambos formalismos continúan siendo relevantes en arquitecturas robóticas modernas (Diprasetya et al., 2025). Sin embargo, las ventajas estructurales atribuidas a una representación no implican necesariamente un mejor desempeño computacional en todas las implementaciones.

El rendimiento efectivo de un algoritmo depende del lenguaje de programación, la estructura de datos, la administración de memoria, las optimizaciones del compilador y la arquitectura de hardware utilizada durante la ejecución. Estudios sobre optimización de código han demostrado que modificaciones en la representación de datos y en las estrategias de compilación pueden producir diferencias relevantes en tiempo de procesamiento, consumo de memoria y confiabilidad computacional (Klepl et al., 2024; Santos et al., 2024). Estas consideraciones adquieren mayor importancia en aplicaciones robóticas que requieren cálculos repetitivos dentro de ciclos de control de alta frecuencia, donde pequeñas variaciones en el tiempo de ejecución pueden afectar la respuesta del sistema. En este tipo de escenarios, la evaluación experimental de las implementaciones constituye un complemento necesario al análisis puramente algebraico.

La necesidad de optimizar el cálculo cinemático resulta especialmente relevante en sistemas de control en tiempo real. En estos entornos, los algoritmos deben procesar transformaciones, estados articulares y referencias de movimiento dentro de intervalos reducidos para mantener una respuesta adecuada del manipulador (Wittmann et al., 2022). De manera adicional, las asignaciones dinámicas de memoria pueden introducir variabilidad en el tiempo de ejecución cuando intervienen mecanismos automáticos de administración de recursos. Por ello, el tiempo de procesamiento y el uso de memoria constituyen indicadores pertinentes para comparar implementaciones alternativas de una misma formulación cinemática.

En el presente estudio, RoboDK se emplea como entorno de simulación para obtener configuraciones articulares y poses asociadas al UR10, facilitando la integración entre el modelo robótico y el procedimiento experimental. Este tipo de plataformas permite reproducir trayectorias y configuraciones de robots industriales en condiciones controladas, además de proporcionar información cinemática para su posterior procesamiento (Pollák & Goryl, 2023). La implementación numérica se desarrolla en Julia, un lenguaje orientado al cálculo científico y al procesamiento de alto rendimiento que ha demostrado utilidad en aplicaciones computacionales intensivas (Roesch et al., 2023). Adicionalmente, BenchmarkTools.jl permite realizar mediciones repetidas y reproducibles del tiempo de ejecución y de las asignaciones de memoria, reduciendo la influencia de variaciones ocasionales durante la evaluación.

Aunque la literatura disponible documenta ampliamente las propiedades algebraicas de los cuaterniones duales y su aplicación en diferentes problemas robóticos, la evidencia comparativa sobre su desempeño computacional frente a las matrices homogéneas continúa siendo limitada. Los estudios existentes se han concentrado principalmente en cinemática, control, modelado, estimación de pose, transformación de coordenadas y formulaciones matemáticas de movimientos rígidos (AlAttar & Kormushev, 2020; Dantam, 2021; Nekoo et al., 2022; Xu & Halse, 2016). En la revisión realizada no se identificó un benchmark centrado específicamente en el UR10 que compare ambas representaciones bajo un mismo entorno de hardware y software, utilizando datos articulares extraídos directamente desde RoboDK. Esta ausencia evidencia la necesidad de evaluar experimentalmente si las diferencias estructurales entre ambas representaciones se traducen en variaciones observables de rendimiento.

Como supuesto de trabajo se plantea que, bajo el entorno evaluado, la composición mediante cuaterniones duales exhibirá menores tiempos de ejecución y una menor presión sobre el recolector de basura, debido a la naturaleza de tamaño fijo de su estructura de datos. Este supuesto se fundamenta en las características compactas de los cuaterniones duales y en su capacidad para representar de manera unificada las transformaciones rígidas, aunque su comportamiento efectivo depende de las condiciones específicas de implementación y ejecución (Adorno & Marinho, 2021; Dantam, 2021). En este contexto, se formula la siguiente pregunta de investigación: ¿cuál es el desempeño computacional, en términos de tiempo de ejecución y uso de memoria, de la composición de transformaciones rígidas mediante matrices homogéneas frente a cuaterniones duales unitarios en la cinemática directa del UR10 con datos obtenidos desde RoboDK, bajo condiciones controladas de hardware y software? Esta interrogante orienta la comparación empírica entre ambas representaciones dentro de un escenario computacional reproducible y delimitado.

El objetivo del presente estudio es evaluar cuantitativamente el desempeño computacional de las matrices de transformación homogénea y de los cuaterniones duales unitarios en la composición de la cinemática directa del UR10, considerando como parámetros de comparación el tiempo de ejecución y el uso de memoria mediante un protocolo de benchmark estandarizado y estadísticamente robusto. Para alcanzar este propósito, se integran datos articulares y de pose obtenidos desde la API de Python de RoboDK con implementaciones desarrolladas en Julia, de manera que ambas representaciones sean evaluadas bajo las mismas condiciones experimentales. La comparación se centra en el comportamiento computacional de implementaciones concretas y no en establecer una superioridad matemática general entre ambos formalismos, puesto que representan transformaciones rígidas equivalentes. Bajo este enfoque, los resultados permiten valorar si las diferencias estructurales descritas en la literatura se traducen en variaciones observables de rendimiento durante la ejecución de la cinemática directa.

Como primer aporte, se propone un flujo reproducible que integra la extracción de datos articulares y de pose del manipulador UR10 mediante la API de Python de RoboDK con la construcción de cuaterniones duales unitarios en Julia mediante diferenciación automática. Este flujo establece una conexión directa entre el entorno de simulación robótica y el entorno de cálculo numérico utilizado para ejecutar las transformaciones y registrar las métricas de desempeño. La integración permite mantener consistencia entre los datos de entrada empleados por ambas representaciones y facilita la repetición del procedimiento bajo configuraciones equivalentes. Las funciones desarrolladas para este propósito se encuentran documentadas dentro del software asociado al estudio, favoreciendo la reproducibilidad del procedimiento experimental (Cargua, 2026).

Como segundo aporte, se establece un protocolo de benchmark estadísticamente robusto compuesto por 10 000 muestras y un proceso de calibración adaptativa mediante BenchmarkTools.jl. El protocolo incorpora la documentación de las especificaciones de hardware y software utilizadas durante las mediciones, aspecto necesario para interpretar correctamente los resultados de rendimiento computacional. Esta estrategia permite disminuir la influencia de variaciones ocasionales en los tiempos de ejecución y obtener métricas comparables dentro de un mismo entorno experimental. Además, la descripción explícita de las condiciones de evaluación facilita la réplica del benchmark en otros sistemas y permite analizar posteriormente la sensibilidad de los resultados frente a diferentes configuraciones de hardware o software.

Como tercer aporte, se generan datos empíricos comparativos sobre el tiempo de ejecución y el uso de memoria de las implementaciones basadas en matrices homogéneas y cuaterniones duales. Esta evidencia permite contrastar las ventajas estructurales y computacionales atribuidas teóricamente a los

cuaterniones duales con mediciones obtenidas durante la ejecución de un caso específico de cinemática robótica (Adorno & Marinho, 2021; Dantam, 2021). La evaluación bajo un mismo entorno de hardware y software reduce la influencia de factores externos que podrían dificultar una comparación directa entre ambas representaciones. De esta manera, el estudio complementa el análisis algebraico disponible en la literatura con evidencia cuantitativa derivada de una implementación reproducible.

Como cuarto aporte, se desarrollan funciones de conversión bidireccional entre matrices homogéneas y cuaterniones duales, destinadas a facilitar la interoperabilidad entre el entorno de simulación y el entorno de cálculo numérico. Estas funciones permiten transformar de manera consistente una pose expresada mediante matrices homogéneas a su representación equivalente mediante cuaterniones duales y realizar también el proceso inverso. Su incorporación favorece la validación cruzada de las transformaciones obtenidas y facilita la integración de ambos formalismos dentro de un mismo flujo de trabajo. Este componente resulta particularmente útil en escenarios donde distintas herramientas o bibliotecas emplean representaciones diferentes para describir la posición y orientación de un manipulador.

El resto del artículo se organiza de la siguiente manera: la Sección 2 presenta las operaciones matemáticas preliminares necesarias para la composición de transformaciones rígidas mediante matrices homogéneas y cuaterniones duales, incluyendo el homomorfismo existente entre ambas estructuras algebraicas. La Sección 3 describe el algoritmo de trabajo, los recursos de hardware y software empleados, el conjunto de datos utilizado y el protocolo de benchmark implementado para evaluar ambas representaciones. La Sección 4 presenta y discute los resultados obtenidos mediante la macro @benchmark, con énfasis en el tiempo de ejecución, las asignaciones y el uso de memoria observados durante las pruebas. Finalmente, se presentan las conclusiones derivadas del análisis comparativo, así como las principales implicaciones y limitaciones asociadas al entorno experimental evaluado.

Fundamentos matemáticos de las representaciones cinemáticas

Una matriz de transformación homogénea es una matriz cuadrada de dimensión 4 × 4 que representa una transformación rígida en el espacio tridimensional mediante coordenadas homogéneas y constituye el primero de los dos formalismos matemáticos evaluados en este benchmark. Su estructura general es:

$$\mathrm{T} = \begin{bmatrix} R & p \\ 0^T & 1 \end{bmatrix}$$

Donde:

$R \in SO(3)$, matriz de cosenos directores ($DCM: 3{\times}3$); $R(R^T) = I$, con $det(R) = 1$.

$p \in \mathbb{R}^3$ es el vector de traslación.

$0^T = [000]$ es un vector fila nulo.

