Informaticae Abstracta. ISSN: 3061-8533
Año 2026, Vol. 4, No. 1, pp. 99-123
ENE | JUN 2026
doi: https://doi.org/10.36677/ia.v4n1.28941

Sistema de coordenadas SPR: optimización algebraica y análisis CORDIC para control robótico

Salas-García, J.*; Hernández-Servín, J. A.; Vilchis-González, A. H.; Ávila-Vilchis, J. C.

Información del Artículo

Recibido: 15 de febrero, 2026
Aceptado: 20 de mayo, 2026

Palabras clave: Sistema de coordenadas SPR, CORDIC, control robótico, optimización algebraica, microcontrolador ESP32

Resumen

Este artículo presenta el Sistema de Coordenadas de Planos Rotatorios (SPR), una forma alternativa de describir la posición de un punto en el espacio mediante un radio y dos ángulos. Está pensado para robots y sistemas de posicionamiento multieje que necesitan convertir coordenadas miles de veces por segundo, con frecuencia en procesadores pequeños. Se desarrolla la base geométrica del sistema y se comparan sus conversiones (de coordenadas SPR a cartesianas y viceversa) con las de los sistemas cilíndrico y esférico, midiendo tiempo de cómputo y estabilidad numérica sobre diez millones de puntos en dos plataformas: un procesador de escritorio x86_64 y un microcontrolador ESP32. En la conversión inversa, que es la que un controlador ejecuta con mayor frecuencia, SPR resultó el más rápido de los sistemas de dos ángulos en todas las plataformas evaluadas: entre 1.15 y 1.5 veces más veloz que el esférico, según la plataforma y el escenario. Además, su precisión se mantiene constante cerca de los polos, la región donde el sistema esférico pierde exactitud. Para procesadores sin unidad de punto flotante se propone una versión de SPR basada en el algoritmo CORDIC con aritmética entera, validada en el ESP32: es 4.8 veces más rápida que cualquier alternativa de doble precisión en ese chip y su tiempo de ejecución es prácticamente constante (variación menor al 0.2 %), una propiedad útil en sistemas de tiempo real. En conjunto, SPR permite sostener lazos de control por encima de 1 kHz en hardware de bajo costo. El trabajo incluye un simulador 3D interactivo y el código necesario para reproducir todas las mediciones.

1. Introducción

En la robótica contemporánea, la precisión y la velocidad de respuesta en tiempo real son requisitos fundamentales para el control de plataformas móviles y manipuladores paralelos. No obstante, un obstáculo persistente es la latencia computacional introducida por las transformaciones de coordenadas complejas en lazos de control de alta frecuencia (superiores a 1 kHz) [1]. Los sistemas convencionales, como el esférico y el cilíndrico, dependen fuertemente de funciones trigonométricas inversas (`atan2`, `acos`), las cuales representan una carga significativa para los procesadores embebidos con recursos limitados y unidades de punto flotante de bajo rendimiento [2].

A esta limitación computacional se suma el problema de las singularidades de representación, como el bloqueo de cardán (*gimbal lock*), que degrada la estabilidad numérica en regiones críticas del espacio de trabajo [3]. Si bien se han propuesto soluciones basadas en cuaterniones o matrices de rotación, estas a menudo incrementan la complejidad algebraica y la redundancia de datos.

Ante este escenario, surge la necesidad de modelos de representación que minimicen el uso de funciones trascendentales costosas, priorizando estructuras algebraicas directas que faciliten la inversión del modelo en tiempo real. El presente trabajo formaliza el sistema de coordenadas SPR y propone una estrategia de optimización dual: por un lado, una profunda refactorización algebraica basada en lógica binaria, y por otro, la fundamentación de una variante algorítmica iterativa basada en CORDIC con ejecución *branchless*. A través de un análisis numérico masivo, se demuestra empíricamente que las formulaciones SPR reducen la

latencia de la transformación inversa frente al modelo esférico en todas las plataformas evaluadas —con una magnitud dependiente de la biblioteca matemática de cada arquitectura—, al mismo tiempo que eliminan las inestabilidades numéricas en zonas de singularidad axial. Por su parte, la variante CORDIC se formula y se valida empíricamente sobre un microcontrolador ESP32, delimitando los regímenes de precisión (doble emulada, simple con FPU y aritmética entera de punto fijo) en los que cada variante resulta óptima.

2. Disponibilidad de Código y Datos

Para asegurar la reproducibilidad de los resultados presentados, el código fuente completo se encuentra alojado en el repositorio de GitHub: https://github.com/JavierSalasGarcia/sistemaSPR. El ecosistema de software se compone de tres herramientas esenciales:

1. `benchmark.c`: Motor de pruebas masivas que ejecuta 10 millones de transformaciones para medir latencia y estabilidad numérica (RMSE).
2. `analysis_and_plots.py`: Script de automatización que gestiona la compilación, ejecución del benchmark y generación de las figuras estadísticas presentadas en este artículo.
3. `simulator_spr.py`: Herramienta de visualización interactiva 3D que permite validar cualitativamente el comportamiento del sistema de planos rotatorios.
4. `benchmark_esp32/`: Firmware ESP-IDF que reproduce el benchmark sobre un microcontrolador ESP32 real en tres regímenes de precisión (doble emulada, simple con FPU y entero de punto fijo), junto con los scripts de captura por puerto serie y de generación de las figuras correspondientes (`host/`).

Para la reproducción íntegra de los datos del artículo, basta con ejecutar el comando `python analysis_and_plots.py` dentro del directorio raíz. Este proceso automatiza la compilación del código en lenguaje C nativo, la ejecución de las pruebas bajo escenarios críticos y la generación de las figuras presentadas.

3. Fundamentación Teórica y Formulación Matemática

Ecuaciones Fundamentales

El sistema SPR define la posición de un punto en el espacio tridimensional, denotado como $P(r_1, \theta_1, \theta_2)$, a través de la intersección de tres superficies continuas. El concepto base del sistema se ilustra mediante trayectorias paramétricas sobre una esfera, como se muestra en la Figura 1.

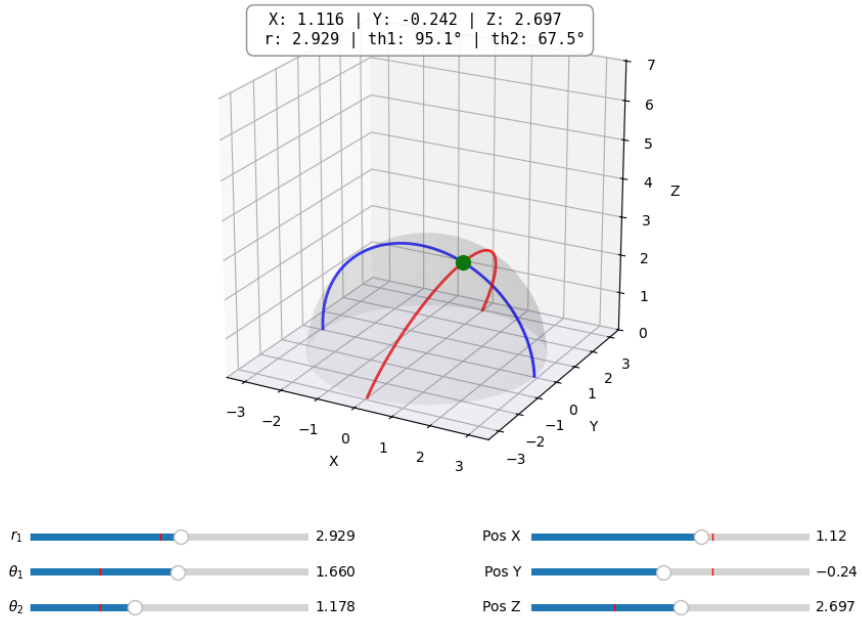


Figura 1: Representación gráfica del sistema SPR obtenida mediante el simulador interactivo desarrollado. Las trazas azules (θ_1 fijo) y rojas (θ_2 fijo) ilustran la ortogonalidad y continuidad del sistema. Esta herramienta, disponible en el repositorio adjunto, permite la validación visual del modelo y la comprobación de la reversibilidad de las ecuaciones.

La primera superficie corresponde a una esfera de radio r_1, expresada en coordenadas cartesianas (x, y, z) mediante la Ecuación 1:

$$x^2 + y^2 + z^2 = r_1^2 \tag{1}$$

Las superficies adicionales consisten en planos que se originan en el centro de coordenadas. Estos emplean relaciones tangenciales para determinar la orientación espacial, lo que deriva en las Ecuaciones 2 y 3:

$$z = y \tan(\theta_1) \tag{2}$$

$$z = x \tan(\theta_2) \tag{3}$$

Al resolver este sistema de ecuaciones, se obtiene la "Transformación Directa", la cual permite convertir las coordenadas SPR al sistema cartesiano (x, y, z). Estas relaciones se presentan en la Ecuación 4:

$$x = \frac{r_1 \cos(\theta_2) \sin(\theta_1)}{\sqrt{1 - \cos^2(\theta_1)\cos^2(\theta_2)}} \quad , \quad y = \frac{r_1 \cos(\theta_1) \sin(\theta_2)}{\sqrt{1 - \cos^2(\theta_1)\cos^2(\theta_2)}} \quad , \quad z = \frac{r_1 \sin(\theta_1) \sin(\theta_2)}{\sqrt{1 - \cos^2(\theta_1)\cos^2(\theta_2)}} \tag{4}$$

Para asegurar la validez del método, el radio debe mantener valores no negativos y los rangos angulares deben cubrir la envolvente esférica. Esta configuración garantiza posiciones unívocas y evita discontinuidades presentes en otros sistemas de representación.

