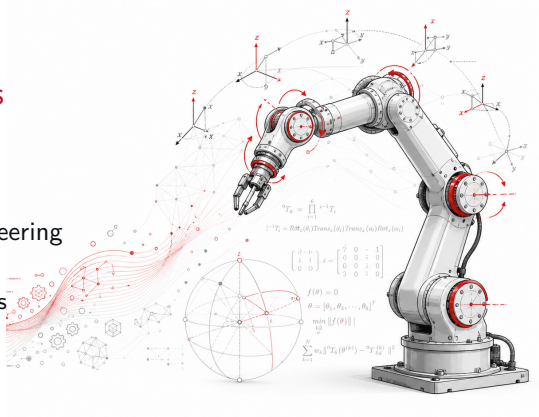
IDR Lab
Intelligent Design and Robotics Laboratory
GEN-AI DESIGN | MR SIMULATION

THE OHIO STATE UNIVERSITY

LLM-Assisted Human-AI

Development of Analytical
Inverse Kinematics Solvers

Closed-form IK for 6R robot manipulators



Hai-Jun Su
Intelligent Design and Robotics Laboratory
Department of Mechanical and Aerospace Engineering
The Ohio State University

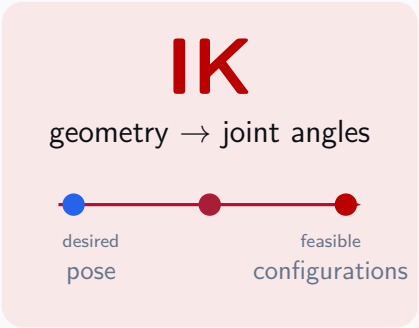
US Symposium on Mechanical Systems and Robotics
(MSR 2026) | University of California, Irvine
May 21–23, 2026

Published in *Mechanism and Machine Theory*, Vol. 222, Article 106392, 2026

Why inverse kinematics still matters

Analytical IK is attractive because it can return **all solution branches** with deterministic runtime.

- No initial guess
- No convergence failure
- Near machine precision
- Real-time control compatible

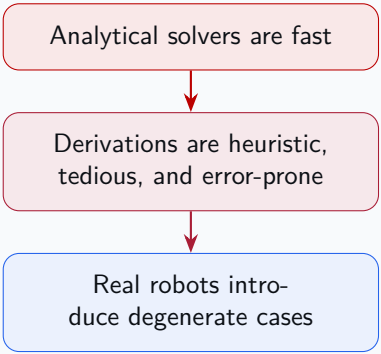


Speaker message

Analytical IK is useful; robust solver creation is the bottleneck.

The persistent bottleneck is derivation

Approach	Strength	Practical limit
Jacobian iteration	Flexible	One local solution; singularities
Homotopy / algebraic	Complete roots	Heavy runtime for control
IKFast / IK-Geo	Automation	Setup and edge cases remain hard
Learning-based IK	Fast samples	Training data; no completeness



Context from Su, Mechanism and Machine Theory 222:106392 (2026).

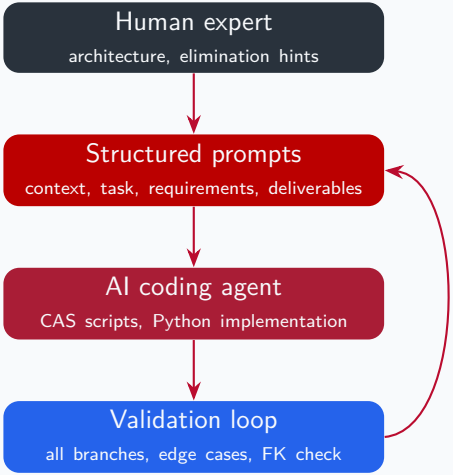
Core idea: human strategy, AI execution

- Human expert supplies **decoupling strategy**
- AI agent performs **symbolic derivation**
- Code is refined through **validation feedback**
- Reusable helper solvers handle common equation patterns