El conjunto de todas las matrices (T) con esta estructura, junto con la operación de multiplicación matricial, forma el grupo especial euclidiano (SE(3)), el grupo de Lie que describe todas las transformaciones rígidas posibles (rotación + traslación) en el espacio tridimensional (Solà et al., 2021). Sea el marco de referencia de la base del UR10, obtenido en la estación de RoboDK como "UR10 Base", cada transformación elemental entre articulaciones consecutivas del manipulador, derivada de los parámetros DH, viene dada por una matriz homogénea local de tamaño 4 × 4.

$$\mathrm{T_i} = \begin{bmatrix} R_i & p_i \\ 0^T & 1 \end{bmatrix}, R_i \in SO(3), p_i \in \mathbb{R}^3$$

Por ejemplo, la sexta transformación (link 6, de la muñeca a la brida), sin rotación relativa y con una traslación pura de 92.2 mm a lo largo del eje z, con el formato de matriz homogénea es:

```
between joint '/joint{5}' and object '/connection':
T_6 = [
 1.000000 -0.000000  0.000000  0.000000;
 0.000000  1.000000 -0.000000  0.000000;
 0.000000  0.000000  1.000000  92.200000;
 0.000000  0.000000  0.000000  1.000000
]
```

Dado que el UR10 posee seis grados de libertad, la pose resultante de la brida T_{total} se obtiene mediante la composición sucesiva de las n = 6 transformaciones elementales $T_1, T_2, T_3, T_4, T_5, T_6$; una por cada joint del manipulador, a través del producto matricial en SE(3) (Liu et al., 2018) :

$$T_{total} = T_1 T_2 T_3 \mathrm{T_4 T_5} T_6 = \begin{bmatrix} R_1 & p_1 \\ 0^T & 1 \end{bmatrix}\begin{bmatrix} R_2 & p_2 \\ 0^T & 1 \end{bmatrix}\begin{bmatrix} R_3 & p_3 \\ 0^T & 1 \end{bmatrix}\dots\begin{bmatrix} R_6 & p_6 \\ 0^T & 1 \end{bmatrix} \quad (1)$$

Por la regla de multiplicación en SE(3), la estructura del resultado es:

$$\mathrm{T_{total}} = \begin{bmatrix} R_{tot} & p_{tot} \\ 0^T & 1 \end{bmatrix}$$

Donde $R_{tot} = R_1 R_2 R_3 R_4 R_5 R_6$ y p_{tot} se obtiene recursivamente como $P_{tot} = p_1 + R_1(p_2 + R_2(p_3 + \cdots))$ es decir, cada traslación local se rota por la orientación acumulada de las articulaciones precedentes antes de sumarse a la traslación total (Huang et al., 2026) .

Cuaterniones

Habiendo establecido en la Sección 2.1 la matriz de transformación homogénea como primer formalismo del benchmark, se presenta a continuación el segundo formalismo evaluado: el cuaternión dual unitario. Su construcción se formaliza a partir de tres bloques conceptuales: cuaterniones simples, números duales y diferenciación automática. Estos conceptos se integran en una sola narrativa, alineada con la implementación desarrollada en Julia mediante el paquete Quaternions.jl.

Cuaterniones Simples

Un cuaternión es un número hipercomplejo de cuatro dimensiones, introducido por Sir William Rowan Hamilton, capaz de representar rotaciones en el espacio tridimensional sin las singularidades propias de los ángulos de Euler:

$$H = w + xi + yj + zk \quad (2)$$

donde: w, x, y y $z \in \mathbb{R}$ e i, j y k son las unidades imaginarias hipercomplejas (Qi et al., 2021). De forma equivalente H admite una representación escalar-vectorial, $H = [w, v]$,con $v = [x, y, z]$ que es la que adopta internamente el tipo Quaternion{T} en la librería Quaternions.jl al instanciarse mediante el constructor quat(w, x, y, z) (Alattar & Kormushev, 2020). Esta correspondencia directa entre la notación matemática $[w, v]$ y la robustez de Julia es la que permite trasladar, sin ambigüedad, cada matriz de rotación R_i de la sección 2.1 a su cuaternión equivalente.

Las operaciones aritméticas: multiplicación escalar, suma, producto de Hamilton (⊗), conjugado y norma, están precargadas de forma nativa sobre Quaternion{T} en Quaternions.jl (operadores *, +, conj, abs). El producto de Hamilton entre dos cuaterniones: $H_1 = [w_1, v_1]$ y $H_2 = [w_2, v_2]$ se define como: $H_1 \otimes H_2 = [w_1 w_2 - v_1 v_2, w_1 v_2 + w_2 v_1 + v_1 \times v_2]$. Un cuaternión unitario, ($\| H \| = 1$) codifica una rotación de un ángulo θ alrededor de un eje unitario $u = [u_x, u_y, u_z]$ mediante: $\mathrm{h}_{\mathrm{rot}}^{\mathrm{u}} = \left[\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right) u\right]$ (Giribet et al., 2025) .

Una traslación pura de magnitud m a lo largo del eje unitario $t = [t_x, t_y, t_z]$, se representa como un cuaternión puro (parte escalar nula): $\mathrm{h}_{\mathrm{trans}}^{\mathrm{t}} = [0, mt] = [0, mt_x, mt_y, mt_z]$, este es el formato que exporta el script de extracción de datos de RoboDK (download-FK.py) para cada transformación local del UR10. Así, la sexta transformación (link 6, de la muñeca a la brida), sin rotación relativa y con una traslación pura de 92.2 mm a lo largo del eje z, con el formato de cuaternión simple y vector de traslación es:

between joint '/joint5' and object '/connection' [cuaternión + traslación]:

h_6 = quat(1.000000, 0.000000, 0.000000, 0.000000) # [qw, qx, qy, qz]

t_6 = [0.000000, 0.000000, 92.200000] # [x, y, z] (mm)

Números Duales

Los números duales extienden los números reales de forma análoga a como los números complejos extienden $\mathbb{R}$ con $i^2 = -1$, para el efecto utilizan la propiedad $\begin{cases} \varepsilon^2 = 0 \\ \varepsilon \neq 0 \end{cases}$ (Zeng et al., 2024). Un número dual se define como $a = p + \varepsilon$ d, donde p corresponde al parte primal y d la parte tangente.

Introducidos por William Kingdon Clifford, estos números son el fundamento algebraico de la diferenciación automática (AD): si se evalúa una función *f* en un argumento dual x + ε (es decir, con parte primal x y parte tangente unitaria), la condición ε² = 0 hace que el resultado se expanda exactamente como $\mathrm{f}(x + \varepsilon) = f(x) + \mathrm{f}'(x)$ de modo que el coeficiente que acompaña a ε es, en ese caso particular, la derivada exacta *f'(x)*, sin las aproximaciones de la diferenciación numérica ni la complejidad simbólica de la diferenciación tradicional.

Conviene distinguir aquí tres nociones que, aunque relacionadas, no son equivalentes: (i) el álgebra dual como estructura puramente algebraica, definida únicamente por la propiedad ε² = 0; (ii) la diferenciación automática (AD), que constituye una de las posibles interpretaciones o aplicaciones de dicha álgebra, mas no la única; y (iii) el significado geométrico que se asigna a la parte tangente en un contexto particular, como el de los cuaterniones duales, donde la misma álgebra codifica una traslación.

Esta distinción importa porque, en la implementación de Julia de este benchmark, se reutiliza la maquinaria de AD (ForwardDiff.jl) como contenedor algebraico ya construido para el nivel (iii); aquello es una elección de implementación y no una necesidad matemática específica; así, cada componente del cuaternión se instancia en Julia como ForwardDiff.Dual{Nothing, T, 1}, de modo que la multiplicación de los mismos, precargada por Quaternions.jl, propaga automáticamente en la parte tangente la información de traslación.

Cuaterniones Duales

Un cuaternión dual $\widehat{H}$ se define como un número dual cuyas partes primal y tangente son, a su vez, cuaterniones: $\widehat{H} = q_r + \epsilon q_t = (w_r + x_r i + y_r j + z_r k) + \varepsilon(w_t + x_t i + y_t j + z_t k)$. Donde q_r es la parte real (rotación), y q_t es la parte tangente (traslación) con $\varepsilon^2 = 0 \wedge \varepsilon \neq 0$; de tal manera que la dupla (q_r, q_t) codifica de forma unificada y sin singularidades la transformación rígida completa (rotación + traslación) en SE(3) (Dantam, 2021) .

En Julia, esta estructura se materializa en el tipo compuesto DualQuaternion{T}, cada una de las cuatro componentes del Quaternion subyacente es, en sí misma, un ForwardDiff.Dual{Nothing, T, 1} (Cohen & Shoham, 2020); el parte primal de cada dual almacena la componente de q_r , y la parte tangente almacena la componente correspondiente a q_t.

Composición de Roto-Traslaciones con $\widehat{H}$ en $\mathbb{R}^3$

Sea $\mathcal{F}_0$ el marco de referencia base del UR10, cada transformación elemental de la brida se representa por un cuaternión dual unitario $\widehat{H}_i = q_{r_i} + \varepsilon q_{t_i}$ donde q_{r_i} es un cuaternión unitario ($w_r^2 + x_r^2 + y_r^2 + z_r^2 = 1$) que codifica la rotación y q_{t_i} recopila la traslación mediante la relación: $q_{t_i} = \frac{1}{2} t_i \otimes q_{r_i}, t_i = [0, d_i]$, siendo t_i el cuaternión puro asociado al vector de traslación $d_i = (d_x, d_y, d_z)$, el vector t que entrega el API de RoboDK. Esta relación es la que implementa la función motodual_from_quat(q_r,t) del script de Julia (dualquat_composicion.jl), tomando como entrada la dupla (q_i, t_i) extraídos de la simulación.

