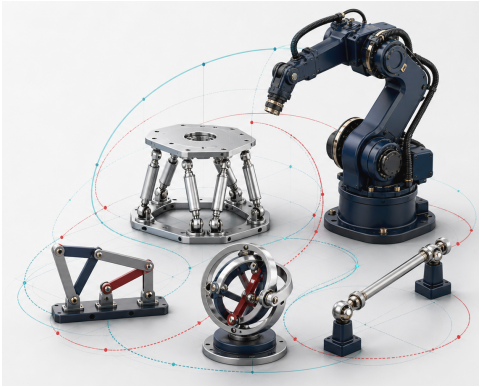
Solving Kinematics Problems by Leveraging the Power of Large Language Models

A practical guide to validation-centered human–AI solver development

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

ASME IDETC/CIE 2026
15-minute talk + 5-minute Q&A



Motivation: from validated 6R inverse kinematics to a broader kinematics toolkit

WORKFLOW

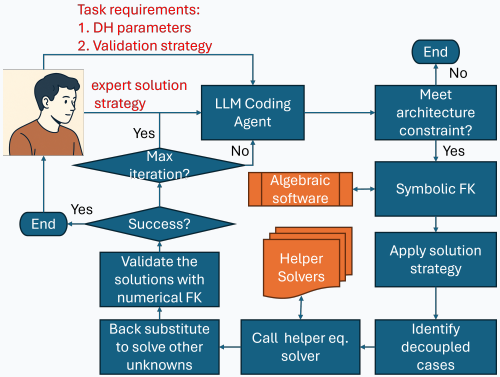
Human–AI collaboration

Human expert: architecture, decomposition, task requirements, and validation strategy.

LLM coding agent: executable implementation, algebraic-tool calls, and iteration on test evidence.

Why expand the problem set?

Can this validation-centered loop transfer from special 6R robots to **classical linkage, synthesis, and parallel-mechanism problems**?



Source: H. Su, *Mechanism and Machine Theory*, 2026 (MMT 6R IK study).

Expansion goal: stress the workflow on three kinematics families

What changes from the MMT case

- classical papers and chapters replace robot-architecture source material
- algebra spans low-degree equations, numerical sensitivity, and symbolic elimination
- evidence adapts: residuals, root counts, planted roots, and degree checks

Out of scope

- no controlled head-to-head benchmark
- no claim of autonomous mathematical discovery
- no guarantee under arbitrary prompt

1. Planar and spherical linkages
Low-degree algebra, known solution counts

2. Spatial SS dyad synthesis
Matrix polynomial with numerically delicate elimination

3. Stewart–Gough forward kinematics
Exact symbolic elimination to a 40th-degree polynomial

Set up the agentic environment before asking it to solve kinematics

WORKFLOW

Four levels of useful tool integration

1. Chat for formulation and prompt refinement
2. Agentic IDE with a local Python interpreter
3. CAS integration through Mathematica / `wolframscript`
4. Sandbox agent that can write, run, inspect, and revise files

Minimum reproducible workspace

- source PDF/chapter + task prompt in one local folder
- Python with NumPy/SciPy and Mathematica installed
- terminal recognizes `python` and `wolframscript`
- dedicated output files for scripts, examples, and reports

validation-centered

not autonomy-centered

Prompt anatomy: define both the solver and its evidence

Every executable prompt includes four blocks

- **Context**: role and source discipline
- **Task**: exact derivation or implementation target
- **Requirements**: tolerances, edge cases, validation
- **Deliverables**: script, example, command log, and report

The agent must run the checks, not just describe them

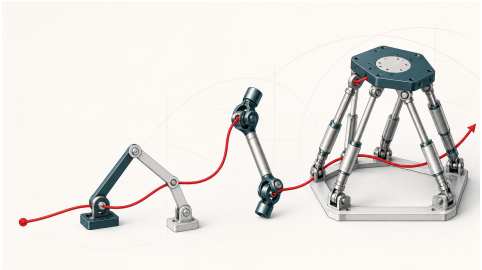
- known solution counts
- residuals against governing equations
- planted-root recovery
- branch coverage and singular-case behavior
- exact arithmetic when elimination is symbolic

Source material + explicit checks > generic prompting

01 | CASES

Case studies

The same agentic loop is tested against progressively more delicate mathematics.