Matriz Jacobiana y Tensor Métrico

La matriz Jacobiana (J) se construye a partir de las derivadas parciales de las ecuaciones de transformación. Para diagnosticar la estabilidad geométrica del sistema, se formula su determinante conforme a la Ecuación 5:

$$\det(J) = \frac{r_1^2}{D^3 \sin(\theta_1) \cos^2(\theta_2)} \tag{5}$$

En esta expresión, D representa el divisor polinómico $\sqrt{1 + \tan^2(\theta_2)/\tan^2(\theta_1) + \tan^2(\theta_2)}$, cuya evaluación excluye divisiones por cero dentro de los rangos angulares válidos. Asimismo, es posible derivar el Tensor Métrico ($g_{ij} = J^T J$), componente esencial para cálculos volumétricos y métricas de distancia en aplicaciones avanzadas [4].

Análisis de Singularidades Geométricas

En los sistemas de coordenadas espaciales, se presentan puntos críticos o singularidades cuando los ángulos alcanzan orientaciones ortogonales (por ejemplo, $\theta_1 = 0$ o $\theta_2 = \pi/2$). El cálculo directo en estos puntos puede generar indeterminaciones numéricas debido a la proximidad del denominador a cero.

No obstante, mediante la aplicación de límites matemáticos, el sistema SPR resuelve estas condiciones de forma natural. Por ejemplo, al evaluar el comportamiento cuando el ángulo θ_1 tiende a cero, la posición converge a $(0, r_1, 0)^T$. Este tratamiento asintótico es fundamental para prevenir fallos computacionales comunes, como el bloqueo de ejes (“Gimbal Lock”) o errores de ejecución, que son frecuentes en sistemas de coordenadas esféricas convencionales [5].

4. Formalización y Derivación Analítica Detallada

Resolución del Sistema de Intersección

La localización exacta de un punto P se determina resolviendo simultáneamente las ecuaciones de la superficie esférica y los planos rotatorios. A partir de las identidades de los planos $z = y\tan(\theta_1)$ y $z = x\tan(\theta_2)$, se establece la relación de proporcionalidad:

$$y = x\frac{\tan(\theta_2)}{\tan(\theta_1)} \tag{6}$$

Al sustituir tanto y como $z = x\tan(\theta_2)$ en la ecuación de la esfera $x^2 + y^2 + z^2 = r_1^2$, se obtiene la expresión factorizada:

$$x^2 \left(1 + \frac{\tan^2(\theta_2)}{\tan^2(\theta_1)} + \tan^2(\theta_2)\right) = r_1^2 \tag{7}$$

De esta derivación emanan las soluciones para las coordenadas cartesianas presentadas anteriormente (Ecuación 4). Este procedimiento asegura que el punto resultante pertenezca con precisión a las tres trayectorias geométricas que definen el sistema.

Ecuaciones de las Trazas Paramétricas

Las trazas sobre la superficie esférica representan el recorrido de un punto cuando uno de los parámetros angulares permanece constante. Estas curvas son esenciales para la visualización y el mapeo de superficies. Para una traza con θ_1 constante, las ecuaciones paramétricas en función de un parámetro angular $t \in [0, 2\pi]$ se expresan como:

$$x = r_1 \cos(t) \quad , \quad y = r_1 \cos(\theta_1) \sin(t) \quad , \quad z = r_1 \cos(\theta_1) \sin(t) \tan(\theta_1) \tag{8}$$

De manera simétrica, para una traza donde θ_2 es constante, se obtiene:

$$x = r_1 \cos(\theta_2) \sin(t) \quad , \quad y = r_1 \cos(t) \quad , \quad z = r_1 \cos(\theta_2) \sin(t) \tan(\theta_2) \tag{9}$$

Estas trayectorias configuran la retícula característica del sistema SPR, permitiendo un barrido ordenado del espacio esférico.

Componentes Detalladas del Tensor Métrico

Para un análisis profundo de la geometría diferencial del sistema, se derivan los elementos del tensor métrico $g_{ij} = J^T J$. Al particionar la matriz Jacobiana J mediante derivadas parciales respecto a r_1, θ_1, θ_2, se identifican componentes clave que dictan la escala local del sistema. El elemento radial se mantiene unitario ($g_{11} = 1$), lo que confirma que el radio r_1 conserva su magnitud métrica natural.

Los elementos angulares más representativos, que determinan la longitud infinitesimal de arco ds^2, se derivan como:

$$g_{22} = \frac{r_1^2 \tan^4(\theta_2) + r_1^2 \tan^2(\theta_2) D^4 \cos^4(\theta_1) + r_1^2 \tan^6(\theta_2)}{D^6 \sin^4(\theta_1) \cos^4(\theta_1)} \tag{10}$$

$$g_{33} = \frac{r_1^2 (1 + \tan^2(\theta_2)/\tan^2(\theta_1))^2 (1 + \tan^2(\theta_2)) + r_1^2 D^4}{D^6 \cos^4(\theta_2)} \tag{11}$$

Donde D es el divisor polinómico definido previamente. Estas expresiones permiten calcular con precisión elementos de área dA y volumen dV dentro del dominio SPR, facilitando su aplicación en simulaciones de física computacional y cálculo integral espacial.

5. Implementación y Optimización Computacional

En esta sección se detallan las estrategias algorítmicas empleadas para transformar las ecuaciones teóricas en rutinas de computación de alto rendimiento, minimizando el costo operativo en transformaciones directas e inversas.

Refactorización de la Transformación Directa

La formulación original del sistema SPR (Ecuación 2) depende intrínsecamente de la función tangente, lo que introduce una dependencia crítica de funciones inversas y raíces cuadradas anidadas para recuperar las coordenadas x e y a partir de z. Tradicionalmente, la

conversión se realizaba mediante el Algoritmo 1, el cual requiere el cálculo de dos tangentes y raíces cuadradas pesadas.

Algorithm 1: Transformación Directa Tradicional (basada en tangentes)

Entrada: $coord = (r_1, \theta_1, \theta_2)$
Salida : (x, y, z)
$t_1 \leftarrow \tan(coord.\theta_1)$;
$t_2 \leftarrow \tan(coord.\theta_2)$;
$it1_2 \leftarrow 1{,}0/(t_1^2 + 10^{-25})$;
$it2_2 \leftarrow 1{,}0/(t_2^2 + 10^{-25})$;
$z \leftarrow coord.r_1/\sqrt{1{,}0 + it1_2 + it2_2}$;
$x \leftarrow z/t_2$;
$y \leftarrow z/t_1$;
return (x, y, z)

En el Algoritmo 1, las variables t_1 y t_2 almacenan las tangentes de los parámetros angulares, mientras que $it1_2$ e $it2_2$ representan sus inversas al cuadrado, regularizadas con un término constante de 10^{-25} para evitar excepciones por división entre cero en caso de ángulos ortogonales.

Como complemento, se derivó una variante puramente algebraica basada en identidades trigonométricas fundamentales ($\sin, \cos$), integrada en el Algoritmo 2. Esta formulación expresa la transformación directa en términos de proyecciones ortogonales y es la que se emplea en el simulador interactivo y en la ruta de cómputo optimizada. Cabe destacar que, en términos de latencia, la versión tradicional basada en tangentes (Algoritmo 1) requiere únicamente dos evaluaciones de $\tan$ frente a las cuatro de $\sin$ / $\cos$ de la versión algebraica, por lo que es la más rápida de las dos formulaciones SPR para la transformación directa, con un costo comparable al del modelo esférico (véase la Figura 4). La formulación sin/cos (Algoritmo 2) es matemáticamente equivalente y resulta conveniente para su uso conjunto con la transformación inversa optimizada (Algoritmo 4).

Algorithm 2: Transformación Directa Optimizada (Identidades $\sin$ / $\cos$)

Entrada: $coord = (r_1, \theta_1, \theta_2)$
Salida : (x, y, z)
$s_1 \leftarrow \sin(coord.\theta_1), \quad c_1 \leftarrow \cos(coord.\theta_1)$;
$s_2 \leftarrow \sin(coord.\theta_2), \quad c_2 \leftarrow \cos(coord.\theta_2)$;
$den \leftarrow \sqrt{\max(10^{-20}, 1{,}0 - c_1^2 \cdot c_2^2)}$;
$k \leftarrow coord.r_1/den$;
$x \leftarrow k \cdot c_2 \cdot s_1$;
$y \leftarrow k \cdot c_1 \cdot s_2$;
$z \leftarrow k \cdot s_1 \cdot s_2$;
return (x, y, z)

Dentro del Algoritmo 2, s_1, c_1, s_2, c_2 denotan respectivamente los senos y cosenos de θ_1 y θ_2. La variable auxiliar den representa computacionalmente al divisor polinómico D de la Ecuación 5, incorporando un límite inferior ($\max(10^{-20}, ...)$) para garantizar robustez numérica en las fronteras, mientras que k actúa como un factor de escala radial para ponderar las

proyecciones resultantes.

Optimización de la Transformación Inversa

Durante la implementación del sistema se observó que la obtención de coordenadas SPR inversas presentaba una latencia elevada debido al uso recurrente de la función arcotangente de dos argumentos (`atan2`), empleada por el procesador para resolver la identidad de los cuadrantes. Con el fin de mitigar esta latencia, se diseñó una refactorización consistente en sustituir `atan2` por divisiones fraccionales simples bajo la función `atan` primitiva, trasladando el control de signos a evaluaciones lógicas binarias.

En el Algoritmo 3 se observa la aproximación convencional que depende de llamadas externas a bibliotecas matemáticas generales, lo cual penaliza el rendimiento en cálculos masivos.