Un punto $p \in \mathbb{R}^3$ representado como cuaternión puro $P = [0, p]$ se transforma mediante $\mathrm{P}' = \widehat{\mathrm{H}}\mathrm{P}\widehat{H}^*$ donde $\widehat{H}^* = q_r^* + \varepsilon q_t^*$ es el conjugado dual, con $q_r^* = [w_r - v_r]$ y $q_t^* = [w_t - v_t]$ (Xu & Halse, 2018) . La composición de dos transformaciones $\widehat{H}_1 = q_{r_1} + \varepsilon q_{t_1}$ y $\widehat{H}_2 = q_{r_2} + \varepsilon q_{t_2}$ se obtiene mediante el producto de cuaterniones duales. Dado que $\epsilon^2 = 0$ la transformación resultante $\widehat{H}_f = \widehat{H}_2 \widehat{H}_1$ se reduce a: $\widehat{H}_u = \mathrm{q_{r_2} q_{r_1}} + \epsilon(\mathrm{q_{r_2} q_{t_1}} + \mathrm{q_{t_2} q_{r_1}})$. En la cadena cinemática serial del cobot UR10, dada la secuencia de transformaciones locales $\widehat{H}_1 = q_{r_i} + \varepsilon q_{t_i}$ para $i = 1, \ldots, 6$, la pose final de la brida se calcula mediante la multiplicación sucesiva de los seis cuaterniones duales:

$$\widehat{H}_{total} = \widehat{H}_1 \widehat{H}_2 \widehat{H}_3 \widehat{H}_4 \widehat{H}_5 \widehat{H}_6 \tag{3}$$

Esta expresión es el análogo exacto, en el álgebra dual, de la composición matricial $T_{\mathrm{total}} = T_1 T_2 T_3 T_4 T_5 T_6$ presentada en la ecuación 2.1, ambas describen la misma cadena cinemática del UR10, pero mediante estructuras algebraicas distintas. Computacionalmente, esta composición es la que se somete a evaluación de desempeño mediante BenchmarkTools.jl, contrastando su costo en tiempo de ejecución y memoria contra el de la composición matricial equivalente.

Homomorfismo de Grupos T $\cong$ $\widehat{H}$

Resolver el problema de la cinemática directa del UR10 admite, como se ha mostrado, dos caminos algebraicos: el de las matrices homogéneas $T \in SE(3)$ y el de los cuaterniones duales $\widehat{H}$. En robótica, el modelo cinemático describe la relación entre el espacio articular y el espacio operativo mediante la colocación de sistemas de coordenadas solidarias a cada eslabón y la determinación de las transformaciones espaciales entre sistemas sucesivos.

Teorema 1. El grupo euclidiano especial $SE(3)$ y el grupo de cuaterniones duales unitarios $\widehat{H}$ están relacionados mediante el homomorfismo: $\phi: SE(3) \rightarrow \widehat{H}_u, \phi(T) = (1 + \varepsilon\tilde{\rho}) q_r$, donde: $\mathrm{T} = \begin{bmatrix} R & \rho \\ 0 & 1 \end{bmatrix}$ y su inverso: $\phi^{-1}: \widehat{H}_u \rightarrow \mathrm{SE(3)}; \phi^{-1}(\widehat{\mathrm{H}}) = \begin{bmatrix} R & \rho \\ 0 & 1 \end{bmatrix}$.

Esto implica, que existe un homomorfismo de grupos de Lie no abelianos $SE(3) \cong \widehat{H}_u$, ambas estructuras describen exactamente los mismos movimientos rígidos, admiten conversión mutua sin pérdida de información (Zhao et al., 2026), y lo más relevante para este benchmark, preservan la operación de composición. Integrar transformaciones en $SE(3)$ mediante producto matricial (ecuación 1) es matemáticamente equivalente a componerlas en $\widehat{H}_u$ mediante producto de cuaterniones duales (ecuación 3). Esta equivalencia es la que justifica, metodológicamente, el diseño del estudio; ambos formalismos calculan exactamente la misma pose final de la brida del UR10 a partir de los mismos seis pares de datos (h_i, t_i) extraídos del API de RoboDK.

2. Metodología

La investigación adoptó un enfoque cuantitativo, debido a que su propósito central consistió en la medición objetiva de variables numéricas relacionadas con el tiempo de ejecución y el uso de memoria. Estas variables se evaluaron mediante instrumentos de medición computacional, específicamente la macro @benchmark del paquete BenchmarkTools.jl. El tratamiento de los datos se realizó a partir de 10 000 muestras obtenidas durante la ejecución de las pruebas. Posteriormente, se analizaron métricas de tendencia central y dispersión para caracterizar el comportamiento computacional de cada implementación.

El estudio se desarrolló mediante un diseño no experimental, de corte transversal y alcance comparativo, orientado específicamente a la ejecución de un benchmark computacional. No se manipularon variables independientes ni se establecieron grupos de control, debido a que la comparación se centró en dos implementaciones ejecutadas bajo un mismo entorno experimental. Las condiciones de hardware, software y configuración se mantuvieron constantes durante las pruebas para reducir la influencia de factores externos sobre las mediciones. Bajo estas condiciones, se observaron y compararon los comportamientos computacionales de las matrices homogéneas y los cuaterniones duales unitarios.

Para la obtención de los datos correspondientes a la cinemática directa del UR10, se empleó la API de Python de RoboDK. El marco de referencia externo respecto del cual se reportó la pose final del manipulador fue identificado en la estación de simulación como “UR10 Base”. Una vez establecidos los parámetros de Denavit-Hartenberg y el sistema de referencia correspondiente, se recuperó la pose relativa mediante la función PoseWrt(). Este procedimiento permitió disponer de la información espacial necesaria para reconstruir posteriormente las transformaciones asociadas con la cadena cinemática del robot.

De manera paralela, se extrajeron los ángulos articulares correspondientes a la configuración evaluada mediante el método robot.Joints(), que proporcionó los seis valores angulares asociados con los grados de libertad del UR10. Estos valores se convirtieron a radianes y se utilizaron como variables articulares $\theta_i$para construir individualmente las seis matrices de transformación homogénea locales. A partir de dichas matrices se efectuó la composición secuencial correspondiente a la cinemática directa del manipulador. Este procedimiento permitió obtener de manera independiente la transformación acumulada desde la base hasta la brida del robot.

Con el propósito de verificar la coherencia entre la cadena cinemática calculada independientemente y la solución proporcionada por RoboDK, se contrastó la composición de las seis matrices homogéneas con el resultado obtenido mediante el método robot.SolveFK(). Este procedimiento resolvió la cinemática directa desde el sistema de referencia de la base hasta la brida del manipulador para la configuración articular seleccionada. La comparación permitió comprobar la correspondencia entre la formulación implementada y la solución generada por el simulador. De esta manera, se estableció una referencia común antes de realizar las posteriores transformaciones y evaluaciones computacionales.

Cabe señalar que la estación utilizada no contó con una herramienta o *Tool Center Point* (TCP) conectada al manipulador durante la extracción de los datos. Por esta razón, las poses obtenidas correspondieron a la posición y orientación final de la brida del UR10 y no al extremo de una herramienta de trabajo. Esta delimitación se mantuvo constante durante todas las etapas de procesamiento y comparación de las representaciones matemáticas. Su consideración permitió evitar discrepancias derivadas de transformaciones adicionales asociadas con herramientas externas al modelo cinemático del robot.

Posteriormente, mediante la función pose_2_quaternion(), se extrajo el cuaternión simple correspondiente, expresado como $[q_w, q_x, q_y, q_z]$, junto con su respectivo vector de traslación $[x, y, z]$. Ambos componentes se utilizaron como representación intermedia para la posterior construcción del cuaternión dual unitario en el entorno de Julia. La información obtenida permitió conservar de manera diferenciada los componentes de rotación y traslación antes de integrarlos en la estructura dual. Este procedimiento estableció el vínculo entre los datos obtenidos desde RoboDK y las operaciones algebraicas implementadas posteriormente en Julia.

Para la implementación computacional se utilizó el entorno REPL de Julia, previa carga de las librerías Quaternions.jl y ForwardDiff.jl. En este entorno se definieron, entre otras, las estructuras y funciones const DualQuaternion{T}, motodual_from_quat(q_r, t), motodual(T) y dualmoto(H). La estructura DualQuaternion{T} representó el cuaternión dual mediante cuatro componentes hipercomplejas, una escalar y tres imaginarias, constituidas por números duales del tipo ForwardDiff.Dual{Nothing,T,1}. Esta implementación permitió integrar dentro de una misma estructura los componentes requeridos para representar las transformaciones rígidas utilizadas durante el estudio.

Cada número dual almacenó internamente un valor primal y una parte tangencial, cuya interacción durante las operaciones aritméticas permitió propagar la información correspondiente a la traslación. La función motodual_from_quat(q_r, t) constituyó el punto de conexión entre los datos extraídos desde RoboDK y el álgebra dual implementada en Julia. Esta función construyó directamente un cuaternión dual a partir del cuaternión simple de rotación q_ry del vector de traslación t, sin requerir una conversión previa a una matriz homogénea de 4 × 4. De este modo, se estableció una ruta directa de transformación entre los datos proporcionados por el simulador y la representación dual empleada en el benchmark.

Las funciones motodual(T) y dualmoto(H) se utilizaron como procedimientos complementarios para realizar conversiones entre matrices homogéneas y cuaterniones duales. Ambas funciones aprovecharon la correspondencia algebraica existente entre las dos representaciones y permitieron transformar matrices en cuaterniones duales y efectuar el procedimiento inverso. Estas funciones, junto con motodual_from_quat(q_r, t), fueron desarrolladas específicamente por el autor como parte de la implementación requerida para el estudio. Su incorporación permitió disponer de mecanismos de conversión necesarios para verificar la equivalencia entre las representaciones evaluadas.

Los cálculos correspondientes a las matrices homogéneas y la conversión hacia cuaterniones duales se realizaron mediante los scripts download-FK.py y dualquat_composicion.jl, desarrollados específicamente para esta investigación. El primer script se utilizó para extraer desde RoboDK la información cinemática necesaria, mientras que el segundo permitió ejecutar las operaciones asociadas con la representación mediante cuaterniones duales en Julia. Ambos scripts formaron parte del flujo computacional utilizado para garantizar la reproducibilidad de la extracción, transformación y procesamiento de los datos. El código desarrollado para estas operaciones se documentó como parte del software asociado al estudio (Cargua, 2026).