Main talk flow: linkage problems, spatial SS dyads, then Stewart–Gough forward kinematics.

Case 1: planar and spherical linkage problems

CASE STUDY

Problems reproduced from McCarthy and Soh, *Geometric Design of Linkages*, 2010.

- four-bar analysis using the Freudenstein equation
- planar RR dyad synthesis
- planar PR dyad synthesis
- spherical RR dyad synthesis

Explicit formulation + known branches + direct residual checks make these useful first tests.

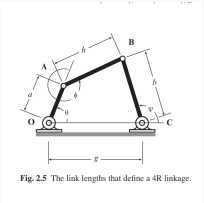


Fig. 2.5 The link lengths that define a 4R linkage.

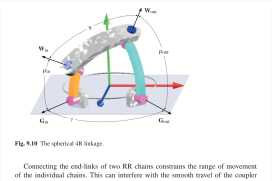


Fig. 9.10 The spherical 4R linkage.

Connecting the end-links of two RR chains constrains the range of movement of the individual chains. This can interfere with the smooth travel of the coupler

Planar 4R: Ch. 2, Fig. 2.5; spherical 4R: Ch. 9, Fig. 9.10.

Prompting lesson

Read the source chapter first; then implement its derivation, enumerate branches, and verify the numerical example.

Linkage results: one structured prompt was often enough

RESULTS

Reported outcomes

Problem	Expected	Found	Match
Four-bar analysis	2	2	Yes
RR dyad (3 pos.)	1	1	Yes
PR dyad (3 pos.)	1	1	Yes
Spherical RR (4 pos.)	∞	∞	Yes
Spherical RR (5 pos.)	up to 6	2 real, 4 complex	Yes

$$< 10^{-10}$$

numerical agreement with textbook results

Interpretation

For explicit low-degree formulations, the agent mostly behaved like a fast implementation assistant rather than a research collaborator.

Case 2: seven-position spatial SS dyad synthesis

CASE STUDY

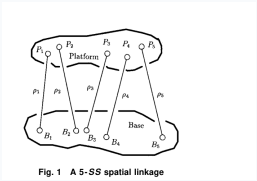


Fig. 1 A 5-*SS* spatial linkage

Liao and McCarthy, 2001, Fig. 1.

Constraint equation

$$\|[A_i]\mathbf{p} + \mathbf{d}_i - \mathbf{B}\|^2 = r^2$$

One constraint for each of seven prescribed positions.

Elimination structure

- 6 linear design equations in $\mathbf{p}$
- 6×4 augmented matrix $C(\mathbf{B})$
- 15 quartic determinantal constraints
- 15×15 matrix polynomial $\mathcal{M}(B_z)$

The hidden numerical trap

$$\mathcal{M}(B_z) = M_0 + M_1 B_z + M_2 B_z^2 + M_3 B_z^3 + M_4 B_z^4$$

Nominal determinant degree can blow up to 60, while the geometry leaves an **effective degree of 20**.

Why the first solver failed

Initial agent behavior

- misread details in the source paper
- misunderstood the elimination process
- expanded $\det(\mathcal{M}(B_z))$ directly

12 / 20

real roots recovered by symbolic determinant expansion

Failure diagnosis from the manuscript

- determinant expansion creates huge intermediate products
- alternating cancellations destroy the discriminating coefficients
- rank deficiency in leading coefficient blocks warns of trouble
- precision sensitivity appears before the final root solve

Expert correction: solve the matrix polynomial directly

Companion pencil reformulation

$$\mathcal{A}\mathbf{v} = B_z\,\mathcal{B}\mathbf{v}$$

$$\mathcal{A} = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ -M_0 & -M_1 & -M_2 & -M_3 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & M_4 \end{bmatrix}$$

Recovery steps

1. keep finite real eigenvalues
2. SVD of $\mathcal{M}(B_z^*)$ for (B_x, B_y)
3. SVD of $C(B_x, B_y, B_z^*)$ for $\mathbf{p}$
4. compute chain length L

The key human contribution was method selection, not code typing.

