Proceedings of the ASME 2026
International Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
IDETC/CIE 2026
August 23–26, 2026, Houston, TX

DETC2026-193726

SOLVING KINEMATICS PROBLEMS BY LEVERAGING THE POWER OF LARGE LANGUAGE MODELS

Hai-Jun Su[1,*], J. Michael McCarthy[2]

[1]Department of Mechanical & Aerospace Engineering, The Ohio State University, Columbus, Ohio, USA
[2]Department of Mechanical & Aerospace Engineering, University of California, Irvine, CA, USA

ABSTRACT

This paper is a practical guide and case-study workflow for using Large Language Models (LLMs) to generate solver code for selected classical kinematics problems, rather than a controlled head-to-head benchmark of commercial agents. The reported solver results were generated with OpenAI Codex 5.3 in a local coding-agent workflow; Claude and Gemini are discussed only as comparable workflow options, not as experimentally compared systems. The study addresses three categories of kinematic challenges: (1) analysis and synthesis of planar and spherical linkages, (2) seven-position synthesis of spatial SS dyads (the spatial Burmester problem), and (3) forward kinematics of general Stewart–Gough platforms involving a 40th-degree characteristic polynomial. For each problem, we document the mathematical formulation, the classical solution procedure, the prompt structure, and the validation checks available from known solution counts, residuals, and planted-root tests. The results show that LLM-generated code can be effective for well-posed tasks when the problem formulation and validation criteria are explicit, but advanced polynomial systems still require human domain expertise to select stable numerical methods and diagnose failures. In particular, the spatial SS dyad example required human direction toward a companion-matrix generalized eigenvalue formulation to avoid catastrophic cancellation in determinant expansion. The contribution is therefore a reproducible workflow template for human–AI collaboration in kinematics solver development, with explicit limitations on autonomy, prompt sensitivity, and numerical method selection.

Keywords: large language models, artificial intelligence, kinematics, kinematic synthesis, kinematic analysis, code generation

1. INTRODUCTION

Kinematics is a foundational discipline in mechanical engineering and robotics that deals with the geometry of motion. The history of kinematics over the past two centuries is characterized by a progressive abstraction of motion and a continuous quest to solve increasingly complex nonlinear polynomial systems. Concurrently, research into closed-loop spatial chains led to the study of parallel robots, most notably the Gough–Stewart platform. The forward kinematics (FK) of this mechanism presented another formidable algebraic challenge, ultimately shown to yield a 40th-degree characteristic polynomial representing up to 40 distinct solutions [1, 2]. These classical milestone problems have historically required sophisticated algebraic elimination techniques and highly robust numerical methods to solve. The analytical solutions to many classical problems are well documented in the literature, yet the process of implementing them—deriving equations, coding solvers, and validating results—remains tedious, error-prone, and time-consuming.

The application of large language models (LLMs) to engineering and scientific problems has rapidly expanded, opening new paradigms for human–AI collaboration in code generation, symbolic manipulation, and computational checking. Recent advances in LLMs such as GPT-4 [3], GPT-5 [4], and Claude 3.5 Sonnet exhibit useful capabilities in mathematical reasoning and symbolic manipulation, but their reliability remains strongly dependent on prompts, tool access, and independent validation. Within robotics, Su presented an LLM-assisted human–AI collaborative framework specifically targeted at developing analytical IK solvers for 6R robots [5].

This paper presents a practical guide for using LLM coding agents to help generate solver scripts for foundational kinematics problems. The examples are kinematic analysis of a four-bar linkage and synthesis of RR, PR, and spherical RR dyads from McCarthy and Soh [6], the spatial SS dyad from Liao and McCarthy [7], and the forward kinematics of the general Stewart–Gough platform from Husty [2]. The purpose is not to establish a statistically rigorous benchmark across LLM vendors. Instead, the paper documents a case-study workflow: how a human expert frames the mathematical problem, supplies source material and expected checks, prompts the coding agent, and then validates or

*Corresponding author: su.298@osu.edu

corrects the generated solver.

2. PROMPTING LLMS FOR CODE GENERATION

Recent LLMs such as OpenAI's Codex 5.3 [4] have shown useful capabilities in code generation and symbolic manipulation, motivating their application to classical kinematics problems. This section describes the collaborative workflow, agent environment, and prompting strategy used throughout the case studies.