Algorithm 3: Transformación Inversa Tradicional

Entrada: (x, y, z)
Salida : $(r_1, \theta_1, \theta_2)$
$r_1 \leftarrow \sqrt{x^2 + y^2 + z^2}$;
$\theta_1 \leftarrow \mathrm{atan2}(z, y)$;
$\theta_2 \leftarrow \mathrm{atan2}(z, x)$;
return $(r_1, \theta_1, \theta_2)$

La versión optimizada (Algoritmo 4) delega la responsabilidad de los cuadrantes a comparaciones atómicas directas. Al reducir el número de operaciones trigonométricas complejas y sustituirlas por lógica booleana de bajo nivel, se logra una reducción drástica de los ciclos de reloj, permitiendo que el sistema escale de manera eficiente en hardware con recursos limitados.

Algorithm 4: Transformación Inversa Optimizada (Refactorización Lógica)

Entrada: (x, y, z)
Salida : $(r_1, \theta_1, \theta_2)$
$r_1 \leftarrow \sqrt{x^2 + y^2 + z^2}$;
if $|y| < 10^{-10}$ **then**
  if $z > 0$ **then**
    $\theta_1 \leftarrow \pi/2$;
  else
    $\theta_1 \leftarrow -\pi/2$;
else
  $\theta_1 \leftarrow \arctan(z/y)$;
  if $y < 0$ **then**
    $\theta_1 \leftarrow \theta_1 + \pi$;
if $|x| < 10^{-10}$ **then**
  if $z > 0$ **then**
    $\theta_2 \leftarrow \pi/2$;
  else
    $\theta_2 \leftarrow -\pi/2$;
else
  $\theta_2 \leftarrow \arctan(z/x)$;
  if $x < 0$ **then**
    $\theta_2 \leftarrow \theta_2 + \pi$;
return $(r_1, \theta_1, \theta_2)$

Un aspecto sutil del Algoritmo 4 es que la corrección de cuadrante ($\theta \leftarrow \theta + \pi$ cuando la coordenada del denominador es negativa) debe residir *exclusivamente* dentro de la rama de la arcotangente. Si dicha corrección se aplicara de forma incondicional tras ambas ramas —como ocurría en una formulación preliminar de este trabajo—, la combinación $|y| < \varepsilon$ con $y < 0$ produciría $\theta_1 = \pi/2 + \pi = 3\pi/2$, y la transformación directa reconstruiría el punto con el signo de z invertido, dado que $\sin(3\pi/2) = -1$. Esta condición de frontera es indetectable con perturbaciones de prueba estrictamente positivas, y fue identificada durante la validación sobre hardware embebido (Sección 7) al someter el algoritmo a perturbaciones simétricas $\pm 10^{-11}$ en régimen recursivo: el RMSE acumulado pasó de $1{,}7 \times 10^{2}$ (divergencia por inversión de signo) a $5{,}7 \times 10^{-12}$ una vez confinada la corrección a la rama del arcotangente, tal como se presenta aquí.

Implementación con CORDIC

El algoritmo CORDIC (*Coordinate Rotation Digital Computer*), introducido por Volder en 1959 [6], es un esquema iterativo que calcula funciones trigonométricas —y sus inversas— mediante únicamente sumas, restas y desplazamientos de bits. Su atractivo fundamental para sistemas embebidos y FPGAs reside en que elimina la necesidad de multiplicadores de

punto flotante y de tablas de biblioteca matemática extensas [7]. En su formulación circular, opera en dos modos:

- **Modo vectorización**: dado un vector cartesiano inicial (x_0, y_0), el algoritmo lo rota iterativamente hasta que su componente y se aproxime a cero ($y \to 0$), acumulando el ángulo recorrido en una variable de fase z. Al converger tras un número total de N iteraciones, el sistema alcanza un estado final (x_N, y_N, z_N), donde la variable z_N aproxima al ángulo de entrada: $z_N \approx \arctan(y_0/x_0)$.
- **Modo rotación**: dado un ángulo objetivo θ, el algoritmo rota un vector inicial configurado como $(x_0, y_0, z_0) = (K_N, 0, \theta)$ paso a paso hasta agotar dicho ángulo ($z \to 0$). Al converger tras N iteraciones, las componentes de estado final resultan en las proyecciones trigonométricas: $x_N \approx \cos\theta$ e $y_N \approx \sin\theta$.

Las iteraciones elementales se definen de manera unificada como:

$$x_{i+1} = x_i - \sigma_i \cdot 2^{-i}\, y_i, \quad y_{i+1} = y_i + \sigma_i \cdot 2^{-i}\, x_i, \quad z_{i+1} = z_i - \sigma_i \cdot \alpha_i \tag{12}$$

donde i representa el índice de la iteración en curso ($0 \leq i < N$), x_i, y_i, z_i son los valores del vector de estado en dicho paso, y $\alpha_i = \arctan(2^{-i})$ es el ángulo precalculado de la i-ésima pseudo-rotación. La dirección de rotación σ_i se evalúa lógicamente como $\sigma_i = -\operatorname{sgn}(y_i)$ en modo vectorización o $\sigma_i = \operatorname{sgn}(z_i)$ en modo rotación. Debido a que las rotaciones se ejecutan sumando tangentes sin normalizar, el vector sufre un alargamiento geométrico en cada paso. Para compensarlo, la magnitud inicial se ajusta multiplicándola por un factor de ganancia invariante (*CORDIC gain*) denotado por K_N, el cual se define estrictamente como:

$$K_N = \prod_{i=0}^{N-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \approx 0{,}6073 \quad (N = 24) \tag{13}$$

Tablas de constantes (datos en ROM / flash)

CORDIC requiere únicamente dos tablas de N entradas que residen permanentemente en la memoria no volátil del microcontrolador. Su cálculo ocurre una sola vez —durante el design-time o el arranque del sistema— y **no se contabiliza en los tiempos de benchmark**; el algoritmo de transformación accede a ellas como datos de sólo lectura sin ningún cálculo adicional:

Tabla 1: Constantes CORDIC pre-calculadas ($N = 24$ iteraciones).

i	2^{-i} (`pow2_table`)	$\alpha_i = \arctan(2^{-i})$ (rad)	α_i (°)
0	1.000000000	0.78539816	45.0000
1	0.500000000	0.46364761	26.5651
2	0.250000000	0.24497866	14.0362
3	0.125000000	0.12435499	7.1250
4	0.062500000	0.06241881	3.5763
5	0.031250000	0.03123983	1.7899
6	0.015625000	0.01562373	0.8952
7	0.007812500	0.00781234	0.4476
8	0.003906250	0.00390623	0.2238
9	0.001953125	0.00195312	0.1119
10	0.000976563	0.00097656	0.0560
11	0.000488281	0.00048828	0.0280
12–23	$\leq 2{,}44 \times 10^{-4}$	$\leq 2{,}44 \times 10^{-4}$	$\leq 0{,}014$
Factor de escala: $K_{24} = 0{,}60725293500888$			

Reducción al primer cuadrante

La ventana de convergencia del bucle CORDIC cubre $\pm 90°$ respecto al estado inicial. Para cubrir la circunferencia completa se adopta la estrategia de *reducir al primer cuadrante* antes del bucle y restaurar el cuadrante correcto mediante una única comparación de signos al final, sin cálculos adicionales.

- **Vectorización** ($\mathrm{atan2}$): se trabaja con $(|x|, |y|)$; el bucle produce $\hat{\alpha} \in [0, \pi/2]$ y el cuadrante del resultado se determina por las siguientes reglas:

$$\mathrm{atan2}(y, x) = \begin{cases} \hat{\alpha} & \text{si } x \geq 0,\ y \geq 0 \quad (Q_1) \\ \pi - \hat{\alpha} & \text{si } x < 0,\ y \geq 0 \quad (Q_2) \\ \hat{\alpha} - \pi & \text{si } x < 0,\ y < 0 \quad (Q_3) \\ -\hat{\alpha} & \text{si } x \geq 0,\ y < 0 \quad (Q_4) \end{cases} \tag{14}$$

  donde las etiquetas Q_1, Q_2, Q_3 y Q_4 denotan explícitamente el primer, segundo, tercer y cuarto cuadrante del plano bidimensional cartesiano, respectivamente.

- **Rotación** ($\sin$ / $\cos$): se explota que $\sin$ es función impar y $\cos(\pi - \theta) = -\cos(\theta)$, reduciendo θ a $[0, \pi/2]$ y ajustando signos al final.

Algorithm 5: CORDIC Vectorización — $\mathrm{atan2}(y, x)$ con reducción a Q1

Entrada: (x_0, y_0), tablas α_i, $p_i = 2^{-i}$ en ROM
Salida : $\mathrm{atan2}(y_0, x_0) \in (-\pi, \pi]$
$\hat{x} \leftarrow |x_0|,\ \hat{y} \leftarrow |y_0|,\ \hat{\alpha} \leftarrow 0$; `// Reducción a Q1`
for $i = 0, 1, \ldots, N-1$ **do**
  $\hat{x}_{\text{prev}} \leftarrow \hat{x}$;
  if $\hat{y} > 0$ **then**
    `//` $\sigma_i = -1$`: rotar en sentido horario`
    $\hat{x} \leftarrow \hat{x} + p_i\,\hat{y}$; $\hat{y} \leftarrow \hat{y} - p_i\,\hat{x}_{\text{prev}}$; $\hat{\alpha} \leftarrow \hat{\alpha} + \alpha_i$;
  else
    `//` $\sigma_i = +1$`: rotar en sentido antihorario`
    $\hat{x} \leftarrow \hat{x} - p_i\,\hat{y}$; $\hat{y} \leftarrow \hat{y} + p_i\,\hat{x}_{\text{prev}}$; $\hat{\alpha} \leftarrow \hat{\alpha} - \alpha_i$;
return $\mathrm{atan2}(y_0, x_0)$ *según Ecuación* (14)

Algorithm 6: CORDIC Rotación — $\sin\theta$ y $\cos\theta$ con reducción de rango