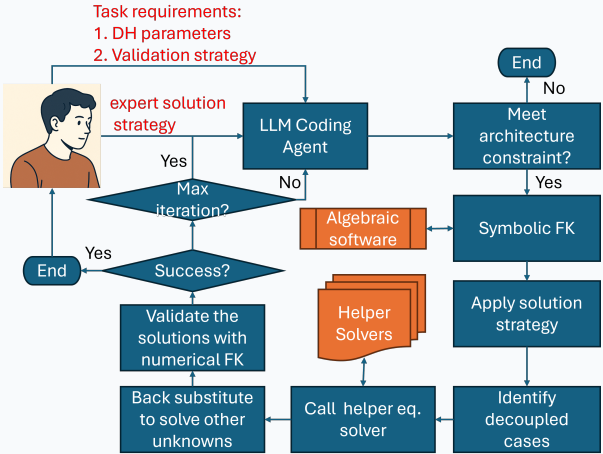
100% success rate

1–2 ms Python solve time

10^{-9} position error scale

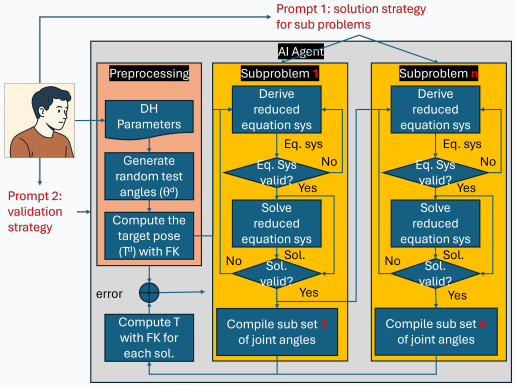


AI-assisted development workflow



The agent derives symbolic FK, identifies decoupled cases, calls helper solvers, back-substitutes, and validates every candidate through numerical FK.

Prompting decomposes the search space



Prompt structure

- Context
- Task
- Requirements
- Deliverables

- Prompts encode solver strategy
- Validation prompts expose missing branches
- Expert guidance narrows symbolic search

Reusable helper solver library

Equation system	Solver function	Max sol.
$a\cos\theta + b\sin\theta + c = 0$	`solve_trig_eq()`	2
$A\begin{bmatrix}\cos\theta\\ \sin\theta\end{bmatrix} = \mathbf{c}$	`solve_trig_sys_single()`	1
$A\begin{bmatrix}\cos\theta_1\\ \sin\theta_1\end{bmatrix} + B\begin{bmatrix}\cos\theta_2\\ \sin\theta_2\end{bmatrix} = \mathbf{c}$	`solve_trig_sys()`	4
$K\,[1, c_1, s_1, c_2, s_2, c_1c_2, c_1s_2, s_1c_2, s_1s_2]^T = \mathbf{0}$	`solve_bilinear_sys()`	8

The helper functions are developed and tested once, then reused as black-box analytical components in the 6R solvers.

Prompt text: helper trigonometric solver

Context and task

Context. Expert mathematician and numerical analyst for trigonometric equations in robot kinematics.

Task. Implement `solve_trig_eq(a,b,c)` for

$$a\cos x + b\sin x + c = 0$$

using the Weierstrass substitution $t = \tan(x/2)$ and mapping roots back to angles.

Requirements and deliverable

- Handle degenerate, sinusoidal, cosine, and endpoint cases.
- Return all real roots normalized to $[-\pi, \pi]$.
- Re-evaluate every candidate in the original equation.
- Include derivation notes and deterministic edge tests.
- Add at least 1000 randomized known-root tests.

Deliverable. A self-contained Python module with the solver and stress-test helper.

Condensed from the helper-solver prompt excerpt in the paper source.