2.1. AI Agent Environment Setup

We established a hierarchical environment setup based on the desired level of tool integration. For more challenging kinematics problems, tool access to a local interpreter and CAS is useful, but the user must still inspect intermediate results and guide the agent when the mathematical strategy is underspecified. The four levels of AI tool integration are:

1. **Conversational Chat Mode**: The first level of interaction is conversational chat. This involves submitting questions/requests to chat sessions (e.g., ChatGPT).

2. **IDE-Based Code Generation**: The second level leverages an integrated development environment (IDE) equipped with an AI coding extension and a local interpreter. The user prompts the LLM to write and execute code, with the extension in "agent mode" when file generation and local test execution are desired. In this work, OpenAI's Codex 5.3 [8] was the agent used for the reported solver results. Anthropic's Claude Opus 4.6 [9] and Google's Gemini 3.1 Pro (Thinking) [10] were considered as alternative coding-agent interfaces for the same workflow, but no comparative performance claims are made among these systems. A local Python 3.11 environment allowed the agent to validate numerical solver scripts without cloud execution overhead.

3. **Computational Tool Integration**: The third level integrates computer algebra systems (CAS) like SymPy, Mathematica or MATLAB to handle computationally extensive tasks requiring heavy symbolic derivations. For our symbolic tasks, we utilized Mathematica 14.3, with the AI agent executing scripts via the `wolframscript` command-line utility.

4. **Self-Debugging via MCP in Sandbox**: The highest level of tool integration employs a Model Context Protocol (MCP) server within a sandbox environment using OpenAI's Codex 5.3 model. This setup allows the AI agent to write code, generate numerical examples, execute scripts, and attempt self-debugging directly on the local machine. The agent can analyze output results and iterate on code, but mathematical correctness still depends on human-specified validation criteria, such as comparison against ground-truth data, planted roots, solution counts, or kinematic residual checking.

The division of labor in the case studies is as follows. The human expert selected the case-study problems, supplied the source papers or textbook chapters, identified the expected solution counts or known roots, judged whether residual checks were meaningful, and diagnosed failures when the generated approach was mathematically unstable. The LLM generated Python or Mathematica scripts, intermediate reports, and revised code after receiving this guidance. Credit for the final solvers should therefore be assigned to the human–AI workflow rather than to the LLM alone.

To successfully implement this workflow and reproduce the results presented in this paper, the following minimum requirements and steps must be met:

1. An active subscription to OpenAI's ChatGPT Plus (or higher), GitHub Copilot, or a Google Gemini Pro account.

2. Installation of a compatible IDE, such as Visual Studio Code (VS Code) or the Antigravity IDE.

3. Installation of the appropriate AI extension (e.g., the Codex extension for VS Code; no additional extension is required for Antigravity, which uses a default Gemini account).

4. Installation of Mathematica 14.3 and Python 3.11 (with NumPy and SciPy). Make sure that the AI agent has access to these tools in the command line. For instance "wolframscript" and "python" should be recognized by the AI agent in the terminal of the IDE.

5. Download the prompt files and related reference materials (e.g., source papers or digital book files) from the associated GitHub repositories and unzip them into a local directory.

6. Open the local directory created in the previous step within the chosen IDE.

7. Initiate a chat session with the AI agent using the files in the local workspace.

2.2. Prompt Structure

Each prompt includes: (1) a **context** block establishing the LLM's role as a domain expert, (2) a **task** block defining the specific problem and solution method, (3) a **requirements** block specifying edge cases, numerical tolerances, and validation tests, and (4) a **deliverables** block describing the expected output format.

To illustrate this structure, consider the following example prompt used for the planar RR dyad synthesis problem:

```
[Context] You are an expert in kinematics and mechanism
design.
[Task] Your task is to read the provided PDF (e.g., Chapter 5
of Geometric Design of Linkages) in the workspace to
understand the derivation of the synthesis equations for a
planar RR dyad. Understand the mathematical reasoning and
derive the constraint equations. Then, write a complete
Python script to solve for the moving and fixed pivots.
[Requirements] Test the generated code using the numerical
example provided in the text. Autonomously debug the code
```

```
until the expected results are obtained.
[Deliverables] Deliver a working Python script containing the
solver function and the numerical example validation.
```

To repeat the results given in the paper, submit a prompt such as:

```
Read this prompt file in the current workspace and finish the
task specified in the prompt file.
```

The prompt file contains the four blocks described above.