Entrada: $\theta \in (-\pi, \pi]$, tablas α_i, p_i en ROM, $K_N \approx 0{,}6073$
Salida : $\cos\theta$, $\sin\theta$
$s_{\text{neg}} \leftarrow (\theta < 0)$; ;
si s_{neg}: $\theta \leftarrow -\theta$; `// Paridad:` $\sin(-\theta) = -\sin(\theta)$
$c_{\text{neg}} \leftarrow (\theta > \pi/2)$; ;
si c_{neg}: $\theta \leftarrow \pi - \theta$; `// Simetría:` $\cos(\pi - \theta) = -\cos(\theta)$
$(x_0, y_0, z_0) \leftarrow (K_N,\ 0,\ \theta)$;
for $i = 0, 1, \ldots, N-1$ **do**
  if $z_i > 0$ **then**
    `//` $\sigma_i = +1$`: rotar en sentido antihorario`
    $x_{i+1} \leftarrow x_i - p_i\,y_i$; $y_{i+1} \leftarrow y_i + p_i\,x_i$; $z_{i+1} \leftarrow z_i - \alpha_i$;
  else
    `//` $\sigma_i = -1$
    $x_{i+1} \leftarrow x_i + p_i\,y_i$; $y_{i+1} \leftarrow y_i - p_i\,x_i$; $z_{i+1} \leftarrow z_i + \alpha_i$;
if c_{neg} **then**
  $\cos\theta \leftarrow -x_N$;
else
  $\cos\theta \leftarrow x_N$;
if s_{neg} **then**
  $\sin\theta \leftarrow -y_N$;
else
  $\sin\theta \leftarrow y_N$;

En los seudocódigos anteriores, variables como p_i y α_i corresponden a los accesos directos en memoria de lectura (las tablas `pow2_table` y `atan_table`). En el Algoritmo 5, $\hat{x}$, $\hat{y}$ y $\hat{\alpha}$ fungen como registros de trabajo temporal para la reducción al primer cuadrante, apoyándose en la variable puente $\hat{x}_{\text{prev}}$ para la asignación cruzada no destructiva. En el Algoritmo 6,

las banderas booleanas s_{neg} y c_{neg} actúan como selectores lógicos que restablecen los signos trigonométricos de paridad y simetría al finalizar el bucle, tras haber agotado el ángulo θ en la variable acumuladora z_i.

Ejemplo de aplicación: cómputo de $\text{atan2}(0{,}6,\ 0{,}8)$

Para ilustrar la mecánica del algoritmo se calcula $\text{atan2}(0{,}6, 0{,}8)$, cuyo valor exacto es $\arctan(0{,}75) = 36{,}870°$. Ambas entradas son positivas (Q1), por lo que se trabaja directamente con $(|x|, |y|) = (0{,}8, 0{,}6)$. La Tabla 2 registra las primeras ocho iteraciones:

Tabla 2: Primeras 8 iteraciones de CORDIC vectorización para $\text{atan2}(0{,}6, 0{,}8)$.

i	α_i (°)	x_i	y_i	$\text{sgn}(y_i)$	$\hat{\alpha}$ acum. (°)	Error (°)
—	—	0.8000	0.6000	$+$	0.0000	36.870
0	45.0000	1.4000	-0.2000	$+$	45.0000	8.130
1	26.5651	1.5000	0.5000	$-$	18.4349	18.435
2	14.0362	1.6250	0.1250	$+$	32.4711	4.399
3	7.1250	1.6406	-0.0781	$+$	39.5961	2.726
4	3.5763	1.6455	0.0246	$-$	36.0198	0.850
5	1.7899	1.6463	-0.0268	$+$	37.8097	0.940
6	0.8952	1.6467	-0.0011	$-$	36.9145	0.045
7	0.4476	1.6467	0.0118	$-$	36.4669	0.403

Valor exacto: $36{,}870°$. Tras $N = 24$: error $< 6{,}8 \times 10^{-6°}$.

En la Tabla 2, las columnas representan las distintas variables computacionales que evolucionan durante el proceso. Específicamente, i representa el índice de la iteración actual (con $i = 0$ como el paso inicial), α_i (°) es el ángulo de corrección precalculado en grados equivalente a $\arctan(2^{-i})$, y las variables x_i, y_i representan las componentes ortogonales del vector rotado en ese paso específico. La columna $\text{sgn}(y_i)$ muestra el resultado de la función de decisión lógica que evalúa el signo de y_i, la cual determina la dirección angular de la siguiente pseudo-rotación para forzar iterativamente la convergencia de la componente y hacia cero. La columna etiquetada como $\hat{\alpha}$ acum. (°) refleja la fase angular total acumulada dentro del bucle de suma, mientras que la métrica de *Error* (°) cuantifica la diferencia absoluta en grados respecto al valor trigonométrico exacto ($36{,}870°$).

El patrón numérico decreciente observado refleja la búsqueda binaria inherente al algoritmo: en cada iteración se añade o sustrae α_i según el signo de y_i, de modo que el acumulador oscila alrededor del valor verdadero con amplitud decreciente. La velocidad de convergencia es de 1 bit por iteración, lo que implica que tras $N = 24$ iteraciones se alcanza una precisión de $\sim 10^{-7}$ rad ($6{,}8 \times 10^{-6°}$), un margen de error suficientemente bajo para satisfacer la totalidad de los requisitos de control y cinemática en robótica avanzada.

Aplicando la misma operación a las dos componentes angulares del sistema SPR para un punto cartesiano (x, y, z):

$$\theta_1 = \text{atan2}_{\text{CORDIC}}(z,\ y), \qquad \theta_2 = \text{atan2}_{\text{CORDIC}}(z,\ x) \tag{15}$$

La fase de reducción al primer cuadrante garantiza que el bucle de 24 iteraciones sea siempre suficiente independientemente del cuadrante del punto de entrada.

La variante **SPR CORDIC** (identificador `SPR_C` en el benchmark) sustituye `atan2` de biblioteca en la transformación inversa por el Algoritmo 5, y los cuatro cálculos trigonométricos de la transformación directa por dos invocaciones del Algoritmo 6, obteniendo (s_1, c_1) y (s_2, c_2) simultáneamente a partir de tablas en ROM.

6. Metodología de Evaluación Experimental

Diversas investigaciones han documentado que los sistemas basados en retículas esféricas presentan inestabilidades en sus fronteras, lo que provoca una pérdida de precisión decimal y la propagación de errores de coma flotante debido al uso iterativo de series trigonométricas [4], [5].

Para corroborar la eficiencia del sistema propuesto, se desarrollaron algoritmos en arquitecturas nativas (lenguaje C) y entornos interpretados (Python 3). El experimento consistió en someter al sistema SPR, junto con modelos esféricos y cilíndricos, a pruebas con un volumen masivo de coordenadas procesadas mediante transformaciones directas e inversas. Se midió el costo operativo frente a singularidades polares, la retención de precisión decimal mediante la métrica RMSE y el tiempo de ejecución comparativo de cada modelo. Los resultados de escritorio aquí reportados corresponden a una plataforma x86_64 bajo Windows 11, compilando con MSVC 19.x (`/O2`) y la biblioteca matemática UCRT.

Nota metodológica sobre la instrumentación. Los lazos de medición acumulan en una variable `volatile` *las tres componentes* de cada resultado. Esta precaución es determinante: se comprobó que, si el sumidero consume una sola componente (por ejemplo, únicamente el radio), los compiladores que tratan las funciones matemáticas como intrínsecos puros eliminan por código muerto las llamadas trigonométricas cuyos resultados no se usan, reduciendo el lazo medido esencialmente a una raíz cuadrada. Este artefacto puede simular ventajas de latencia de más de un orden de magnitud (tiempos del orden de 5 ns por punto, físicamente imposibles para una secuencia de raíz, divisiones y arcotangentes) y afectó a una versión preliminar de este benchmark; todas las cifras del presente artículo provienen de la instrumentación corregida, cuyos tiempos por operación son consistentes con el costo de instrucción documentado de las funciones involucradas.

El estudio se complementó con una campaña de validación sobre hardware embebido real: un microcontrolador ESP32-D0WD-V3 (Xtensa LX6, doble núcleo, 240 MHz medidos) programado con ESP-IDF v5.4.1 y optimización `-O2`. Esta plataforma resulta particularmente adecuada porque dispone de FPU de precisión *simple* en hardware pero emula la precisión *doble* por software (*soft-float*), lo que permite medir sobre un mismo silicio tres regímenes aritméticos: doble emulada, simple con FPU y aritmética entera de punto fijo. La latencia se midió con el contador de ciclos del núcleo (resolución de un ciclo de reloj) tras una pasada previa de calentamiento de la caché de instrucciones; los lotes de $1\,024$ puntos se generan con un PRNG determinista (*xorshift32*, semilla fija), de modo que las once variantes evaluadas procesan exactamente los mismos puntos, y cada medición promedia $16\,384$ transformaciones. El protocolo de estabilidad ($1\,000$ ciclos recursivos ida–vuelta) es idéntico al de PC,

con la salvedad de que las perturbaciones del escenario crítico abarcan el rango simétrico $\pm 10^{-11}$ (en lugar de valores exclusivamente positivos), condición que resultó determinante para exponer la condición de frontera discutida en la Sección 7. La variante entera implementa la transformación completa en punto fijo (posiciones Q11.20, ángulos Q3.28 en radianes, funciones trigonométricas Q1.30) con bucles CORDIC *branchless* mediante negación condicional por máscara de signo y raíz cuadrada entera de 64 bits; los resultados se transmiten por UART como CSV y se capturan con los scripts del directorio `benchmark_esp32/host/`.