SS dyad evidence: the corrected solver recovers all solutions

RESULTS

Method comparison

Method	Real roots	Planted root
Symbolic $\det(\mathcal{M})$ + NSolve	12 / 20	No
Companion-matrix eigenvalue	20 / 20	Yes

$$\sim 10^{-13}$$

constant-distance residual across all 7 frames

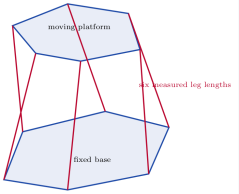
First four recovered solutions

Sol	L	**B**
1	2.862	$(-1.453,\ -0.413,\ -1.178)$
2	3.379	$(-0.376,\ -0.269,\ -2.255)$
3	3.435	$(-0.405,\ -0.884,\ -1.240)$
4	3.721	$(0.810,\ -0.974,\ -2.716)$

The appendix carries the full 7-position input table; the paper lists all 20 solutions.

Case 3: Stewart–Gough forward kinematics

CASE STUDY



Generated schematic (Paper 1); problem source: Husty (1996).

Problem statement

Given 6 leg lengths for a general 6–6 platform, recover all assembly configurations of the moving platform.

Study-parameter reduction

$$X_1x_1 + X_2x_2 + X_3x_3 + X_4x_4 = 0$$

$$x_1 = kx_4, \quad x_2 = mx_4, \quad x_3 = px_4$$

Target algebraic object

Seven constraint vectors $R_1, \ldots, R_7$ are reduced through determinant, resultant, and GCD stages to a **40th-degree characteristic polynomial** in p.

up to 40 configurations

order-sensitive elimination

The Stewart solver worked only when the prompt respected the published elimination order

Algorithmic sequence

1. form determinants $G_i(k, m, p)$
2. extract the null-space parameterization
3. substitute back to obtain $G_7(k, m, p)$
4. eliminate k with Sylvester resultants
5. eliminate m with a second resultant cascade
6. isolate the degree-40 GCD polynomial

Non-negotiable constraint

Exact integer / rational arithmetic

Floating-point symbolic elimination can introduce spurious factors, obscure common divisors, and change the recovered degree.

Source: Husty, *Mechanism and Machine Theory*, 1996.

Stewart results: stronger prompt specificity, less iteration than SS

RESULTS

Reported evaluation

Metric	Value
Polynomial degree	40
Known root $p = -1$ recovered	Yes
CAS used	Mathematica
Arithmetic during elimination	Exact integer / rational
Alternative elimination order	Not reliable
Derivation runtime	$\sim$ 30 min

Interpretation

- the agent can translate a published symbolic algorithm into CAS code
- success depends on preserving the original elimination order
- robustness to mathematically equivalent reformulations is still weak

What changes across the three cases?

Explicit low-degree problems

Source chapter + solution count + residual checks

Outcome: one structured prompt often suffices.

Numerically delicate polynomial systems

The agent needs help selecting a stable formulation

Outcome: expert diagnosis becomes central.

Order-sensitive symbolic elimination

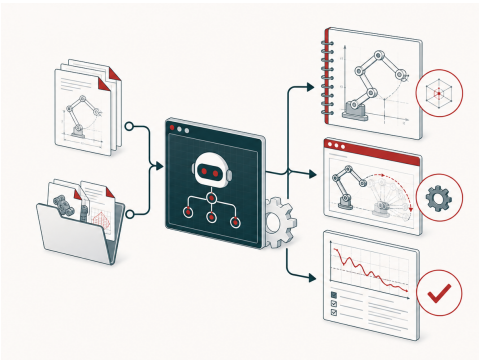
Prompt must encode the exact algorithmic sequence and arithmetic mode

Outcome: LLM acts as a faithful translator.

Problem complexity dictates the human–AI interaction pattern.

Workflow

The practical result is a runnable solver and a validation record.



Source material and a local toolchain feed an agent; the outputs must be scripts, numerical examples, and validation reports.

Practical setup: make the agent's local workspace executable

PRACTICAL GUIDE

Create one task folder

1. put the source PDF/chapter and prompt file together
2. open that folder in an agentic IDE
3. allow the agent to create scripts and reports inside it
4. keep outputs separate from the supplied source material

Verify the toolchain before the task

```
python -version
wolframscript -version
```

Python runs numerical tests; Mathematica executes the symbolic elimination when required.

local source context | executable tools | contained outputs

Generate, run, inspect, and validate: the practical agent loop

Ask the agent to produce

- a runnable `.py` or `.wls` solver
- the numerical example from the source
- saved intermediate coefficients / roots when meaningful
- a concise `report.md` with commands, outputs, and status