Alternatively, to start from scratch without using a predefined prompt file, submit a prompt such as:

```
Read the paper given in this PDF file and create Mathematica
scripts to reproduce the results published in this paper.
Compare the results with the numerical example given in the
paper, and debug the script until the expected results are
obtained.
```

This informal prompt typically requires multiple iterations to produce the expected results. Human intervention is required to analyze the results and provide feedback to the AI agent.

3. KINEMATIC ANALYSIS AND SYNTHESIS OF PLANAR AND SPHERICAL LINKAGES

This section presents LLM-generated solutions for several classical problems from the textbook *Geometric Design of Linkages* by McCarthy and Soh [6]. These problems are representative of cases where the mathematical formulation is explicit, the expected solution structure is known, and validation can be stated before code generation.

3.1. Problem Statement

We address three example problems:

- **Analysis of planar four-bar linkages** (Ch. 2): Given link dimensions and the input angle, compute the output angle using the Freudenstein equation.
- **Synthesis of planar RR and PR dyads** (Ch. 5): Given a set of task positions, find the dimensions of an RR or PR dyad that guides a rigid body through these positions.
- **Synthesis of spherical RR dyads** (Ch. 9): Extend the planar synthesis to spherical mechanisms using rotation matrices.

3.2. Kinematics Equations

3.2.1. Kinematic Analysis of Planar Four-Bar Linkages For a planar four-bar (4R) linkage, let the link lengths be defined as g (ground), a (input crank), h (coupler), and b (output crank). Assume the origin of a fixed frame is at the input fixed pivot O, with the x-axis passing through the output fixed pivot C. Let θ be the input angle and ψ be the output angle.

The coordinates of the moving pivots A and B are $\mathbf{A} = (a\cos\theta, a\sin\theta)^T$ and $\mathbf{B} = (g + b\cos\psi, b\sin\psi)^T$. The constant length of the coupler link requires $(\mathbf{B}-\mathbf{A})\cdot(\mathbf{B}-\mathbf{A}) - h^2 = 0$. Substituting the coordinates and grouping terms yields the constraint equation:

$$A(\theta)\cos\psi + B(\theta)\sin\psi = C(\theta) \tag{1}$$

The above equation can be solved for ψ given θ using the tangent half-angle substitution $t = \tan(\psi/2)$, yielding two assembly configurations (open and crossed circuits) for the linkage.

To generate the solver code for this problem, the LLM is supplied with the following structured prompt:

```
[Context] You are an expert in kinematics and mechanism
design. Work like a careful human engineer: read the source
material, extract the governing equations, implement them in
code, and verify results with explicit checks.
[Task] Read the provided ‘Ch2_GDL.txt‘ chapter file in the
workspace (the chapter on planar linkage analysis from
Geometric Design of Linkages). From that text, identify the
formulation for planar 4-bar (4R) kinematic analysis, then
write a complete Python script that: - computes the linkage
configuration from given link dimensions and input angle(s),
- solves for the unknown angle(s) using the chapter’s
equations and conventions, - and reports all physically
valid solution branches.
[Requirements] 1. Do not assume formulas from memory; first
read the provided ‘.txt‘ and follow the derivation given
there. 2. Implement clean, runnable Python code (‘.py‘) with
clear functions for: equation setup, solver logic, branch
handling, and residual/error checking. 3. Include robust
numerical handling: angle wrapping, inverse-trig domain
clipping, tolerance-based validation, and clear behavior for
non-assembly or singular cases. 4. Validate the script with
at least one numerical example from the chapter if available,
otherwise create a reasonable test case and state that it is
user-defined. 5. Autonomously debug until the script runs
successfully and the validation checks pass. 6. Generate a
‘report.md‘ file summarizing: the command(s) run, the
input/link parameters and test angles, key branch outputs,
and the final validation status.
[Deliverables] 1. One working Python script containing the
solver implementation, a reproducible numerical example, and
printed validation results. 2. A short summary in plain text
that includes which equations/sections from the ‘.txt‘ were
used, what test case was run, and whether validation passed.
```

3.2.2. RR Dyad, PR Dyad, and Spherical Linkage Synthesis The synthesis equations for these problem types follow the formulations developed in detail in [6] and are summarized here for completeness. For an RR dyad, the design constraints enforce a constant distance between the fixed pivot $\mathbf{G}$ and the moving pivot $\mathbf{W}$ across all N prescribed task positions; expanding these constraints yields a bilinear system in the pivot coordinates that is solved directly by linear algebra. For a PR dyad, the sliding guide direction $\mathbf{h}$ is attached rigidly to the moving body, and the constraint requires that the relative motion of the fixed pivot in the body frame remain parallel to $\mathbf{h}$, again producing a linear system.