Explicación del Algoritmo

Como se estableció en la Sección 2.2.3, el cuaternión dual unitario fue capaz de representar transformaciones rígidas generales, propiedad que permitió emplear la convención de Denavit-Hartenberg de forma análoga al caso matricial, sustituyendo cada matriz homogénea local por su cuaternión dual equivalente. El estudio adoptó una evaluación numérica orientada al procesamiento de datos obtenidos a partir de la estación del UR10 modelada en RoboDK. Esta estrategia permitió trabajar con las mismas transformaciones locales en ambos formalismos y mantener condiciones equivalentes durante la comparación. De este modo, se garantizó que las diferencias observadas correspondieran al procedimiento de composición implementado en cada representación.

Cabe precisar que el objeto del benchmark fue la operación de composición de transformaciones rígidas y no el proceso completo de cálculo de la cinemática directa. En consecuencia, la evaluación se concentró específicamente en la multiplicación encadenada de las seis transformaciones locales, expresadas mediante matrices homogéneas o mediante sus cuaterniones duales unitarios equivalentes. Esta delimitación permitió aislar la operación computacional de interés y reducir la intervención de procesos adicionales que pudieran afectar las mediciones de rendimiento. Por tanto, los resultados obtenidos correspondieron exclusivamente al desempeño de la composición sucesiva bajo las condiciones experimentales establecidas.

El algoritmo se organizó en dos etapas acopladas: la extracción de los datos fuente desde RoboDK y la composición y evaluación de las transformaciones rígidas en Julia. La primera etapa proporcionó las configuraciones articulares, poses y transformaciones requeridas para construir las representaciones evaluadas. La segunda etapa ejecutó la composición sucesiva mediante matrices homogéneas y cuaterniones duales unitarios, además de registrar las métricas computacionales correspondientes. El flujo general de ambas etapas se resume en las tablas 1 y 3.

Tabla 1

Etapa A. Extracción de datos cinemáticos (Python/RoboDK). Script download-FK.py

Paso	Procedimiento
1	Se estableció la conexión con la estación de RoboDK mediante Robolink() y se seleccionó el robot UR10 utilizando ItemUserPick(..., ITEM_TYPE_ROBOT).
2	Se localizó el sistema de referencia externo denominado "UR10 Base" mediante RDK.Item(), con selección interactiva dentro de la estación.
3	Se obtuvieron los seis ángulos articulares correspondientes a la configuración vigente mediante robot.Joints() y posteriormente se convirtieron de grados a radianes para su utilización como variables articulares θ_i.
4	Para cada eslabón se construyó la matriz homogénea local a partir de los parámetros de Denavit-Hartenberg del UR10 presentados en la tabla 2. Para ello se utilizó la función build_DH_matrix(a, d, α, θ), que implementó exactamente la forma estándar: $T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$
5	De cada transformación se extrajo el par conformado por el cuaternión de rotación y el vector de traslación mediante pose_2_quaternion() del módulo robodk.robomath. Los valores fueron generados en el formato quat(qw, qx, qy, qz) para su posterior utilización en el entorno REPL de Julia.
6	Se validó la composición Base→Brida mediante su comparación con la solución nativa obtenida a través de robot.SolveFK(). De igual manera, la composición Referencia→Brida se contrastó con robot.PoseWrt(frame_ref), obteniéndose un error máximo absoluto de 1 mm en ambos casos.

Tabla 2

Parámetros DH del UR10 (mm; α y θ en grados), usados por build_DH_matrix()

Articulación	a_i(mm)	d_i(mm)	α_i(°)	θ_i(°)
1	0.0	127.300	90.0	0.0
2	−612.7	0.000	0.0	−90
3	−572.3	0.000	0.0	−90
4	0.0	163.941	90.0	0.0
5	0.0	115.700	−90.0	90
6	0.0	92.200	0.0	0.0

Tabla 3

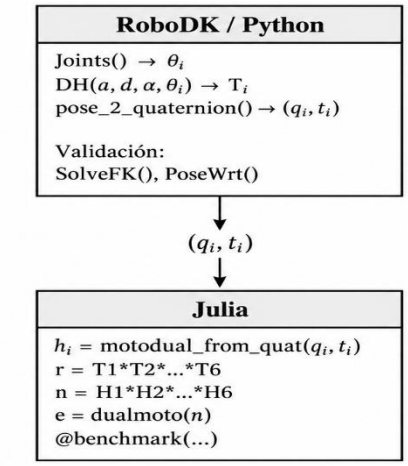
Etapa B. Composición en cuaterniones duales (Julia). El script dualquat_composicion.jl

Paso	Procedimiento
1	Construcción de cada matriz homogénea equivalente $T_1, \ldots, T_6$y de cada cuaternión dual local $H_1, \ldots, H_6$mediante motodual_from_quat(q_r, t), que implementó la relación de la ecuación correspondiente.
2	Composición en ambos formalismos: r = T_1*T_2*T_3*T_4*T_5*T_6 para la representación matricial y n = H_1*H_2*H_3*H_4*H_5*H_6 para los cuaterniones duales, correspondientes, respectivamente, a la ecuación (1) y a la ecuación (3) de la Sección 2.
3	Reconstrucción inversa mediante e = dualmoto(n), que convirtió el cuaternión dual compuesto nuevamente en una matriz de 4 × 4 mediante rotmatrix_from_quat() y translation(), permitiendo comparar numéricamente r con e como verificación directa del homomorfismo.
4	Evaluación del desempeño de ambas composiciones mediante la macro @benchmark de BenchmarkTools.jl.

La figura 1 muestra el flujo metodológico utilizado para la extracción, transformación y evaluación de los datos cinemáticos del UR10.

Figura 1

Flujo de extracción (RoboDK/Python) y composición-verificación-benchmark (Julia), adaptado al enfoque numérico basado en datos reales de simulación



En la primera etapa, desarrollada en RoboDK/Python, se obtuvieron los ángulos articulares mediante Joints(), se construyeron las matrices de transformación homogénea a partir de los parámetros DH y se extrajeron los cuaterniones de rotación junto con sus vectores de traslación mediante pose_2_quaternion(). Además, se realizó la validación de la cinemática directa mediante SolveFK() y PoseWrt(). Posteriormente, los pares (q_i, t_i)fueron transferidos al entorno Julia, donde se generaron los cuaterniones duales locales, se efectuó la composición de matrices homogéneas y cuaterniones duales, se reconstruyó la matriz equivalente y se evaluó el desempeño computacional mediante @benchmark.

Materiales

El estudio se ejecutó en una computadora de escritorio HP equipada con un procesador AMD Ryzen 7 de 2.00 GHz, 8 GB de memoria RAM y arquitectura de 64 bits. Como lenguaje de programación se utilizó Julia 1.12, mientras que BenchmarkTools.jl se empleó como herramienta para la ejecución del benchmark computacional. Para las operaciones relacionadas con el álgebra de cuaterniones y la diferenciación automática se utilizaron las librerías Quaternions.jl y ForwardDiff.jl. La simulación y extracción de los datos cinemáticos se realizaron mediante RoboDK 5.9 y su API de Python, específicamente a través del módulo robolink y del script download-FK.py.

Las matrices de transformación homogénea se implementaron mediante el tipo Matrix{Float64}, con una dimensión de 4 × 4. Cada operación de composición se efectuó mediante la multiplicación de matrices densas de esta dimensión. En términos aritméticos, cada multiplicación involucró 64 productos y 48 sumas de punto flotante. Esta implementación constituyó la representación matricial utilizada posteriormente durante las pruebas de desempeño computacional.

Los cuaterniones duales unitarios se implementaron mediante el tipo compuesto de Julia const DualQuaternion{T} = Quaternion{ForwardDiff.Dual{Nothing, T, 1}}. Esta estructura encapsuló las partes real y dual en un único objeto inmutable de tamaño fijo, conformado por ocho números de punto flotante almacenados sin recurrir a arreglos dinámicos. La composición se ejecutó mediante el operador * implementado en Quaternions.jl y aplicado sobre elementos del tipo ForwardDiff.Dual. Durante esta operación, la parte tangencial permitió propagar automáticamente la información correspondiente a la traslación.

Dataset y parámetros cinemáticos

Se calcularon las matrices de transformación homogénea y los cuaterniones duales equivalentes para las seis articulaciones del UR10, utilizando los parámetros de Denavit-Hartenberg extraídos directamente del panel de parámetros nominales de RoboDK. Cada transformación local fue representada mediante ambos formalismos con el propósito de disponer de entradas equivalentes para la comparación computacional. Las matrices homogéneas se implementaron como estructuras Matrix{Float64} de dimensión 4 × 4. Los cuaterniones duales se representaron mediante el tipo Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} (tabla 4).

Se evaluó la composición de seis transformaciones elementales, una por cada articulación del UR10, con el propósito de obtener la pose final de la brida. Todas las transformaciones correspondieron a la configuración articular vigente en la estación de RoboDK al momento de la extracción. Esta misma configuración se mantuvo para ambos formalismos con el fin de garantizar condiciones equivalentes durante la comparación. De este modo, la evaluación se concentró exclusivamente en las diferencias computacionales asociadas con cada representación.