Problem setup: modified DH kinematics

Link	α_{i-1}	a_{i-1}	d_i	θ_i
1	α_0	a_0	d_1	θ_1
2	α_1	a_1	d_2	θ_2
3	α_2	a_2	d_3	θ_3
4	α_3	a_3	d_4	θ_4
5	α_4	a_4	d_5	θ_5
6	α_5	a_5	d_6	θ_6

Forward kinematics

$$^{0}\mathbf{T}_{6} = {}^{0}\mathbf{T}_{1}\,{}^{1}\mathbf{T}_{2}\,{}^{2}\mathbf{T}_{3}\,{}^{3}\mathbf{T}_{4}\,{}^{4}\mathbf{T}_{5}\,{}^{5}\mathbf{T}_{6}$$

$$^{i}\mathbf{T}_{i+1} = \mathbf{X}(\alpha_i, a_i)\mathbf{Z}(d_{i+1}, \theta_{i+1})$$

The solvers target general DHM parameters subject only to the architecture constraints introduced next.

Normalize the target pose before solving

Base/tool offsets are absorbed

$$\mathbf{T}_6' = \mathbf{Z}(-d_1, 0)\,\mathbf{X}(-\alpha_0, -a_0)\,{}^0\mathbf{T}_6^d\,\mathbf{Z}(-d_6, 0)$$

$$\mathbf{T}_6' = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

solver requirements

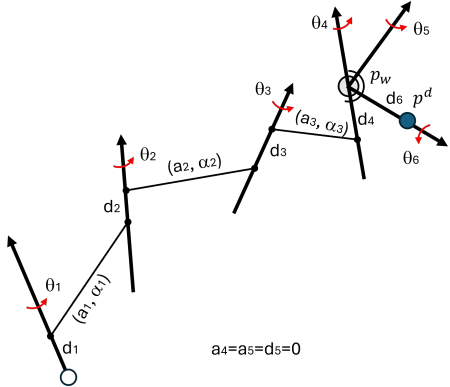
- all real solutions
- singularity handling
- arbitrary DH parameters
- numerical FK validation

up to 8
joint configurations

Case study 1

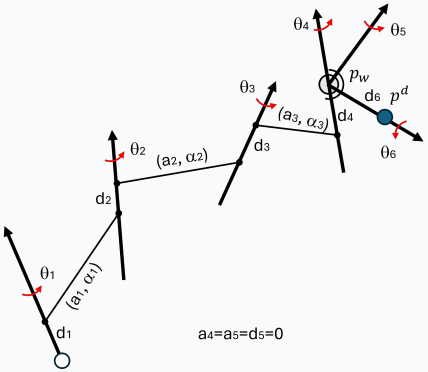
6R robots with spherical wrists

Decouple position from orientation
using the intersecting wrist axes



Constraints: $a_4 = 0,\ d_5 = 0,\ a_5 = 0$

Spherical wrist: architecture and constraints



Minimal constraints

$$a_4 = 0, \qquad d_5 = 0, \qquad a_5 = 0$$

- Joints 4, 5, 6 intersect at a wrist center
- First 3R subchain remains fully general
- Orientation is solved after position

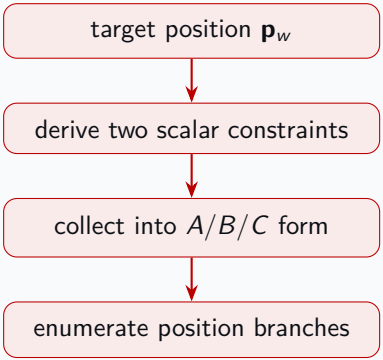
Unlike classical restrictions on the first 3R chain, only the wrist intersection is imposed.