For spherical RR dyads, the planar formulation is extended to a sphere: the synthesis equations are cast entirely in terms of rotation matrices $[R_j]$, with the constant great-circle arc length between the moving pivot $\mathbf{u}$ and fixed pivot $\mathbf{v}$ replacing the Euclidean distance constraint. Specifically, the 4-position and 5-position spherical RR dyad synthesis problems are highly complex, as solving them requires deriving a polynomial equation from the constraints and determining its roots. These two problems necessitate the use of a Computer Algebra System (CAS),

such as Mathematica, to robustly perform the algebraic elimination and obtain all possible solutions.

The prompts used to generate solvers for the planar RR, planar PR, and 4-position spherical RR dyad synthesis problems are available in the project's GitHub repository [11]. To illustrate the integration of CAS for these advanced synthesis challenges, the complete prompt guiding the LLM to solve the 5-position spherical RR problem is provided below, formatted consistently with the previous example:

```
[Context] You are an expert in mechanism synthesis, spherical
kinematics, and symbolic/numeric algebraic elimination. Work
like a careful human engineer: read the chapter text, derive
the exact elimination workflow, implement reproducible CAS
code, and verify all computed solutions. Assume the
workspace contains only this prompt file and Ch9_GDL.txt.
[Task] Read Ch9_GDL.txt and solve the 5-position spherical
RR dyad synthesis problem for all solutions. Follow Section
9.4.6 and implement the algebraic elimination method using:
design equations (9.25), matrix setup for five positions
(9.55), cubic minors R_j (9.56), dehomogenized polynomial
form (9.57), coefficient matrix in powers of y (9.58), and
sixth-degree elimination polynomial P(x) (9.59). Compute all
roots of P(x), determine candidate Burmester axes, and
recover associated moving axes for each valid solution.
[Requirements] 1. Use CAS as the primary solver and provide a
Mathematica script (.wls) that runs end-to-end. 2. Do not
assume formulas from memory; extract and follow Chapter 9
notation directly. 3. Ensure reproducibility: fixed
precision/solver settings, deterministic ordering of roots,
no manual intervention. 4. Report all solutions from the
elimination pipeline: all polynomial roots, real Burmester
axes G_i, corresponding moving axes W_{1,i},
multiplicities/degenerate cases if present. 5. Validate each
candidate solution using residuals of the original
constraints (9.25) across all five positions. 6. Assume only
this prompt file and Ch9_GDL.txt exist in the workspace. 7.
Autonomously debug until results are reproducible. 8.
Generate a report.md summarizing equations used, run
commands, full solution tables, and validation outcomes.
[Deliverables] 1. A runnable Mathematica script (e.g.,
ch9_rr_5pos_synthesis.wls) implementing the full elimination
workflow. 2. Reproducible output files containing polyno-
mial coefficients, roots, recovered axes, and residual checks.
3. A concise report file with chapter-equation traceability,
input orientation data, all computed solutions, and final
validation/pass-fail summary.
```

The reader is referred to Chapters 5 and 9 of [6] for the complete derivations.

3.3. Code Generation by LLM

Each reported problem in this section was solved using a structured prompt submitted to OpenAI's Codex 5.3 [8]. The prompt specified the mathematical formulation (referencing the relevant chapter of [6]), the expected number of solutions, and validation test cases from the textbook exercises. For the spherical RR dyad synthesis problems, the LLM executed Mathematica scripts via the command line to perform the required algebraic elimination under human-specified requirements.