Tabla 4

Matrices de transformación homogéneas y cuaterniones duales equivalentes

Transformación	Matriz homogénea	Cuaternión dual
1	T_1 = 4 × 4 Matrix{Float64}: 1.0 −0.0 0.0 0.0 0.0 0.0 −1.0 0.0 0.0 1.0 0.0 127.3 0.0 0.0 0.0 1.0	H_1 = Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} s = Dual{Nothing}(0.7071067811865476, 0.0) v1 = Dual{Nothing}(0.7071067811865475, 0.0) v2 = Dual{Nothing}(0.0, 45.00734662252375) v3 = Dual{Nothing}(0.0, 45.007346622523755)
2	T_2 = 4 × 4 Matrix{Float64}: 0.0 1.0 −0.0 −0.0 −1.0 0.0 −0.0 612.7 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0	H_2 = Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} s = Dual{Nothing}(0.7071067811865476, 0.0) v1 = Dual{Nothing}(0.0, −216.62216241649884) v2 = Dual{Nothing}(−0.0, 216.62216241649887) v3 = Dual{Nothing}(−0.7071067811865475, 0.0)
3	T_3 = 4 × 4 Matrix{Float64}: 0.0 1.0 −0.0 −0.0 −1.0 0.0 −0.0 572.3 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0	H_3 = Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} s = Dual{Nothing}(0.7071067811865476, 0.0) v1 = Dual{Nothing}(0.0, −202.33860543653054) v2 = Dual{Nothing}(−0.0, 202.33860543653057) v3 = Dual{Nothing}(−0.7071067811865475, 0.0)
4	T_4 = 4 × 4 Matrix{Float64}: 1.0 −0.0 0.0 0.0 0.0 0.0 −1.0 0.0 0.0 1.0 0.0 163.941 0.0 0.0 0.0 1.0	H_4 = Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} s = Dual{Nothing}(0.7071067811865476, 0.0) v1 = Dual{Nothing}(0.7071067811865475, 0.0) v2 = Dual{Nothing}(0.0, 57.96189640725189) v3 = Dual{Nothing}(0.0, 57.9618964072519)
5	T_5 = 4 × 4 Matrix{Float64}: 0.0 −0.0 −1.0 0.0 1.0 0.0 0.0 0.0 0.0 −1.0 0.0 115.7 0.0 0.0 0.0 1.0	H_5 = Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} s = Dual{Nothing}(0.5, −28.925) v1 = Dual{Nothing}(−0.5, 28.925) v2 = Dual{Nothing}(−0.5, −28.925) v3 = Dual{Nothing}(0.5, 28.925)
6	T_6 = 4 × 4 Matrix{Float64}: 1.0 −0.0 0.0 0.0 0.0 1.0 −0.0 0.0 0.0 0.0 1.0 92.2 0.0 0.0 0.0 1.0	H_6 = Quaternion{ForwardDiff.Dual{Nothing, Float64, 1}} s = Dual{Nothing}(1.0, 0.0) v1 = Dual{Nothing}(0.0, 0.0) v2 = Dual{Nothing}(0.0, 0.0) v3 = Dual{Nothing}(0.0, 46.1)

El conjunto de prueba del benchmark estuvo compuesto por 10 000 muestras para cada formalismo. Estas muestras correspondieron exclusivamente a ejecuciones repetidas de la misma composición de seis transformaciones fijas y no a 10 000 configuraciones articulares diferentes. Durante la ejecución se aplicó interpolación mediante $ sobre las variables de entrada para evitar el acceso a variables globales dentro de BenchmarkTools.jl. Esta estrategia permitió mantener constantes los datos de entrada y concentrar las mediciones en el costo computacional de la composición.

Protocolo Benchmark

El protocolo de benchmark se diseñó con el propósito de maximizar la validez interna y externa de las mediciones, aprovechando las capacidades proporcionadas por BenchmarkTools.jl. Las variables de entrada correspondientes a las matrices $T_1, \dots, T_6$y a los cuaterniones duales $H_1, \dots, H_6$se declararon fuera del bloque de benchmark. Posteriormente, estas variables se interpolaron dentro de cada expresión mediante el operador $, con el fin de evitar el uso de variables globales que pudieran degradar el rendimiento. Antes de iniciar las mediciones se realizó una ejecución previa o *warm-up* para excluir de los resultados el tiempo asociado con la compilación JIT.

El muestreo se efectuó de manera adaptativa, sin establecer previamente un número fijo de evaluaciones por muestra. BenchmarkTools.jl ajustó dinámicamente el parámetro evals para cada implementación de acuerdo con su velocidad de ejecución. Este procedimiento permitió obtener mediciones suficientemente representativas dentro de cada muestra y reducir la influencia de ejecuciones aisladas. La estrategia se mantuvo de manera equivalente para ambos formalismos con el propósito de preservar condiciones comparables durante el benchmark.

El tamaño de la muestra se fijó en 10 000 ejecuciones para cada formalismo. Esta cantidad permitió obtener una distribución amplia de los tiempos de ejecución y calcular estadísticos descriptivos asociados con tendencia central y dispersión. Las 10 000 muestras correspondieron a repeticiones de una misma composición de seis transformaciones fijas bajo condiciones controladas. En consecuencia, el benchmark evaluó la estabilidad y el costo computacional de cada implementación y no la variabilidad derivada de diferentes configuraciones articulares.

Los benchmarks se definieron mediante expresiones equivalentes para ambas representaciones. Para las matrices de transformación homogénea se utilizó la expresión @benchmark(T_1*T_2*T_3*T_4*T_5*T_6), mientras que para los cuaterniones duales unitarios se empleó @benchmark(H_1*H_2*H_3*H_4*H_5*H_6). En ambos casos, la operación evaluada correspondió exclusivamente a la composición sucesiva de las seis transformaciones locales asociadas a las articulaciones del UR10. Esta equivalencia en la definición experimental permitió comparar directamente el costo computacional de cada representación, manteniendo constante la operación matemática evaluada y evitando diferencias derivadas del procedimiento de cálculo.

Las métricas registradas comprendieron el tiempo de ejecución, expresado mediante valores mínimo, máximo, mediana, media y desviación estándar. También se evaluó el impacto del recolector de basura mediante las métricas de GC, incluyendo media, desviación estándar, mínimo y máximo. De forma complementaria, se registraron la memoria estimada y el número de asignaciones realizadas durante cada operación de composición. Estos indicadores permitieron caracterizar tanto la velocidad de ejecución como el comportamiento de memoria de ambas implementaciones.

Para garantizar la replicabilidad del procedimiento, todos los benchmarks se ejecutaron dentro del mismo proceso activo de Julia y utilizando las mismas versiones de los paquetes requeridos. Se mantuvieron constantes las condiciones de hardware, software y configuración durante toda la

ejecución de las pruebas. Esta estrategia permitió reducir fuentes externas de variabilidad que pudieran afectar la comparación entre las matrices homogéneas y los cuaterniones duales. De este modo, las diferencias observadas se atribuyeron principalmente al comportamiento computacional de las implementaciones evaluadas.

3. Resultados

Verificación Numérica de la Composición

Antes de comparar el desempeño computacional, se verifica que ambos formalismos produzcan la misma pose final a partir de los mismos seis pares (q_i, t_i) extraídos de la estación del UR10 en RoboDK, confirmando empíricamente el homomorfismo $SE(3) \cong \widehat{H}_u$ del Teorema 1.

La composición matricial directa (ecuación 1) entrega:

$$r = T_1T_2T_3T_4T_5T_6 = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 664.500 \\ -1.0 & 0.0 & 0.0 & -163.941 \\ 0.0 & -1.0 & 0.0 & 855.700 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

y la composición de cuaterniones duales (ecuación 3), $n = H_1 * H_2 * H_3 * H_4 * H_5 * H_6$. Al reconvertir el resultado a matriz homogénea mediante e = dualmoto(n), se obtiene:

$$e = \begin{bmatrix} 0.0 & 3.331 \times 10^{-16} & 1.0 & 664.500 \\ -1.0 & 0.0 & 0.0 & -163.941 \\ 0.0 & -1.0 & 2.220 \times 10^{-16} & 855.700 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

El error máximo elemento a elemento entre (r) y (e) se encontró en el orden de (10^{-16}), valor que corresponde aproximadamente al límite de precisión numérica de la representación de doble precisión (Float64). Esta diferencia no representa una discrepancia algebraica entre ambas formulaciones, sino el efecto acumulado del redondeo asociado a las operaciones sucesivas de punto flotante durante la composición de las transformaciones. En consecuencia, la reconstrucción mediante cuaterniones duales reprodujo la misma transformación rígida obtenida mediante matrices homogéneas dentro del margen de precisión computacional disponible. Este resultado verificó numéricamente, sobre las transformaciones obtenidas del modelo del UR10 en RoboDK, la equivalencia formal entre ambas representaciones establecida en la Sección 2.3.

Resultados del Benchmark

El REPL de Julia presenta los siguientes resultados, para matrices de transformación homogéneas:

La figura 2 presenta la salida generada por la macro @benchmark de BenchmarkTools.jl durante la evaluación de la composición sucesiva de las seis matrices de transformación homogénea del UR10. El experimento fue ejecutado con 10 000 muestras y 120 evaluaciones por muestra, permitiendo caracterizar la distribución del tiempo de ejecución de la operación matricial bajo condiciones controladas.

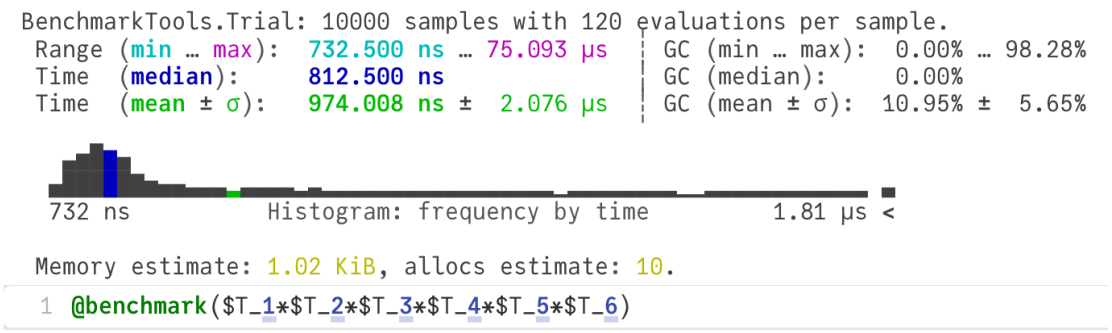
Los resultados muestran un tiempo mínimo de ejecución de 732.500 ns, un tiempo máximo de 75.093 μs y una mediana de 812.500 ns. El tiempo promedio obtenido fue de 974.008 ns ± 2.076 μs, evidenciando una dispersión asociada a variaciones propias del entorno de ejecución. El histograma generado por la herramienta muestra una mayor concentración de mediciones en los intervalos inferiores de tiempo,

mientras que una menor cantidad de ejecuciones presentó valores superiores, reflejados en la cola extendida de la distribución.

En relación con la gestión de memoria, la operación mediante matrices homogéneas presentó una estimación de 1.02 KiB de memoria utilizada y 10 asignaciones por ejecución. Además, se observó una participación del recolector de basura con un promedio de 10.95 % ± 5.65 %, alcanzando valores máximos de 98.28 % en algunas muestras. Estos resultados evidencian el comportamiento computacional de la implementación matricial y constituyen la referencia experimental para la comparación posterior con la representación mediante cuaterniones duales unitario.