Spherical wrist: position subproblem

Coupled two-angle system

$$\mathbf{A}\begin{Bmatrix}\cos\theta_1\\ \sin\theta_1\end{Bmatrix}+\mathbf{B}\begin{Bmatrix}\cos\theta_3\\ \sin\theta_3\end{Bmatrix}=\mathbf{C}$$

- Extract wrist center $\mathbf{p}_w$
- Solve (θ_1,θ_3) with `solve_trig_sys()`
- Recover θ_2 with `solve_trig_sys_single()`

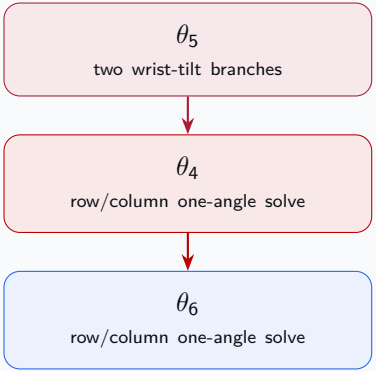


Spherical wrist: orientation branches

Reduced wrist orientation

$$^{4}\mathbf{R}_{6}^{d} = \mathbf{R}_z(\theta_4)\mathbf{R}_x(\alpha_4)\mathbf{R}_z(\theta_5)\mathbf{R}_x(\alpha_5)\mathbf{R}_z(\theta_6)$$

- Solve θ_5 from the r_{33} relation
- Solve θ_4 and θ_6 as one-angle systems
- Validate complete six-angle tuples by FK



Singular cases with $\sin(\theta_5) = 0$ are handled by fixing one wrist angle and solving the other.

Prompt text: spherical wrist solver

Context and task

Context. Analytical robotics expert guiding a coding agent for general 6R robots with spherical wrists. Reuse `solve_trig_sys()` and `solve_trig_sys_single()`.

Task. Build `ik_6r_spherical_wrist.py` with a public `solve_ik(...)` API.

Key instructions

- Split IK into spatial 3R position and wrist orientation subproblems.
- Absorb base/tool offsets before solving.
- Decouple θ_1 from θ_3; recover θ_2 with a single-angle helper.
- Solve θ_5 from wrist-axis alignment; solve θ_4, θ_6 with one-angle systems.
- Validate edge cases and at least 200 random robot/pose samples.

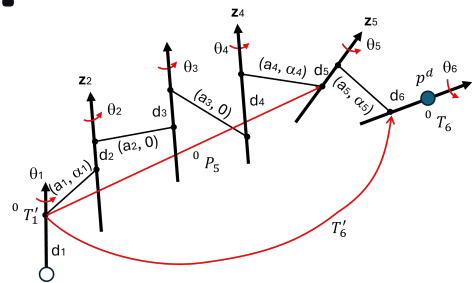
Condensed from llm_ik_6r_wrist prompt.

Case study 2

6R robots with

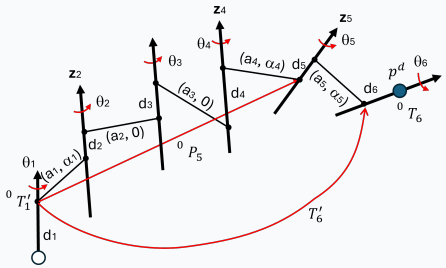
parallel joints 2–4

Reduce joints 1 and 6 to a bilinear system,
then solve a planar 3R subproblem



Constraints: $\alpha_2 = 0^\circ$, $\alpha_3 = 0^\circ$

Parallel joints: bilinear reduction



Two geometric constraints

$$\mathbf{z}_4 \cdot \mathbf{z}_5 - \cos(\alpha_4) = 0$$

$$(\mathbf{P}_{05}^R - \mathbf{P}_{05}^L) \cdot \mathbf{z}_2 = 0$$

$$\begin{bmatrix} a_{10} & a_{11c} & \cdots & a_{1ss} \\ a_{20} & a_{21c} & \cdots & a_{2ss} \end{bmatrix} \mathbf{m} = \mathbf{0}$$

$$\mathbf{m} = (1, c_1, s_1, c_6, s_6, c_1c_6, c_1s_6, s_1c_6, s_1s_6)^T$$

Minimal constraints

$$\alpha_2 = 0^\circ, \qquad \alpha_3 = 0^\circ$$

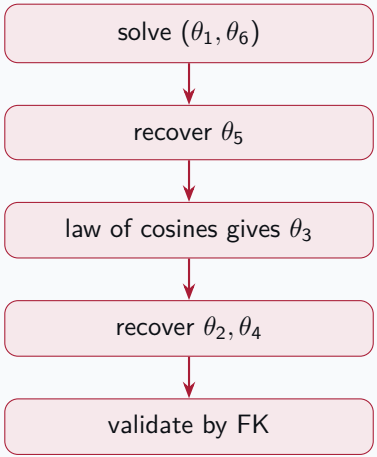
Parallel joints: finish with planar 3R IK