7. Resultados Numéricos y Discusión

Las siguientes subsecciones presentan el análisis comparativo de los cinco modelos evaluados: Cilíndrico, Esférico, SPR Tradicional (`SPR_T`), SPR Refactorizado (`SPR_O`) y la nueva variante **SPR CORDIC** (`SPR_C`).

Desempeño en Regiones de Singularidad

La literatura científica advierte sobre la carga computacional requerida al procesar ubicaciones que interceptan los polos cardinales [8]. Para evaluar este aspecto, se procesaron millones de posiciones con una separación mínima respecto al eje normal ($x, y \approx 10^{-13}$).

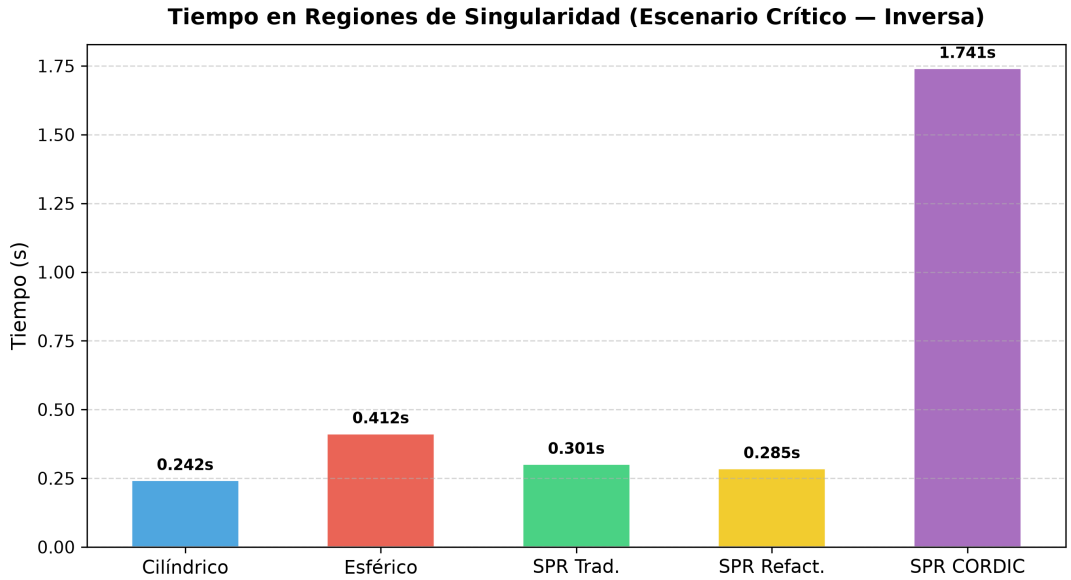


Figura 2: Tiempo de ejecución en la resolución de coordenadas próximas al polo. El incremento en el tiempo refleja la complejidad matemática que representan las fronteras polares para los sistemas convencionales.

Como se ilustra en la Figura 2, los modelos convencionales incrementan su latencia al procesar el entorno polar: el cilíndrico un 65 % y el esférico un 11 % respecto al escenario estándar. En contraste, las variantes SPR se mantienen estables o incluso mejoran: SPR Refact. reduce su tiempo un 25 % en el escenario crítico —sus arcotangentes de razón extrema

convergen por la vía rápida de la biblioteca— y se convierte en el modelo tridimensional más veloz en la vecindad de la singularidad. La variante SPR CORDIC exhibe una latencia esencialmente idéntica en ambos escenarios (~ 1 % de variación), reflejo de su convergencia iterativa de paso fijo, cuya arquitectura de acceso a memoria resulta además más predecible para cachés de instrucción reducidas (relevante en microcontroladores de gama baja).

Estabilidad Numérica y Degradación Decimal (RMSE)

La Figura 3 revela un comportamiento crítico en el límite del dominio. En el *Escenario Estándar*, el sistema SPR Trad. (Algoritmo 3/1) mantiene una deriva de $\sim 10^{-14}$, comparable a la de los demás modelos. La variante SPR Refact. (Algoritmo 4/2) muestra un RMSE estándar de $\sim 2{,}3\times10^{-12}$ como consecuencia del acumulado de redondeo numérico en la formulación sin/cos con cuatro evaluaciones trigonométricas por paso, aunque esta magnitud sigue siendo despreciable en la práctica de ingeniería. En el *Escenario Crítico* (singularidades axiales, $x, y \approx 10^{-11}$) se expone la fragilidad de los sistemas clásicos.

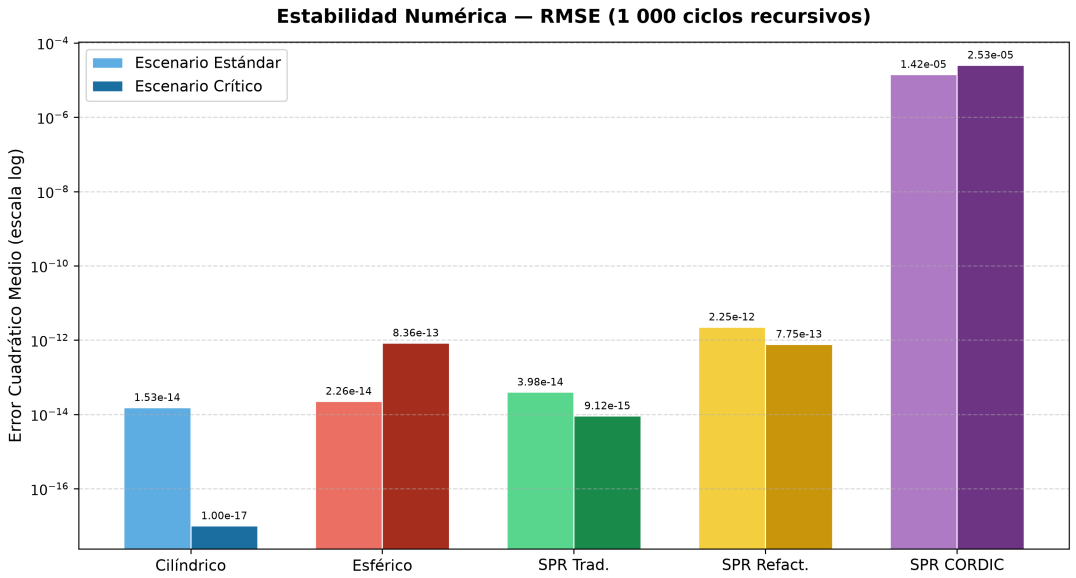


Figura 3: Estabilidad numérica (RMSE) comparando Escenario Estándar vs Escenario Crítico. El sistema SPR mantiene una precisión constante incluso en la proximidad de singularidades axiales.

La degradación de precisión en los modelos esféricos (unas 40 veces: de $\sim 2 \times 10^{-14}$ en el escenario estándar a $\sim 8 \times 10^{-13}$ en el crítico) se origina en la sensibilidad de la función `acos(z/r)`. Cuando un punto se aproxima al polo, ligeras variaciones en z o r producen grandes cambios en el ángulo resultante debido a la pendiente casi infinita de la arcocoseno en sus límites operativos. Por el contrario, el sistema SPR mantiene una precisión constante e independiente de la ubicación del punto, actuando como un filtro natural ante la erosión decimal del estándar IEEE-754.

La variante SPR CORDIC introduce un error de redondeo adicional derivado de la convergencia iterativa: con $N = 24$ iteraciones la precisión alcanzable es de aproximadamente

$2^{-24} \approx 6 \times 10^{-8}$ radianes por ángulo. En régimen acumulativo (1 000 ciclos recursivos), esto se manifiesta como un RMSE ligeramente superior al de las versiones SPR con funciones de biblioteca, pero inferior al de los modelos esféricos en el escenario crítico. Dicho compromiso resulta aceptable en arquitecturas embebidas que privilegian la eliminación de multiplicadores (FPGAs, DSPs de núcleo fijo) frente a la precisión absoluta [7].

Análisis Crítico de Tiempos y Latencia Computacional

Los resultados temporales tras procesar más de 10^7 puntos muestran el perfil característico de SPR: una conversión inversa más rápida que la del esférico y una directa más costosa (Figuras 4 y 5). Dado que en un lazo de control la conversión inversa se ejecuta con mayor frecuencia, este perfil favorece a SPR en su dominio de aplicación.

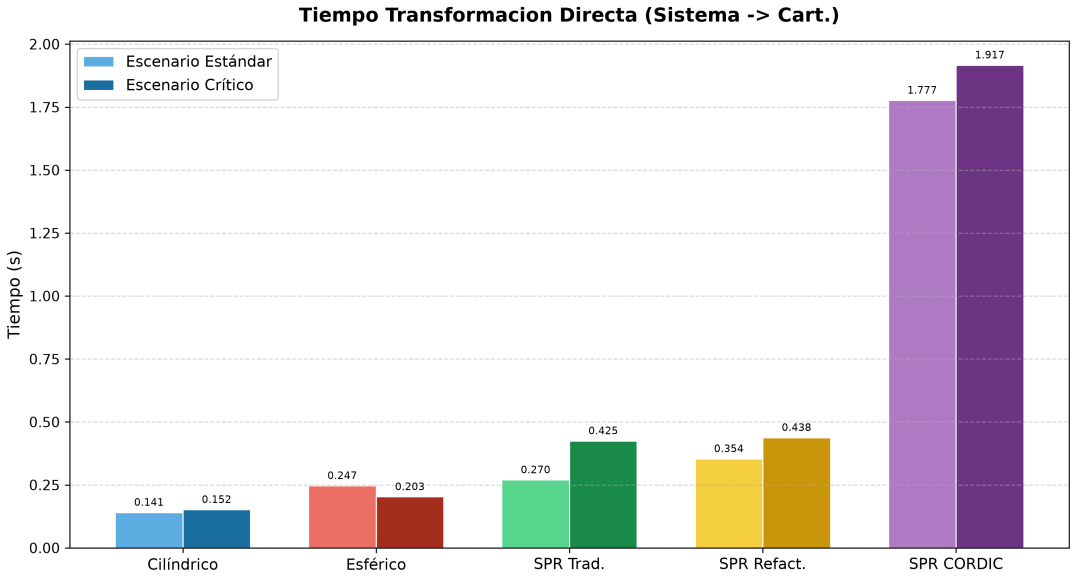


Figura 4: Eficiencia de la Transformación Directa comparando los cinco modelos evaluados (x86_64, MSVC/UCRT). El modelo cilíndrico lidera al requerir una única pareja sin/cos; SPR Trad. (dos tangentes) presenta un costo comparable al del esférico; SPR Refact. paga sus cuatro evaluaciones trigonométricas; SPR CORDIC refleja el costo de iterar 48 pseudo-rotaciones (24 por ángulo) en punto flotante de doble precisión.