Figura 2

Imagen de salida de la macro @benchmark para matrices homogéneas

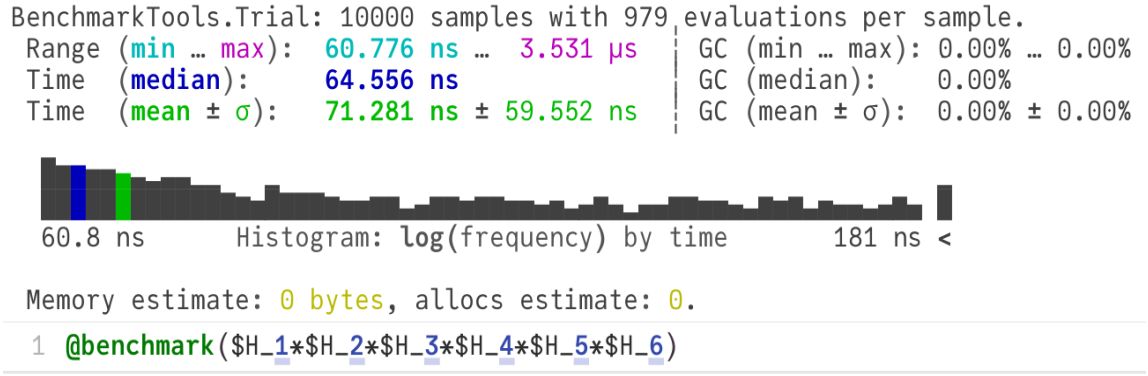


Para cuaterniones duales unitarios:

La figura 3 presenta la salida generada por la macro @benchmark de BenchmarkTools.jl durante la evaluación de la composición sucesiva de los seis cuaterniones duales unitarios correspondientes al modelo cinemático del UR10. La prueba se ejecutó con 10 000 muestras y 979 evaluaciones por muestra, utilizando la expresión de composición $H_1 * H_2 * H_3 * H_4 * H_5 * H_6$. Esta configuración permitió obtener una caracterización estadística del comportamiento computacional de la representación dual bajo condiciones experimentales controladas.

Figura 3

Imagen de salida de la macro @benchmark para cuaterniones duales



Los resultados muestran un tiempo mínimo de ejecución de 60.776 ns, un tiempo máximo de 3.531 µs y una mediana de 64.556 ns. El tiempo promedio obtenido fue de 71.281 ns ± 59.552 ns, evidenciando una baja dispersión respecto al rango total observado. El histograma generado por BenchmarkTools.jl representa la distribución logarítmica de frecuencias por tiempo de ejecución,

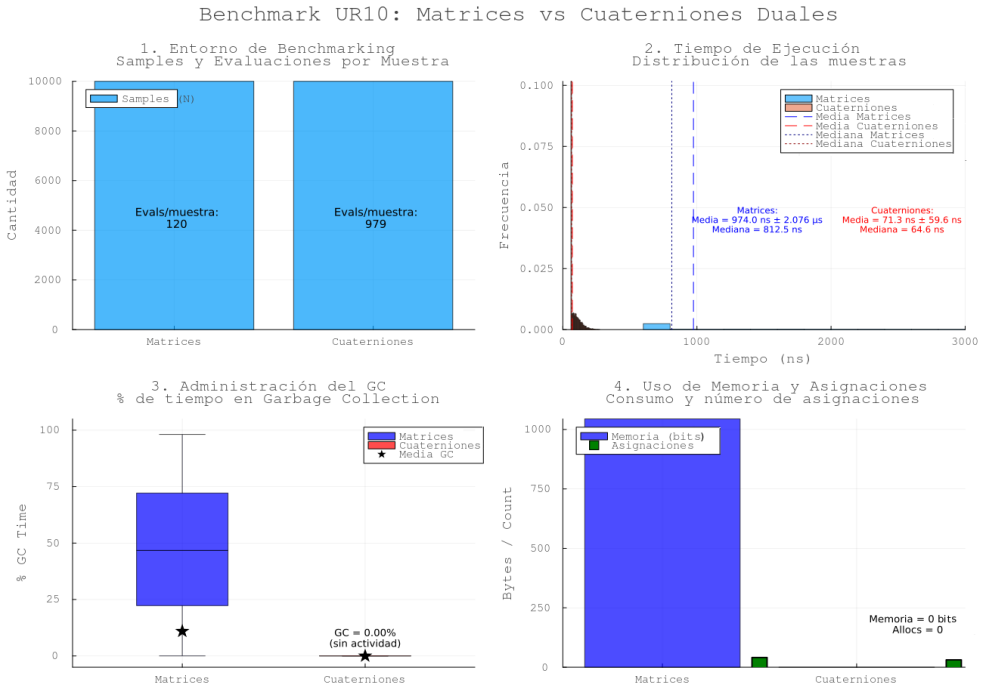
donde se aprecia una concentración predominante de mediciones alrededor de los valores más bajos, correspondientes a ejecuciones de corta duración.

En cuanto al comportamiento de memoria, la composición mediante cuaterniones duales unitarios registró una estimación de 0 bytes de memoria utilizada y 0 asignaciones durante la operación evaluada. De manera consistente, el recolector de basura no presentó actividad durante las mediciones, registrando valores de 0.00 % tanto en la media como en el rango máximo observado. Estos resultados muestran el comportamiento computacional de la implementación basada en cuaterniones duales y permiten establecer una comparación directa con la representación mediante matrices homogéneas presentada en la figura 2.

La figura 4 presenta una comparación global del comportamiento computacional de las dos representaciones evaluadas: matrices de transformación homogénea y cuaterniones duales unitarios. La figura integra cuatro indicadores principales obtenidos durante la ejecución del benchmark: número de evaluaciones realizadas por muestra, distribución del tiempo de ejecución, comportamiento del recolector de basura y consumo de memoria junto con el número de asignaciones generadas durante la composición de las seis transformaciones locales del UR10.

Figura 4

Imágenes comparativas de los resultados de la macro @benchmark



En el primer panel, correspondiente al entorno de benchmarking, se observa que BenchmarkTools.jl realizó 120 evaluaciones por muestra para la composición matricial y 979 evaluaciones por muestra para la composición mediante cuaterniones duales. Esta diferencia refleja el ajuste adaptativo del número de repeticiones efectuado por la herramienta, asociado con la menor duración individual de la operación basada en cuaterniones duales.

El segundo panel muestra la distribución de los tiempos de ejecución para ambas implementaciones. Las matrices homogéneas presentaron una media de 974.0 ns ± 2.076 µs y una mediana de 812.5 ns, mientras que los cuaterniones duales alcanzaron una media de 71.3 ns ± 59.6 ns

y una mediana de 64.6 ns. La separación entre las distribuciones evidencia una diferencia en el costo computacional de la operación de composición, con menores tiempos registrados para la representación mediante cuaterniones duales bajo las condiciones experimentales evaluadas.

El tercer panel representa el comportamiento del recolector de basura (*Garbage Collection*, GC). La implementación matricial mostró una mayor variabilidad en la intervención del GC, con valores medios superiores y presencia de muestras con elevada participación del recolector. En contraste, la composición mediante cuaterniones duales registró una actividad del GC de 0.00 %, indicando ausencia de intervención del mecanismo de recolección de memoria durante las ejecuciones analizadas.

El cuarto panel compara el consumo de memoria y el número de asignaciones generadas durante la ejecución. La composición mediante matrices homogéneas presentó un consumo aproximado de 1.02 KiB y 10 asignaciones, mientras que los cuaterniones duales registraron 0 bytes de memoria adicional y 0 asignaciones. En conjunto, la figura resume las diferencias observadas entre ambas implementaciones y complementa los resultados numéricos presentados en la tabla 5, evidenciando el comportamiento temporal y de gestión de memoria de cada representación dentro del entorno evaluado.

A partir de los valores obtenidos en la tabla 5 se calcularon razones comparativas entre ambas implementaciones con el propósito de cuantificar las diferencias relativas en tiempo de ejecución, comportamiento adaptativo del muestreo y gestión de memoria.

Tabla 5

Resultados comparativos del @benchmark para la composición de las seis transformaciones locales del UR10

Métrica	**Matrices homogéneas**	**Cuaterniones duales**
Muestras	10 000	10 000
Evaluaciones por muestra	120	979
Tiempo mínimo	732.500 ns	60.776 ns
Tiempo máximo	75.093 μs	3.531 μs
Tiempo mediano	812.500 ns	64.556 ns
Tiempo medio ± σ	974.008 ns ± 2.076 μs	71.281 ns ± 59.552 ns
GC medio ± σ	10.95 % ± 5.65 %	0.00 % ± 0.00 %
GC máximo	98.28 %	0.00 %
Memoria estimada	1.02 KiB	0 bytes
Asignaciones (*allocs*)	10	0

Nota. Los valores corresponden a la composición sucesiva de seis transformaciones locales del UR10 mediante matrices homogéneas y cuaterniones duales unitarios, ejecutada bajo las mismas condiciones de hardware, software y configuración experimental. Las métricas fueron obtenidas mediante la macro @benchmark de BenchmarkTools.jl con 10 000 muestras por formalismo.

En términos de velocidad de ejecución, la comparación basada en la mediana mostró que la composición mediante cuaterniones duales presentó un tiempo de ejecución aproximadamente 12.6 veces menor que la composición mediante matrices homogéneas (812.500/64.556). De forma

equivalente, considerando el tiempo medio, la diferencia fue de aproximadamente 13.7 veces (974.008/71.281). Estos resultados evidenciaron una reducción del costo temporal de la operación de composición bajo la implementación basada en cuaterniones duales en el entorno evaluado.