Because the underlying mathematics of the planar examples largely involves direct linear algebra or low-degree polynomial equations, code generation was comparatively straightforward. In these cases, the agent often produced runnable scripts and reports from a single structured prompt, with the human role concentrated on selecting the source material and checking the reported residuals and solution branches. This result should be interpreted as evidence of effective prompt-guided implementation for well-specified examples, not as evidence that the agent can solve arbitrary kinematics problems without expert review.

3.4. Evaluation Results

Table 1 summarizes the results across all problems in this category.

TABLE 1: Summary of planar and spherical linkage problems solved by LLM

Problem	Expected Solutions	LLM 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.)	∞ (curve/cone)	∞ (curve/cone)	Yes
Spherical RR (5 pos.)	up to 6	2 real, 4 complex	Yes

All LLM-generated solutions matched the textbook results within numerical precision ($< 10^{-10}$). The code was typically generated in a single structured prompt with minimal iteration, but informal prompts that omitted the source chapter, expected solution count, or residual checks required additional correction. All prompts, generated scripts, and test reports are publicly available in the GitHub repository [11], so the reported outcome can be reproduced and prompt sensitivity can be inspected directly.

Building upon these foundational problems, we now transition to solving two significantly more challenging kinematics problems: the synthesis of spatial SS dyads [7] and the forward kinematics of the general Stewart–Gough platform [1, 2]. Both are classical problems that have been solved previously, but they typically require substantially more analytical derivation and algebraic elimination efforts.

4. SYNTHESIS OF SPATIAL SS DYADS

4.1. Problem Statement

The goal of synthesis of spatial SS dyads is to find the fixed and moving pivots of a spherical dyad that satisfies a given set of spatial positions. This problem is often referred to as the spatial Burmester problem. Here we reproduce the seven-position spatial synthesis solver presented by Liao and McCarthy [7].

4.2. Kinematics Equations

Let $\mathbf{B} = (B_x, B_y, B_z)^T$ denote the fixed pivot and $\mathbf{p} = (p_x, p_y, p_z)^T$ denote the moving pivot. For 7 prescribed spatial positions defined by rotation matrices $[A_i]$ and translations $\mathbf{d}_i$, the SS chain constraint requires:

$$|[A_i]\mathbf{p} + \mathbf{d}_i - \mathbf{B}|^2 = r^2, \quad i = 1, \ldots, 7 \tag{2}$$

4.3. Solution Process

To eliminate the other unknowns to obtain a univariate polynomial in B_z, we take the following steps:

Subtracting pairs of equations eliminates r^2, $\mathbf{p}^T\mathbf{p}$, and $\mathbf{B}^T\mathbf{B}$, yielding 6 design equations linear in $\mathbf{p}$:

$$\mathbf{M}_i^T\mathbf{p} + N_i = 0, \quad i = 1, \ldots, 6 \tag{3}$$

where $\mathbf{M}_i$ and N_i are linear in $\mathbf{B}$. The 6 equations form a 6×4 augmented matrix C, and the requirement that every 4×4 submatrix be singular produces $\binom{6}{4} = 15$ determinantal constraints. Each is a degree-4 polynomial in (B_x, B_y, B_z).

Expanding in the 15 monomials of (B_x, B_y) up to total degree 4 yields a 15×15 coefficient matrix $\mathcal{M}(B_z)$:

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

The condition $\det(\mathcal{M}(B_z)) = 0$ yields a polynomial of effective degree 20 in B_z.

The numerical difficulty is caused by the structure of $\mathcal{M}(B_z)$ as a 15×15 matrix polynomial whose entries come from dependent quartic determinantal constraints. A naive determinant expansion has a nominal degree as high as 60 and contains a very large number of signed product terms, while the geometry of the problem cancels most high-degree contributions and leaves an effective degree of 20. In floating-point arithmetic, the large intermediate products and alternating cancellations can destroy the small residual coefficients that distinguish the true roots, which explains why direct symbolic expansion followed by numerical root finding recovered only part of the solution set.

This risk can be detected before expanding the determinant. Algorithmic warning signs include a large gap between the nominal determinant degree and the known geometric degree, rank deficiencies in the leading coefficient blocks $M_4, M_3, \ldots$, rapidly growing coefficient norms in trial expansions, and sensitivity of reconstructed determinant coefficients to working precision. In practice, one can form the matrix-polynomial coefficients exactly or at high precision, inspect the rank profile of the leading blocks, and compare roots obtained from determinant expansion at increasing precision against the roots of a companion linearization. These tests flag the expansion as ill-conditioned before relying on the scalar polynomial.