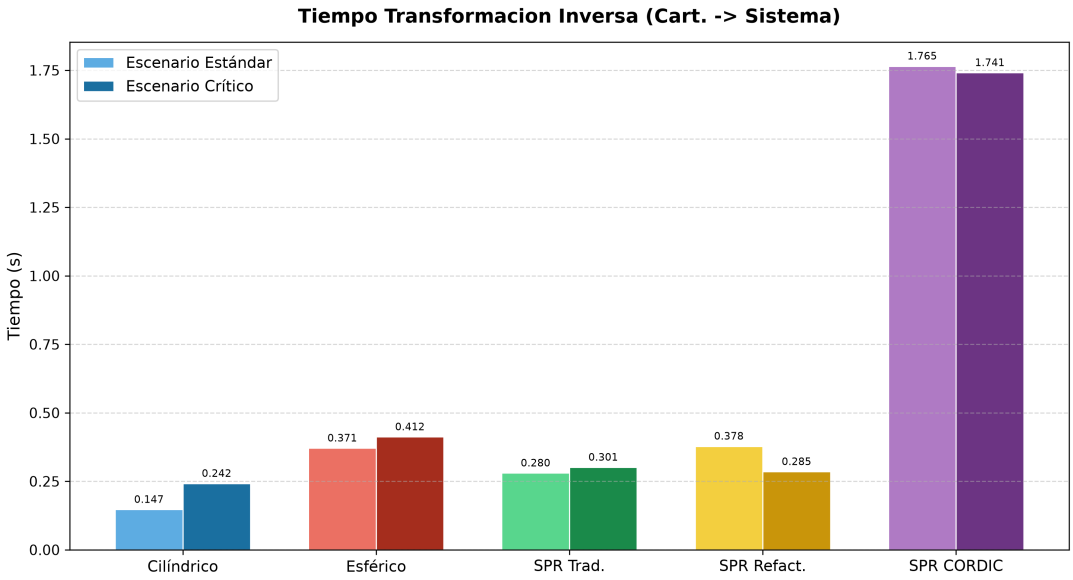


Figura 5: Tiempo de la Transformación Inversa para los cinco modelos (x86_64, MSVC/UCRT). Entre los modelos tridimensionales, SPR Trad. es el más rápido en el escenario estándar y SPR Refact. en el crítico, superando en ambos casos al esférico. El costo de SPR CORDIC en doble precisión anticipa la discusión sobre el rol de la FPU (Sección 7.3.1).

Con la instrumentación corregida, la transformación inversa de SPR supera al modelo esférico en ambos escenarios sobre x86_64: SPR Trad. es $1{,}33$ veces más rápida en el escenario estándar ($0{,}28$ s frente a $0{,}37$ s por 10^7 puntos) y SPR Refact. $1{,}45$ veces en el crítico ($0{,}29$ s frente a $0{,}41$ s). Matemáticamente, la ventaja proviene de sustituir la evaluación de `acos` sobre la hipotenusa tridimensional por arcotangentes simples y lógica condicional binaria. La magnitud del beneficio depende de la implementación de `atan`/`atan2`/`acos` de cada biblioteca matemática, y se conserva sobre la *newlib* del ESP32 ($1{,}15\times$–$1{,}46\times$; Sección 7). Cabe hacer constar que versiones preliminares de este trabajo reportaban un factor de $15\times$, identificado posteriormente como manifestación del artefacto de eliminación de código muerto descrito en la nota metodológica de la Sección 6.

El Rol de la FPU y la Paradoja de Latencia de CORDIC

Al observar los resultados del benchmark, resulta contraintuitivo que CORDIC —un algoritmo históricamente célebre por su velocidad en hardware— registre un tiempo de ejecución notablemente superior ($\sim 1{,}8$ s) frente a las formulaciones algebraicas de SPR ($0{,}28$–$0{,}38$ s). Para comprender esta aparente asimetría, es indispensable entender el impacto de la Unidad de Punto Flotante (FPU, por sus siglas en inglés).

La FPU es un coprocesador de hardware integrado en prácticamente todos los procesadores modernos (como el x86_64 utilizado en el experimento o los núcleos ARM Cortex-A de una Raspberry Pi). Su propósito exclusivo es resolver cálculos matemáticos pesados (multiplicaciones, divisiones y raíces de precisión doble) a nivel de transistores, empleando paralelismo masivo. Cuando se ejecuta el sistema SPR Algebraico en estas arquitecturas, la FPU absorbe la carga trigonométrica en silicio puro y responde en pocos ciclos de reloj. Bajo

este escenario, CORDIC pierde ventaja: el procesador moderno se ve forzado a iterar secuencialmente un bucle de software de 24 pasos (desperdiciando la potencia de su FPU), siendo inevitablemente más lento. Por lo tanto, **en hardware con FPU, el enfoque SPR Refactorizado (Algebraico) es inmensamente superior**.

El verdadero valor de la variante CORDIC emerge cuando se traslada la cinemática a plataformas embebidas restrictivas para robótica autónoma o industrial:

1. **Microcontroladores sin FPU (o de precisión mixta)**: En arquitecturas restrictivas (como ARM Cortex-M0, familias Arduino ATmega, o plataformas de uso masivo como el ESP32 operando forzosamente en doble precisión), no existe circuitería matemática para realizar cálculos de 64 bits de forma nativa. Si se solicita trigonometría estándar, el microcontrolador debe emular matemáticamente cada división mediante una pesada librería de software (*soft-float*), demorando miles de ciclos de reloj. Aquí, el enfoque SPR CORDIC (implementado en aritmética entera) brilla al prescindir de la FPU y limitarse al uso de la Unidad Aritmético Lógica (ALU) estándar para realizar sumas y desplazamientos de bits a máxima velocidad. La Sección 7 cuantifica este efecto sobre un ESP32 físico y confirma un matiz esencial: la ventaja exige la implementación *entera* del algoritmo, pues ejecutar las iteraciones CORDIC sobre aritmética `double` emulada resulta contraproducente.

2. **FPGAs y Hardware Analítico**: En Arreglos de Compuertas (FPGA), implementar el circuito de una FPU consume cantidades desorbitadas de área lógica (*LUTs*). CORDIC, en cambio, se sintetiza naturalmente como un circuito en cascada (*pipeline*) de bajo coste espacial, capaz de entregar coordenadas trigonométricas puras en fracciones de microsegundo [6], [7].

Para maximizar su eficiencia general, la implementación embebida del CORDIC se refactorizó para ser estructuralmente *branchless* (sin saltos condicionales): la dirección de cada pseudo-rotación se resuelve mediante negación condicional por máscara de signo, eliminando las instrucciones `if-else` del bucle iterativo y asegurando un flujo lineal sin penalizaciones por fallos en el predictor de saltos del procesador (*branch misprediction*). Fundamentalmente, esta estructura ejecuta la misma secuencia de instrucciones con independencia de los datos, lo que garantiza un tiempo de ciclo estricto y determinista para Sistemas Operativos de Tiempo Real (RTOS), propiedad verificada empíricamente sobre ESP32 (variación menor al $0{,}2$ % entre escenarios; Sección 7).

En términos de precisión, la Tabla 3 resume el error máximo esperado para distintas profundidades de iteración:

Tabla 3: Error de aproximación CORDIC por número de iteraciones.

Iteraciones (N)	**Error máx. (rad)**	**Bits de precisión**
12	$\sim 2{,}4 \times 10^{-4}$	~ 12
16	$\sim 1{,}5 \times 10^{-5}$	~ 16
20	$\sim 9{,}5 \times 10^{-7}$	~ 20
24	$\sim 6{,}0 \times 10^{-8}$	~ 24
32	$\sim 2{,}3 \times 10^{-10}$	~ 32

En un procesador embebido a 100 MHz, el ahorro de microsegundos de las variantes SPR (Refact. y CORDIC) es lo que permite ejecutar el lazo de control de una plataforma móvil con latencia total inferior al umbral de inestabilidad dinámica (típicamente < 1 ms).

Validación Empírica en Hardware Embebido: ESP32

Para contrastar las hipótesis anteriores fuera del entorno de escritorio, el benchmark completo se ejecutó sobre un microcontrolador ESP32-D0WD-V3 físico, bajo las condiciones descritas en la metodología. La Tabla 4 resume el costo de la transformación inversa en los tres regímenes aritméticos disponibles en el chip.

Tabla 4: Ciclos de CPU por transformación inversa en ESP32 (escenario estándar, promedio sobre 16 384 operaciones).