Respecto al muestreo adaptativo, BenchmarkTools.jl realizó 979 evaluaciones por muestra para la composición mediante cuaterniones duales y 120 evaluaciones por muestra para la representación matricial, equivalente a una relación aproximada de 8.2 veces más repeticiones. Este comportamiento se relacionó con la menor duración individual de la operación basada en cuaterniones duales, ya que el mecanismo adaptativo incrementó el número de evaluaciones requeridas para obtener mediciones estadísticamente representativas en operaciones de menor tiempo de ejecución.

En cuanto al consumo de memoria y asignaciones, la composición mediante cuaterniones duales registró 0 bytes de memoria estimada y 0 asignaciones durante las 10 000 muestras evaluadas. En contraste, la implementación mediante matrices homogéneas presentó un consumo de 1.02 KiB y 10 asignaciones por operación. Esta diferencia mostró un comportamiento distinto en la gestión de memoria entre ambas representaciones dentro del entorno de ejecución utilizado.

Finalmente, el análisis del recolector de basura (*Garbage Collector*, GC) mostró que la composición matricial presentó una participación media de 10.95 % ± 5.65 %, con valores máximos de hasta 98.28 % en determinadas muestras. Por otro lado, la implementación mediante cuaterniones duales registró una participación del GC de 0.00 % ± 0.00 %, sin actividad detectable durante las ejecuciones analizadas. Estos resultados fueron consistentes con las diferencias observadas en las asignaciones de memoria y permitieron complementar la evaluación del desempeño computacional de ambas representaciones.

4. Discusión

Costo computacional por operación elemental

El comportamiento adaptativo de BenchmarkTools.jl proporcionó evidencia sobre las diferencias de velocidad entre ambos formalismos evaluados. Para obtener una medición estadísticamente representativa, el framework ejecutó 979 evaluaciones por muestra en la composición mediante cuaterniones duales, frente a 120 evaluaciones por muestra en la composición matricial. Esta diferencia estuvo asociada con la menor duración individual de la operación basada en cuaterniones duales, debido a que el mecanismo adaptativo incrementó el número de repeticiones necesarias para caracterizar operaciones de corta duración con mayor precisión.

La reducción del tiempo de ejecución observada es coherente con las propiedades estructurales de los cuaterniones duales, los cuales permiten representar movimientos rígidos mediante una formulación compacta que integra rotación y traslación en una única entidad algebraica (Qi et al., 2021; Cohen & Shoham, 2020). Esta característica ha favorecido su incorporación en aplicaciones de cinemática directa, control robótico y planificación de movimientos, donde se busca reducir la complejidad computacional asociada con las representaciones tradicionales (Adorno & Marinho, 2021; Dantam, 2021).

Los resultados obtenidos mostraron que la composición mediante cuaterniones duales presentó menores tiempos de ejecución respecto a la representación matricial bajo las condiciones experimentales establecidas. Este comportamiento puede relacionarse tanto con la estructura matemática de la representación como con la implementación computacional utilizada en Julia, donde el tipo compuesto DualQuaternion{T} permitió trabajar con una estructura de tamaño fijo, evitando procesos adicionales asociados con la manipulación de matrices dinámicas. Estudios recientes han empleado representaciones

basadas en cuaterniones duales para resolver problemas cinemáticos y de control debido a su capacidad para manejar transformaciones rígidas evitando redundancias presentes en otras parametrizaciones (Condurache & Popa, 2023; Giribet et al., 2025).

Estabilidad y predictibilidad temporal

Además de la reducción en el tiempo medio de ejecución, la composición mediante cuaterniones duales mostró un comportamiento temporal diferente respecto a las matrices homogéneas. La desviación estándar registrada para los cuaterniones duales fue de 59.552 ns, mientras que para las matrices alcanzó 2.076 μs, evidenciando una mayor dispersión temporal en la implementación matricial. Esta diferencia indica que, dentro del entorno evaluado, la operación basada en matrices presentó una mayor variabilidad entre ejecuciones consecutivas.

En la implementación matricial se observó un valor máximo de 75.093 μs, aproximadamente 100 veces superior a su mediana de 812.500 ns. En sistemas robóticos donde los cálculos cinemáticos forman parte de ciclos de control de alta frecuencia, las variaciones temporales pueden afectar la predictibilidad del proceso computacional y generar fluctuaciones en los tiempos de actualización del controlador (Wittmann et al., 2022). Por esta razón, la estabilidad temporal constituye un criterio relevante además del tiempo promedio de ejecución.

En contraste, la implementación mediante cuaterniones duales presentó un rango comprendido entre 60.776 ns y 3.531 μs, con una concentración predominante de muestras en los intervalos inferiores observados en el histograma correspondiente. Este comportamiento coincide con investigaciones que han utilizado cuaterniones duales para representar movimientos rígidos en sistemas robóticos, destacando su capacidad para mantener formulaciones compactas y adecuadas para aplicaciones con restricciones computacionales (Alattar & Kormushev, 2020; Zivković et al., 2022). No obstante, estos resultados corresponden específicamente a la implementación desarrollada y al entorno computacional utilizado.

Presión sobre el recolector de basura

Una de las diferencias más relevantes identificadas en el benchmark correspondió al comportamiento asociado con la gestión de memoria. La representación mediante cuaterniones duales registró 0 bytes de memoria estimada y 0 asignaciones, mientras que la implementación matricial presentó 1.02 KiB de memoria utilizada y 10 asignaciones por operación. Esta diferencia refleja la interacción entre las estructuras de datos utilizadas y el modelo de administración de memoria del lenguaje Julia.

El tipo DualQuaternion{T} empleado en este estudio corresponde a una estructura compuesta de tamaño fijo, mientras que Matrix {Float64} representa un arreglo cuya gestión requiere memoria dinámica. En consecuencia, la composición matricial generó objetos temporales asociados con las operaciones de multiplicación, mientras que la implementación dual evitó asignaciones detectables durante la operación evaluada. Este comportamiento coincide con estudios sobre optimización computacional, donde la reducción de asignaciones temporales y una gestión eficiente de memoria constituyen factores relevantes para mejorar el rendimiento de aplicaciones numéricas (Klepl et al., 2024; Santos et al., 2024).

La ausencia de actividad del GC en la representación dual frente al 10.95 % ± 5.65 % de participación media registrada en matrices homogéneas constituye un hallazgo relevante dentro del benchmark realizado. Sin embargo, este resultado debe interpretarse como una consecuencia de la implementación específica en Julia y de la estructura utilizada, no como una propiedad exclusiva del formalismo

matemático de los cuaterniones duales independientemente del lenguaje de programación o del compilador empleado.

Consistencia con el marco teórico

Los resultados obtenidos demostraron que las diferencias de rendimiento no implicaron pérdida de exactitud en la representación de la transformación rígida. Como se presentó en la Sección 4.1, ambas implementaciones produjeron la misma pose final de la brida del UR10 con diferencias del orden de la precisión numérica de Float64, atribuibles al redondeo acumulado durante las operaciones de punto flotante. Este comportamiento confirma que la comparación realizada evaluó exclusivamente las características computacionales de cada representación y no diferencias asociadas con la solución cinemática obtenida.

La equivalencia observada entre matrices homogéneas y cuaterniones duales fue coherente con el fundamento matemático descrito en la literatura, donde ambas estructuras representan transformaciones rígidas pertenecientes al grupo $SE(3)$mediante formulaciones algebraicamente equivalentes (Solà et al., 2018; Xu & Halse, 2016; Zeng et al., 2024). Por tanto, el benchmark permitió aislar los efectos relacionados con tiempo de ejecución, asignaciones de memoria y actividad del recolector de basura, manteniendo constante la transformación geométrica representada.

Los resultados obtenidos complementan trabajos previos enfocados en cinemática robótica basada en cuaterniones duales, como los desarrollados para modelado, control, calibración y transformación de coordenadas (Dantam, 2021; Huang et al., 2026; Diprasetya et al., 2025). En este sentido, el presente estudio aporta una evaluación experimental centrada específicamente en el costo computacional de la composición de transformaciones rígidas del UR10 mediante una comparación controlada entre matrices homogéneas y cuaterniones duales unitarios.

Limitaciones del Estudio

El presente estudio presenta limitaciones que delimitan el alcance de sus resultados e interpretación. En primer lugar, las mediciones obtenidas corresponden a una única configuración de hardware y software, conformada por un procesador AMD Ryzen 7 de 2.00 GHz, 8 GB de RAM y arquitectura de 64 bits, junto con implementaciones específicas desarrolladas en Julia mediante Quaternions.jl, ForwardDiff.jl y BenchmarkTools.jl. En consecuencia, los valores registrados representan el comportamiento de estas implementaciones bajo dicho entorno experimental y no pueden extrapolarse directamente a otras arquitecturas computacionales, lenguajes de programación o versiones diferentes de las librerías utilizadas. Por tanto, las diferencias observadas deben interpretarse como características de la implementación evaluada y no como propiedades absolutas de los formalismos matemáticos comparados.

En segundo lugar, el análisis se restringió a un único manipulador robótico, el UR10 de seis grados de libertad, y exclusivamente al problema de cinemática directa. No se evaluaron escenarios adicionales como cinemática inversa, control dinámico, planificación de trayectorias o manipuladores con estructuras cinemáticas diferentes. Aunque los resultados permiten caracterizar el comportamiento computacional de ambas representaciones en el caso estudiado, su extensión hacia otros robots o problemas robóticos requiere evaluaciones adicionales bajo condiciones específicas.

En tercer lugar, los datos articulares empleados fueron obtenidos mediante simulación en RoboDK y no mediante adquisición directa de un robot físico en operación. El propósito del estudio fue comparar el costo computacional de dos representaciones matemáticas bajo condiciones controladas, por lo que no se abordó la validación experimental del modelo cinemático del simulador frente al comportamiento real del UR10.

En consecuencia, posibles diferencias derivadas de tolerancias mecánicas, sensores, controladores industriales o incertidumbres del sistema físico se encuentran fuera del alcance de esta investigación.

El protocolo de benchmark aplicado permitió obtener mediciones repetibles y estadísticamente consistentes; sin embargo, las pruebas fueron ejecutadas en un entorno de escritorio convencional y no sobre sistemas operativos de tiempo real ni plataformas embebidas utilizadas habitualmente en controladores robóticos industriales. Por esta razón, los tiempos reportados representan el costo computacional de las operaciones evaluadas dentro del entorno utilizado, pero no necesariamente equivalen al comportamiento que podría observarse dentro de un ciclo de control físico desplegado en hardware especializado.