Therefore, rather than computing $\det(\mathcal{M}(B_z))$ symbolically as a degree-20 polynomial, we linearize the matrix polynomial $\mathcal{M}(B_z) = M_0 + M_1 B_z + M_2 B_z^2 + M_3 B_z^3 + M_4 B_z^4$ into a 60×60 companion matrix pencil:

$$\underbrace{\begin{pmatrix} 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ -M_0 & -M_1 & -M_2 & -M_3 \end{pmatrix}}_{\mathcal{A}} \mathbf{v} = B_z \underbrace{\begin{pmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & M_4 \end{pmatrix}}_{\mathcal{B}} \mathbf{v} \tag{5}$$

where I and 0 denote 15×15 identity and zero blocks. The finite real eigenvalues of $(\mathcal{A}, \mathcal{B})$ directly yield all roots of $\det(\mathcal{M}(B_z)) = 0$ without ever expanding the determinant. Each real B_z^* is then back-substituted via SVD to recover (B_x, B_y) and $\mathbf{p}$.

The complete procedure is summarized in Algorithm 1.

Algorithm 1 SS Chain 7-Position Synthesis

Require: Rotation matrices $[A_i]$ and translations $\mathbf{d}_i$, $i = 1, \ldots, 7$
Ensure: All real solution pairs $(\mathbf{p}, \mathbf{B}, L)$

```
Subtract position 7 to build 6 design equations M_i^T p + N_i = 0
Assemble 6 × 4 augmented matrix C(B)
Compute 15 = (6 4) minor determinants as degree-4 polynomials in (B_x, B_y, B_z)
Extract coefficients w.r.t. 15 (B_x, B_y) monomials → M(B_z) = Σ_{k=0}^{4} M_k B_z^k
Assemble 60 × 60 companion pencil (A, B) from Eq. (5)
Solve A v = λ B v for eigenvalues λ
Filter: keep finite real λ as candidate B_z values
for each real B_z^* do
    SVD of M(B_z^*) → null vector → extract B_x, B_y
    SVD of C(B_x, B_y, B_z^*) → null vector → extract p
    L ← |B − [A_1]p − d_1|
end for
return all (L, p, B) sorted by L
```

4.4. Code Generation by LLM

Initially, the LLM was provided with a general, high-level prompt to read the Liao & McCarthy [7] paper, generate the numerical solver code, and test it. After many iterations, the LLM repeatedly failed due to (1) misreading critical details in the paper, (2) misunderstanding the algebraic elimination process, and (3) attempting to solve for B_z by expanding $\det(\mathcal{M}(B_z)) = 0$ as a scalar polynomial. In these trials, the agent did not infer from the representation $\mathcal{M}(B_z) = \sum_{k=0}^{4} M_k B_z^k$ that the problem could be treated directly as a matrix-polynomial eigenvalue problem. The blocking reasoning step was recognizing that the determinant need not be formed: the roots can be obtained by companion linearization of the matrix polynomial, followed by null-space recovery of (B_x, B_y) and $\mathbf{p}$.

To resolve the numerical instability, the human expert explicitly directed the LLM to abandon determinant expansion and instead apply the companion-matrix generalized eigenvalue method, a numerical linear-algebra formulation not stated explicitly in the source paper. This debugging and diagnosis process required several hours. The expert-guided prompt specified: (1) the constraint equations and elimination procedure, (2) the companion matrix pencil structure, (3) the back-substitution via SVD, and (4) validation against the 20 known solutions from [7]. Both the initial broad prompt and the final detailed prompt are publicly available in the GitHub repository [12]. The final Mathematica script generated by the LLM is given in Appendix A.

4.5. Evaluation Results

Table 2 lists the 7 spatial positions used as input data, taken from Table 1 of [7]. Each position is defined by translations (d_x, d_y, d_z) and Euler angles θ (about y), $-\phi$ (about x), and ψ (about z), so that $[A_i] = R_y(\theta_i)\, R_x(-\phi_i)\, R_z(\psi_i)$.

Table 3 compares the two solver methods on the Liao–McCarthy seven-position data.