Variante	**`double` (soft-float)**	**`float` (FPU)**	**entero Q20**
Cilíndrico	5 100	458	—
Esférico	10 050	958	—
SPR Trad.	8 821	802	—
SPR Refact.	8 769	**659**	—
SPR CORDIC	24 939	949	**1 808**

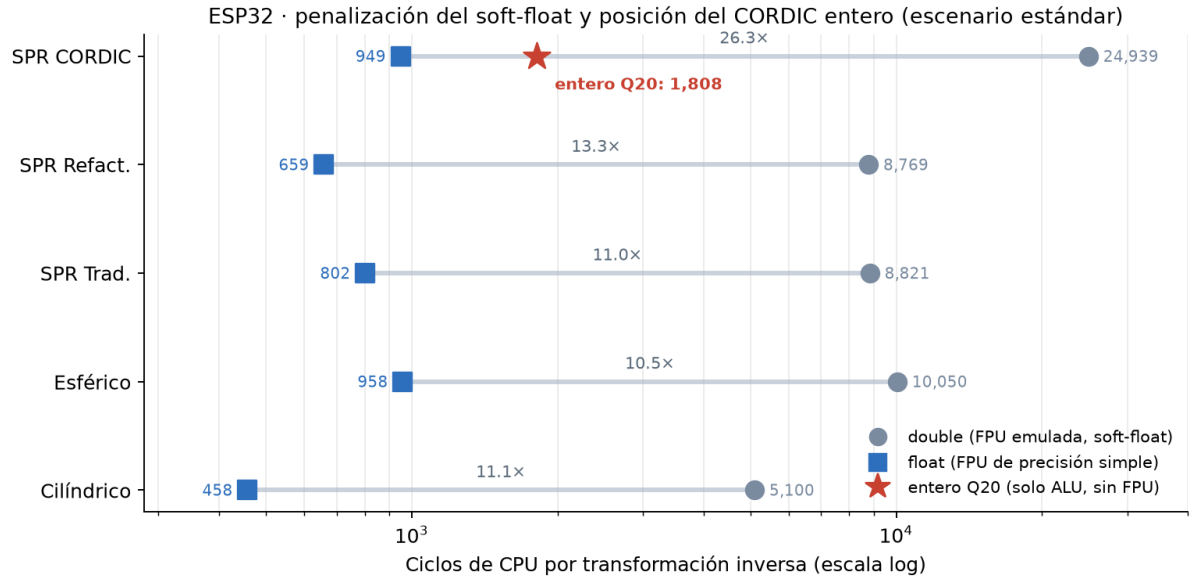


Figura 6: Penalización del *soft-float* en ESP32 para la transformación inversa. Cada segmento conecta la implementación en `float` (cuadrados, FPU de precisión simple) con su equivalente en `double` (círculos, emulación por software) del mismo modelo; la anotación indica el factor de penalización. La estrella señala la variante SPR CORDIC en aritmética entera Q20, que no emplea la FPU.

La Figura 6 revela la estructura del costo computacional. La emulación de doble precisión penaliza a los modelos basados en biblioteca matemática con factores de $10{,}5\times$ a $13{,}3\times$, pero castiga aún más severamente al CORDIC en punto flotante ($26{,}3\times$): al iterar 24 pasos cuyas sumas y productos pasan individualmente por *soft-float*, el algoritmo hereda la penalización en cada operación elemental, convirtiéndose en la variante más lenta del chip (24 939 ciclos). Este resultado empírico refina la hipótesis planteada en la Sección anterior: **la ventaja de CORDIC en hardware restringido no surge de su estructura iterativa por sí sola, sino de su capacidad de formularse en aritmética entera pura**. En efecto, la implementación en punto fijo Q20 —que reemplaza multiplicaciones por desplazamientos y elimina todo salto condicional del bucle— resuelve la inversa en 1 808 ciclos: $4{,}8\times$ más rápida que la mejor alternativa algebraica en `double` (SPR Refact., 8 769) y $13{,}8\times$ más rápida que el propio CORDIC flotante emulado.

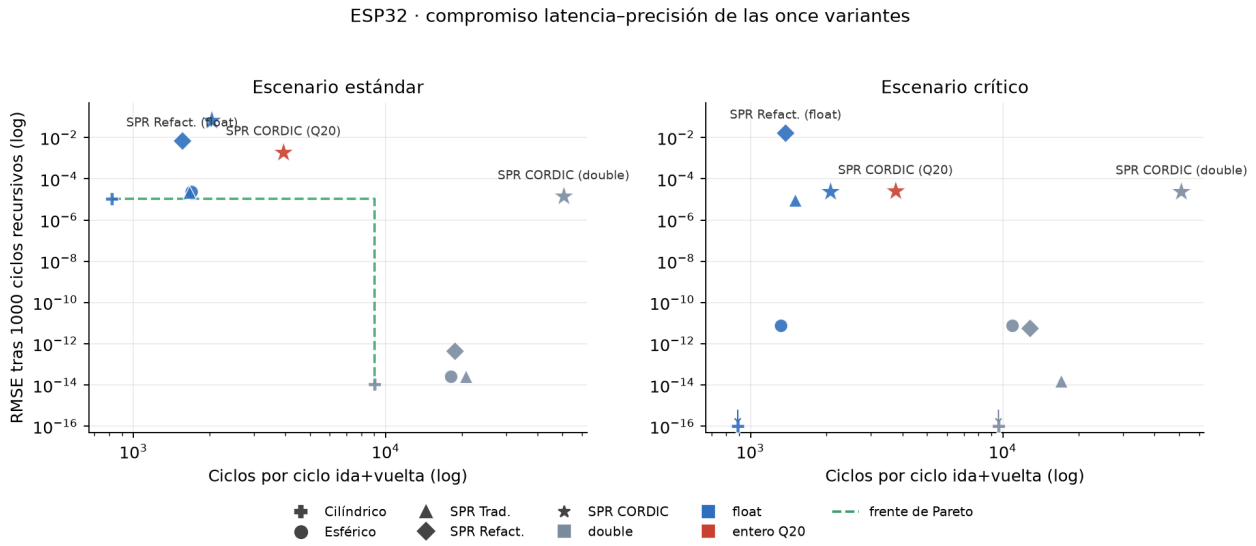


Figura 7: Compromiso latencia–precisión de las once variantes evaluadas en ESP32. Cada punto representa una variante (forma: modelo; color: régimen aritmético); el eje horizontal es el costo del ciclo cinemático completo (inversa + directa) y el vertical el RMSE tras 1 000 ciclos recursivos. Las flechas indican valores por debajo del piso del eje. La línea discontinua traza el frente de Pareto del escenario estándar.

El mapa de la Figura 7 sitúa a cada variante en el espacio latencia–precisión y permite seleccionar la implementación óptima según el requisito dominante. Tres observaciones merecen destacarse. Primera: el frente de Pareto del escenario estándar está definido por las variantes `float` en el extremo de baja latencia y por las variantes `double` en el extremo de alta precisión; la variante entera Q20 se ubica en el codo intermedio, ofreciendo precisión de $\sim 10^{-3}$ (dominada por la cuantización Q20, no por el algoritmo) a una fracción del costo de cualquier ruta `double`. Segunda: en el escenario crítico, las variantes CORDIC (tanto `float` como Q20) resultan *más precisas* que la refactorización algebraica en `float`: al ser la vectorización y la rotación CORDIC operaciones mutuamente inversas construidas con las mismas tablas, el par transformación–antitransformación es autoconsistente y el error recursivo no se amplifica cerca de la singularidad. Tercera: la degradación relativa de las variantes `float` tras 1 000 ciclos recursivos refleja el piso de precisión simple bajo realimentación extrema del error, no el error por operación individual (del orden de 10^{-6} relativo), que es el relevante en un lazo de control real donde cada consigna se procesa una sola vez.

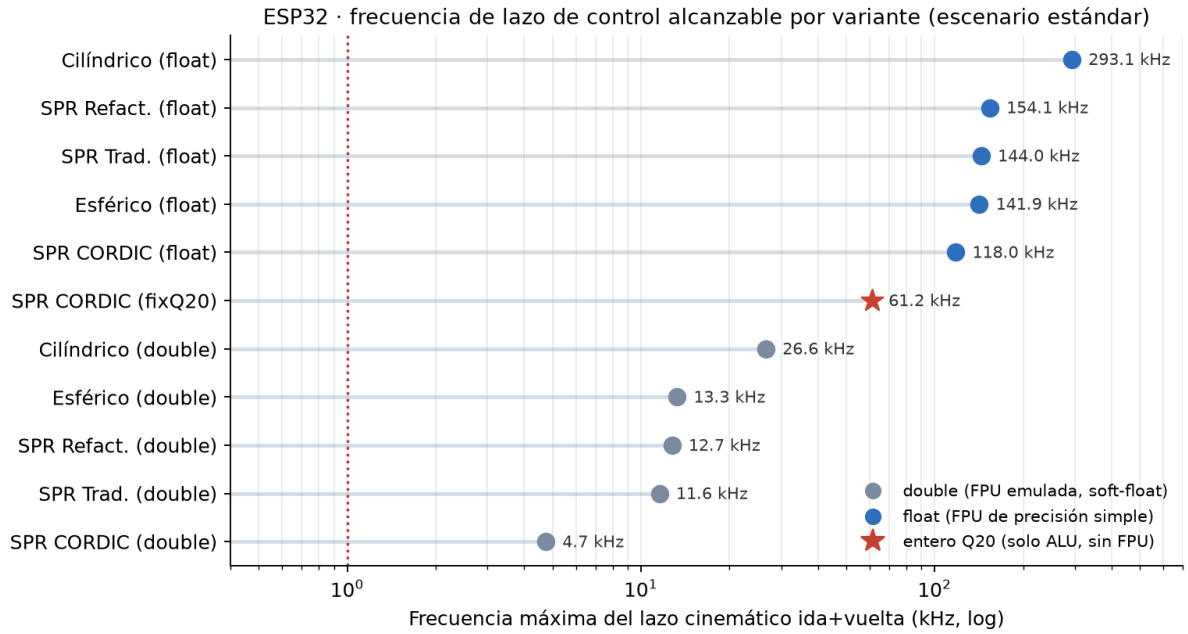


Figura 8: Frecuencia máxima teórica del lazo cinemático (transformación inversa + directa) para cada variante en ESP32 a 240 MHz, frente al umbral de estabilidad dinámica de 1 kHz típico en robótica móvil. Escala logarítmica.