El estudio comparó únicamente matrices de transformación homogénea y cuaterniones duales unitarios implementados en el lenguaje Julia. No se incluyeron otras formulaciones matemáticas, como representaciones basadas en teoría de tornillos, álgebra de Lie o parametrizaciones alternativas, ni implementaciones desarrolladas en otros lenguajes o paradigmas computacionales como C++, Rust o arquitecturas paralelas mediante GPU. Estas alternativas representan líneas potenciales para investigaciones futuras orientadas a ampliar la comparación del rendimiento computacional entre diferentes enfoques para la representación y composición de movimientos rígidos.

5. Conclusiones

En el escenario experimental definido, con hardware, software y configuración controlados, datos articulares obtenidos desde la estación del UR10 en RoboDK y composición de seis transformaciones locales mediante implementaciones desarrolladas en Julia, los cuaterniones duales unitarios presentaron un menor costo computacional respecto a las matrices de transformación homogénea 4×4 evaluadas en este estudio. La composición basada en cuaterniones duales alcanzó una reducción aproximada de 12.6 veces en el tiempo mediano de ejecución, junto con una menor variabilidad temporal durante las mediciones. Además, no registró asignaciones de memoria ni actividad del recolector de basura, mientras que la implementación matricial presentó 1.02 KiB de memoria estimada, 10 asignaciones y una participación media del GC de 10.95 %.

Los resultados obtenidos permitieron comprobar experimentalmente que las ventajas estructurales atribuidas a los cuaterniones duales en la literatura pueden reflejarse en diferencias cuantificables de rendimiento computacional cuando se implementan bajo condiciones controladas. En particular, la representación dual mostró un comportamiento más eficiente en la operación específica de composición de transformaciones rígidas analizada. Sin embargo, estos resultados corresponden exclusivamente a las implementaciones realizadas en Julia mediante Quaternions.jl, ForwardDiff.jl y BenchmarkTools.jl, por lo que no representan una comparación absoluta entre los formalismos matemáticos independientemente del entorno computacional.

La verificación numérica realizada previamente permitió establecer que ambas representaciones generaron la misma pose final de la brida del UR10, con diferencias del orden de la precisión de máquina de doble precisión (Float64). Este resultado confirmó que las diferencias observadas durante el benchmark estuvieron asociadas al costo computacional de cada implementación y no a discrepancias en la solución cinemática obtenida. De esta manera, el estudio logró comparar el rendimiento de dos representaciones algebraicamente equivalentes manteniendo constante la transformación rígida evaluada.

En consecuencia, este trabajo aporta evidencia experimental sobre el comportamiento computacional de matrices homogéneas y cuaterniones duales unitarios en la composición de la cinemática directa de un manipulador industrial específico. Los resultados obtenidos

proporcionan una referencia reproducible para futuras evaluaciones que incorporen diferentes arquitecturas de hardware, lenguajes de programación, plataformas embebidas o problemas robóticos adicionales, con el propósito de ampliar la caracterización del rendimiento de estas representaciones en otros escenarios de aplicación.

Referencias

Adorno, B., & Marinho, M. (2021). DQ Robotics: A library for robot modeling and control. *IEEE Robotics & Automation Magazine, 28*(3), 102–116. https://doi.org/10.1109/MRA.2020.2997920

Alattar, A., & Kormushev, P. (2020). Kinematic-model-free orientation control for robot manipulation using locally weighted dual quaternions. *Robotics, 9*(4), Article 76. https://doi.org/10.3390/robotics9040076

Cargua, M. (2026). *Benchmark entre matrices homogéneas y cuaterniones duales* (Version 0.1) [Computer software]. GitHub. https://github.com/marleyns/benchmark_dual_quat.git

Cohen, A., & Shoham, M. (2020). Hyper dual quaternions representation of rigid bodies kinematics. *Mechanism and Machine Theory, 150*, Article 103861. https://doi.org/10.1016/j.mechmachtheory.2020.103861

Condurache, D., & Popa, I. (2023). A minimal parameterization of rigid body displacement and motion using a higher-order Cayley map by dual quaternions. *Symmetry, 15*(11), Article 2011. https://doi.org/10.3390/sym15112011

Dantam, N. (2021). Robust and efficient forward, differential, and inverse kinematics using dual quaternions. *The International Journal of Robotics Research, 40*(10–11), 1087–1105. https://doi.org/10.1177/0278364920931948

Diprasetya, M., Pöppelbaum, J., & Schwung, A. (2025). KineNN: Kinematic neural network for inverse model policy based on homogeneous transformation matrix and dual quaternion. *Robotics and Computer-Integrated Manufacturing, 94*, Article 102945. https://doi.org/10.1016/j.rcim.2024.102945

Giribet, J., Ghersin, A., Mas, I., Neves Marciano, H., Khede Dourado Villa, D., & Sarcinelli-Filho, M. (2025). *Adaptive multirobot virtual structure control using dual quaternions*. arXiv. https://doi.org/10.48550/arXiv.2504.05560

Huang, T., Yang, B., Li, B., Li, W., Li, H., Li, W., & Liu, Y.-H. (2026). *A unified calibration framework for coordinate and kinematic parameters in dual-arm robots* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2603.14809

Klepl, J., Šmelko, A., Rozsypal, L., & Kruliš, M. (2024). Abstractions for C++ code optimizations in parallel high-performance applications. *Parallel Computing, 121*, Article 103096. https://doi.org/10.1016/j.parco.2024.103096

Liu, Q., Yang, D., Hao, W., & Wei, Y. (2018). Research on kinematic modeling and analysis methods of UR robot. In *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)* (pp. 159–164). IEEE. https://doi.org/10.1109/ITOEC.2018.8740681

Menon, A., Prakash, R., & Behera, L. (2019). *Adaptive critic based optimal kinematic control for a robot manipulator* [Preprint]. arXiv. https://arxiv.org/abs/1908.02077

Nekoo, S., Acosta, J., & Ollero, A. (2022). Quaternion-based state-dependent differential Riccati equation for quadrotor drones: Regulation control problem in aerobatic flight. *Robotica, 40*(9), 3120–3135. https://doi.org/10.1017/S0263574722000091

Petrone, V., Ferrentino, E., & Chiacchio, P. (2025). The dynamic model of the UR10 robot and its ROS2 integration. *IEEE Transactions on Industrial Informatics, 21*(5), 3828–3838. https://doi.org/10.1109/TII.2025.3534415

Pollák, M., & Goryl, K. (2023). Simulation design and measurement of welding robot repeatability utilizing the contact measurement method. *Machines, 11*(7), Article 734. https://doi.org/10.3390/machines11070734

Qi, L., Ling, C., & Yan, H. (2021). *Dual quaternions and dual quaternion vectors* [Preprint]. arXiv. https://arxiv.org/abs/2111.04491

Roesch, E., Greener, J. G., MacLean, A. L., Nassar, H., Rackauckas, C., Holy, T. E., & Stumpf, M. P. H. (2023). Julia for biologists. *Nature Methods, 20*(5), 655–664. https://doi.org/10.1038/s41592-023-01832-z

Santos, F., Carro, L., Vella, F., & Rech, P. (2024). Assessing the impact of compiler optimizations on GPUs reliability. *ACM Transactions on Architecture and Code Optimization, 21*(2), Article 26. https://doi.org/10.1145/3638249

Solà, J., Deray, J., & Atchuthan, D. (2018). *A micro Lie theory for state estimation in robotics* [Preprint]. arXiv. https://arxiv.org/abs/1812.01537

Wittmann, J., Kist, A., & Rixen, D. (2022). Real-time predictive kinematics control of redundancy: A benchmark of optimal control approaches. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 11759–11766). IEEE. https://doi.org/10.1109/IROS47612.2022.9981675

Xu, J., & Halse, K. H. (2016). *Dual quaternion variational integrator for rigid body dynamic simulation* [Preprint]. arXiv. https://arxiv.org/abs/1611.00616

Yaqub, Z., & Schröcker, H. (2025). Rational motions of minimal quaternionic degree with prescribed line trajectories. *Mechanism and Machine Theory, 215*, Article 106182. https://doi.org/10.1016/j.mechmachtheory.2025.106182

Zeng, H., Wang, Z., Li, J., Li, S., Wang, J., & Li, X. (2024). Dual-quaternion-based iterative algorithm of the three dimensional coordinate transformation. *Earth, Planets and Space, 76*, Article 20. https://doi.org/10.1186/s40623-024-01967-z

Zhao, J., Zhu, L., & So, A. (2026). *Dual quaternion SE(3) synchronization with recovery guarantees* [Preprint]. arXiv. https://arxiv.org/abs/2602.00324

Živković, N., Vidaković, J., Mitrović, S., & Lazarević, M. (2022). Implementation of dual quaternion-based robot forward kinematics algorithm in ROS. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. https://doi.org/10.1109/MECO55406.2022.9797160

Transparencia

Conflicto de interés

Los autores declaran que no existen conflictos de interés de naturaleza alguna como parte de la presente investigación.

Fuente de financiamiento

Los autores financiaron completamente la investigación.

Contribución de autoría

Marlon Stalyn Cargua Cando: Conceptualización, metodología, software, validación, análisis formal, investigación, gestión de datos, visualización, redacción - preparación del borrador original, redacción - revisión y edición, financiamiento, administración del proyecto, recursos.

Guillermo Edvin Machado Sotomayor : Conceptualización, metodología, software, validación, análisis formal, investigación, gestión de datos, visualización, redacción - revisión y edición, financiamiento, recursos, supervisión.

Los autores contribuyeron activamente en el análisis de los resultados, revisión y aprobación del manuscrito final.

Anexo

Los scripts de Python y Julia utilizados en esta investigación están disponibles públicamente en portal GitHub (https://github.com/marleyns/benchmark_dual_quat.git) para permitir la verificación y reproducción de los resultados.