Require evidence before acceptance

- source-equation traceability and branch conventions
- residuals, root/solution counts, or planted-root recovery
- explicit arithmetic mode and tolerance settings
- stated failure behavior for singular / non-assembly cases

No validation target = no trustworthy conclusion

The Coding-Agent Landscape Has Moved Fast

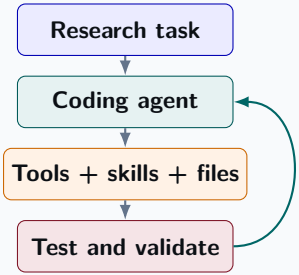
A dated benchmark snapshot

This work was produced in late 2025 and early 2026: an evidence-based snapshot of LLM performance on difficult kinematics problems with expert-specified routes and independent validation.

- **OpenAI Codex** and **Claude Code** couple models to a **harness**: workspace, tools, shell, tests, and iterative edits.
- **Agent skills** package reusable instructions, tools, and examples—a manual for recurring coding tasks.
- **Harness, loop, and graph engineering** are emerging ways to organize tool use, verification, and multi-step execution.

Interpret the paper carefully

Stronger execution environments can improve implementation reliability; they do not prove autonomous selection of a new kinematic derivation.



Can the agent choose the right symbolic strategy, not merely execute it?

Takeaways: expand the problem set, not the trust boundary

1. Start from a proven workflow

The MMT 6R result motivates expansion, but each new family needs its own source-based formulation and evidence target.

2. Tool access turns prompts into experiments

An agentic IDE plus Python/CAS makes it possible to generate scripts, execute them, inspect intermediates, and preserve reports.

3. Human insight remains the method selector

The SS dyad required a companion eigenvalue formulation; Stewart required a prescribed exact-arithmetic elimination order.

4. Validation is the deliverable

Residuals, known roots, solution counts, arithmetic settings, and reproducible reports make generated solver code credible.

Questions? su.298@osu.edu

Appendix: build an executable agent workspace first

1. Local task folder

1. copy the source PDF/chapter and refined prompt into one workspace
2. open that directory in the agentic IDE
3. allow the agent to create scripts, numerical examples, and reports there

2. Tool probes

```
python -version
wolframscript -version
```

3. Execution contract for the agent

- read the supplied source before deriving equations
- generate a runnable `.py` or `.wls` file
- execute the source numerical example locally
- save outputs and a `report.md` validation record

4. Human gate

The human approves the method, checks intermediate quantities, and decides whether the evidence is sufficient.

Appendix: structured prompt template

Prompt skeleton used in the paper

[Context] Read the provided source material first; work like a careful kinematics engineer.

[Task] Derive the governing equations and write a complete solver for the specified problem.

[Requirements] Use the published derivation, preserve branch conventions, include validation cases, and debug until checks pass.

[Deliverables] Provide a runnable `.py` or `.wls` script, a reproducible numerical example, saved outputs, and a report summarizing commands, inputs, outputs, and validation status.

Appendix: SS dyad input data from the paper

Pos.	d_x	d_y	d_z	θ	$-\phi$	ψ
1	0	0	0	0	0	0
2	1	−0.742	−0.134	6.180	4.279	−97.93
3	0.318	−0.509	−0.792	−83.26	−18.23	73.61
4	−0.179	−1.784	−1.043	−170.0	39.54	−50.94
5	−1.258	0.836	−1.499	−84.74	−29.18	150.3
6	−3.594	2.728	−2.033	−8.304	5.036	−68.25
7	−0.050	0.570	−1.486	118.3	−33.80	139.0

Appendix: why the SS dyad example is a useful stress test

What the agent must do

- extract the matrix-polynomial coefficients correctly
- recognize that determinant expansion is unstable
- recover (B_x, B_y) and $\mathbf{p}$ from null spaces
- confirm all 20 real solutions

Why it generalizes

- separates coding skill from method-selection skill
- exposes prompt sensitivity clearly
- rewards explicit validation more than generic fluency
- resembles many polynomial robotics workflows

Appendix: Stewart–Gough caution

What worked

- prompt followed Husty's elimination order
- Mathematica used exact integer / rational arithmetic
- known root $p = -1$ was recovered

What did not generalize

- alternative elimination orders caused expression growth
- mathematically equivalent prompts were not equally robust
- success should not be mistaken for arbitrary symbolic flexibility