En términos de ingeniería de control (Figura 8), todas las variantes superan el umbral de 1 kHz en este chip de 240 MHz, pero con márgenes radicalmente distintos: el ciclo cinemático completo de SPR CORDIC entero admite lazos de hasta 61 kHz, frente a los 12.7 kHz de SPR Refactorizado en `double` y los 4.7 kHz del CORDIC flotante emulado; con FPU de precisión simple, SPR Refactorizado alcanza 154 kHz. Estos márgenes determinan cuánta capacidad de cómputo queda disponible para el resto del lazo (dinámica, filtrado, comunicación) o, alternativamente, si un microcontrolador de menor frecuencia (y consumo) puede sostener la misma tasa de control.

Un atributo adicional de la variante entera, medido directamente, es su **determinismo**: la latencia difiere menos del $0{,}2$ % entre el escenario estándar y el crítico (1 808 frente a 1 806 ciclos), pues el bucle *branchless* ejecuta exactamente la misma secuencia de instrucciones con independencia de los datos. Esta propiedad, inalcanzable para las bibliotecas matemáticas con rutas condicionales internas, es la que garantiza tiempos de ciclo estrictos en planificadores de tiempo real.

Finalmente, la campaña embebida aporta dos acotaciones de honestidad metodológica. Por un lado, confirma que la magnitud de la ventaja de SPR depende de la biblioteca matemática de cada plataforma: con la *newlib* de Xtensa, la superioridad de SPR sobre el modelo esférico en la inversa es de $1{,}15\times$ en `double` y $1{,}46\times$ en `float`, frente al $1{,}33\times$–$1{,}45\times$ medido con la UCRT en x86_64; el signo del beneficio es consistente en todas las plataformas evaluadas, pero su escala no es transferible entre bibliotecas. Por otro lado, fue precisamente esta validación con perturbaciones simétricas la que expuso la condición de frontera del Algoritmo 4 descrita en la Sección 5.2, lo que subraya el valor del ensayo *hardware-in-the-loop* como complemento indispensable de los benchmarks de escritorio.

Síntesis: posición de SPR frente a los demás sistemas

Reuniendo las mediciones de ambas plataformas, la posición de cada sistema de coordenadas puede resumirse en cuatro puntos:

- **Conversión inversa** (de cartesianas al sistema; la que un controlador ejecuta con mayor frecuencia al procesar lecturas de posición). El sistema cilíndrico es el más rápido en todos los casos, pero maneja un solo ángulo y no basta para describir la orientación de un posicionador de dos ejes. Entre los dos sistemas que sí la describen —esférico y SPR—, **SPR ocupó el primer lugar en todas las pruebas**: 1.33 veces más rápido que el esférico en la PC (escenario estándar), 1.45 veces en el escenario crítico, y entre 1.15 y 1.46 veces en el ESP32 según la precisión empleada.
- **Conversión directa** (del sistema a cartesianas). Aquí el orden se invierte: el cilíndrico y el esférico son más rápidos, y SPR paga sus evaluaciones trigonométricas adicionales. En un lazo de control esta conversión se ejecuta con menor frecuencia que la inversa, por lo que el balance del ciclo completo sigue siendo favorable, pero es la limitación a tener en cuenta si una aplicación ejecuta principalmente conversiones directas.
- **Precisión cerca del polo**. SPR es el único sistema de dos ángulos cuyo error no crece cerca de la singularidad: tras mil conversiones repetidas del mismo punto, su error se mantiene en el orden de 10^{-12}, mientras que el esférico se degrada unas 40 veces respecto a su comportamiento en el resto del dominio.
- **Procesadores sin FPU o limitados a doble precisión emulada**. La versión entera de SPR con CORDIC fue la opción más rápida del ESP32 en este régimen (4.8 veces sobre la mejor alternativa de doble precisión) y la única cuyo tiempo de ejecución es constante, lo que simplifica el análisis de plazos en sistemas de tiempo real. Ninguno de los otros sistemas evaluados cuenta con una implementación equivalente validada.

En síntesis: cuando la aplicación requiere dos ángulos, SPR ofrece la conversión inversa más rápida y la mejor estabilidad numérica de los sistemas evaluados; y cuando el hardware es muy limitado, su variante entera mantiene la velocidad y añade tiempo de ejecución constante sin necesidad de unidad de punto flotante.

8. Conclusiones

Esta investigación presentó el sistema de coordenadas SPR y lo comparó con los sistemas cilíndrico y esférico midiendo diez millones de conversiones en dos plataformas: un procesador de escritorio y un microcontrolador ESP32. Los resultados sitúan a SPR en una posición concreta: entre los sistemas de dos ángulos, su conversión inversa fue la más rápida en todas las pruebas (entre 1.15 y 1.5 veces sobre el esférico) y su precisión fue la única que no se degradó cerca de los polos. Su conversión directa es más costosa que la del esférico, de modo que la ventaja global de SPR se concentra en las aplicaciones donde domina la conversión inversa, que es el caso típico de los lazos de control.

En términos prácticos, esto se traduce en margen de cómputo sobre hardware económico. En el ESP32 a 240 MHz, el ciclo completo de conversiones de SPR admite lazos de control de 12.7 kHz en doble precisión, 154 kHz en precisión simple y 61 kHz en su versión entera: en todos los casos, muy por encima del umbral de 1 kHz habitual en robótica móvil. El tiempo restante del procesador queda disponible para las demás tareas del controlador (dinámica, filtrado, comunicaciones).

La estabilidad cerca del polo es la segunda característica distintiva. El error de SPR no depende de la ubicación del punto: tras mil conversiones repetidas se mantiene en el orden de 10^{-12}, mientras que el sistema esférico se degrada unas 40 veces en la vecindad de su singularidad. Para mecanismos que atraviesan esas orientaciones (plataformas de apuntamiento, muñecas robóticas), esto elimina una fuente de error difícil de diagnosticar.

La variante entera de SPR con el algoritmo CORDIC [6], [7] completa el panorama: fue la opción más rápida del ESP32 cuando se exige doble precisión o no hay unidad de punto flotante (4.8 veces sobre la mejor alternativa) y la única con tiempo de ejecución constante (variación menor al 0.2 %), lo que simplifica el diseño de sistemas de tiempo real. Cuando el procesador dispone de FPU de precisión simple, la versión algebraica de SPR es la más rápida (2,7 μs por conversión inversa a 240 MHz). Ambas variantes comparten la misma formulación geométrica: el diseñador puede elegir la implementación según su hardware sin cambiar el modelo matemático. Finalmente, la validación sobre hardware real permitió detectar y corregir una condición de frontera en la conversión inversa y un artefacto en la instrumentación de medición, lo que refuerza la fiabilidad de las cifras aquí publicadas.

Direcciones de correo de los autores

Salas-García, J.*: Facultad de Ingeniería, Universidad Autónoma del Estado de México, Toluca, Estado de México, México
proyectos@javiersaalsg.com

Hernández-Servín, J. A.: Facultad de Ingeniería, Universidad Autónoma del Estado de México, Toluca, Estado de México, México
joseph_servin@uaemex.mx

Vilchis-González, A. H.: Facultad de Ingeniería, Universidad Autónoma del Estado de México, Toluca, Estado de México, México
avilchisg@uaemex.mx

Ávila-Vilchis, J. C.: Facultad de Ingeniería, Universidad Autónoma del Estado de México, Toluca, Estado de México, México
jcavilav@uaemex.mx

Referencias

[1] A. Codourey, «Dynamic Modeling of Parallel Robots for Computed-Torque Control Implementation,» *The International Journal of Robotics Research*, vol. 17, n.º 12, págs. 1325-1336, 1998, doi: 10.1177/027836499801701205.

[2] J.-P. Merlet, *Parallel Robots*, 2.ª ed. Dordrecht: Springer, 2006, doi: 10.1007/1-4020-4133-0.

[3] C. Gosselin y J. Angeles, «Singularity analysis of closed-loop kinematic chains,» *IEEE Transactions on Robotics and Automation*, vol. 6, n.º 3, págs. 281-290, 1990, doi: 10.1109/70.56660.

[4] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob y P. N. Swarztrauber, «A standard test set for numerical approximations to the shallow water equations in spherical geometry,» *Journal of Computational Physics*, vol. 102, n.º 1, págs. 211-224, 1992, doi: 10.1016/S0021-9991(05)80016-6.

[5] B. Siciliano, L. Sciavicco, L. Villani y G. Oriolo, *Robotics: Modelling, Planning and Control*. London: Springer, 2009, doi: 10.1007/978-1-84628-642-1.

[6] J. E. Volder, «The CORDIC Trigonometric Computing Technique,» *IRE Transactions on Electronic Computers*, vol. EC-8, n.º 3, págs. 330-334, 1959, doi: 10.1109/TEC.1959.5222693.

[7] R. Andraka, «A Survey of CORDIC Algorithms for FPGA Based Computers,» en *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, New York: ACM, 1998, págs. 191-200, doi: 10.1145/275107.275139.

[8] R. Sadourny, «Conservative Finite-Difference Approximations of the Primitive Equations on Quasi-Uniform Spherical Grids,» *Monthly Weather Review*, vol. 100, n.º 2, págs. 136-144, 1972, doi: 10.1175/1520-0493(1972)100<0136:CFAOTP>2.3.CO;2.