Planar position equations

$$p_x^2 + p_y^2 = a_2^2 + a_3^2 + 2a_2a_3\cos\theta_3$$

$$p_x = a_2\cos\theta_2 + a_3\cos(\theta_2+\theta_3)$$

$$p_y = a_2\sin\theta_2 + a_3\sin(\theta_2+\theta_3)$$



Prompt text: parallel-joint solver

Context and task

Context. Analytical robotics expert and prompt engineer for 6R manipulators whose joint axes 2, 3, and 4 are parallel.

Task. Generate `ik_6r_3parallel_solver.py` with a public `solve_ik(...)` API and an automated validation harness.

Key instructions

- Move ${}^{5}T_{6}$ to form reduced wrist/hand equations.
- Use CAS to derive two scalar equations in c_1, s_1, c_6, s_6.
- Arrange the result as a bilinear two-angle system.
- Recover θ_5, then solve planar 3R for $\theta_2, \theta_3, \theta_4$.
- Run deterministic tests plus 100×10 randomized stress cases.

Condensed from ik_6r_parallel prompt.

Validation campaign

Real robot validation

- RobotKinematicsCatalogue models
- 10 random poses per robot/configuration
- FK residual check for every returned solution

Stress testing

- Random DHM parameter sets
- Degenerate and edge cases
- Branch completeness and numerical residuals

842
industrial robots

1000
random configurations

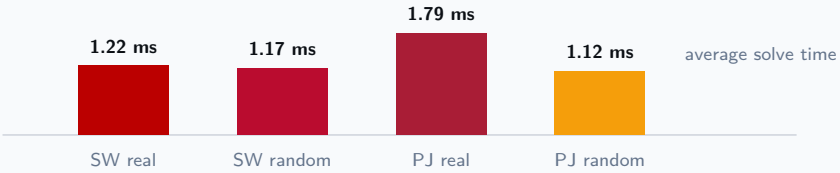
all
candidate branches validated

Tests ran in Python on Intel i7-10700KF @ 3.80 GHz with 32 GB RAM.

Benchmark results

Table 1: Solver validation results

Family	Test	Robots	Tests	Success	Time	Max error
Spherical wrist	Real	723	10	100%	1.22 ms	2.19×10^{-9}
Spherical wrist	Random	1000	20	100%	1.17 ms	2.68×10^{-11}
Parallel joints	Real	119	10	100%	1.79 ms	3.45×10^{-9}
Parallel joints	Random	100	10	100%	1.12 ms	1.82×10^{-9}



What the results mean relative to alternatives

Method family	Typical limitation	This work's advantage
Jacobian methods	local, iterative, singularity sensitive	all real branches; no convergence criterion
Homotopy continuation	complete but seconds to minutes	analytical Python solve in 1.1–1.8 ms
IKFast / IK-Geo	strong automation, setup still expert-heavy	prompts encode expert strategy; agent handles implementation
Learning-based IK	training data and generalization concerns	no training data; near machine precision

main shift expert time moves from symbolic coding to strategic supervision.

Takeaways

- LLM agents can execute symbolic derivation, code generation, testing, and refinement.
- Human experts remain essential for architecture recognition and solver strategy.
- Two general 6R families were solved with minimal constraints.
- Validation shows 100% success and near machine-precision accuracy.

Closing message

Human expertise becomes an orchestration layer for analytical engineering software.

Resources

Official paper DOI
Project page
Helper solvers
Spherical wrist code
Parallel joint code

Contact
su.298@osu.edu