All 20 real solutions recovered by the eigenvalue solver are listed in Table 4. An independent planted-root validation confirmed the solver's correctness: the solver successfully recovered

TABLE 2: Input data: 7 spatial positions from [7]

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

TABLE 3: SS dyad solver comparison

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

the known planted solution as Solution 1. All 20 solutions satisfy the constant-distance constraint across all 7 frames to machine precision ($\sim 10^{-13}$). The final prompt, generated Mathematica script, and test report are available in the GitHub repository [12].

5. FORWARD KINEMATICS OF STEWART–GOUGH PLATFORM

5.1. Problem Statement

We reproduce Husty's algorithm [2] for the forward kinematics of a general 6–6 Stewart–Gough platform. Given 6 leg lengths connecting 6 base points to 6 platform points, the FK problem seeks all assembly configurations of the moving platform. Raghavan [1] showed that the general platform has at most 40 configurations.

5.2. Kinematics Equations

Husty [2] utilizes Study's kinematic mapping to project spatial displacements into a seven-dimensional homogeneous quasi-elliptic space characterized by eight Study parameters $(X_1, X_2, X_3, X_4, x_1, x_2, x_3, x_4)$ constrained to the Study quadric S_6^2:

$$X_1x_1 + X_2x_2 + X_3x_3 + X_4x_4 = 0 \tag{6}$$

The motion of each leg constrains a platform point to move on a sphere relative to the base, yielding six constraint manifolds $C_1, \ldots, C_6$. To reduce the system, a parametric subspace substitution is introduced:

$$x_1 = k\,x_4, \quad x_2 = m\,x_4, \quad x_3 = p\,x_4 \tag{7}$$

Substituting into the constraint equations generates seven constraint vectors $R_1, \ldots, R_7$, each containing six coefficients that govern five unknowns grouped as $\mathbf{y} = [X_1, X_2, X_3, X_4, \Phi]^T$, where Φ captures the nonlinear homogenized remainder. The system takes the form:

$$\sum_{j=1}^{5} R_{i,j}\,\mathbf{y}_j + R_{i,6} = 0, \quad i = 1, \ldots, 7 \tag{8}$$

For Husty's numerical example (Eq. 23 of [2]), the explicit constraint vectors are:

$$\begin{aligned}
R_1 &= \{0,\ 0,\ 0,\ 0,\ -1 - k^2 - m^2 - p^2,\ 1\}\\
R_2 &= \{8{+}8m{+}8p,\ -8{-}8k{-}8p,\ 8{-}8k{+}8m,\ -8k{+}8m{-}8p,\\
&\quad 11{+}11k^2{+}16m{-}16km{+}11m^2{+}16p{+}16kp{+}11p^2,\ 1\}\\
R_3 &= \{4{-}4m{+}12p,\ -4{+}4k{+}4p,\ 4{-}12k{-}4m,\ -4k{+}4m{-}4p,\\
&\quad -1{-}8k{+}7k^2{-}4m{+}4km{-}m^2{+}4p{+}4kp{-}8mp{+}7p^2,\ 1\}\\
R_4 &= \{4m{+}4p,\ 4{-}4k,\ 4{-}4k,\ -4m{-}4p,\\
&\quad 1{-}4k{+}k^2{+}m^2{+}4mp{+}p^2,\ 1\}\\
R_5 &= \{-20m{+}4p,\ -4{+}20k,\ 4{-}4k,\ 4m{-}4p,\\
&\quad 1{-}12k{+}25k^2{+}25m^2{-}12mp{+}p^2,\ 1\}\\
R_6 &= \{-8{+}12m,\ -16{-}12k{-}8p,\ -4{+}8m,\ 8k{+}16m{+}4p,\\
&\quad 13{+}24k{+}5k^2{-}16m{+}16km{+}21m^2{+}16p{+}16kp{+}8mp{-}3p^2,\ 1\}\\
R_7 &= \{k,\ m,\ p,\ 1,\ 0,\ 0\}
\end{aligned}$$

The geometric objective is to algebraically eliminate both the spatial variables $\mathbf{y}$ and the coupled parameters $\{k, m, p\}$ to obtain a single characteristic 40^{th}-degree polynomial purely in p.

5.3. Solution Process

The derivation proceeds through the following elimination steps:

1. **Linear determinants (G_i):** Construct five 5×6 matrices M_i from distinct subsets of the R vectors and compute their determinants $G_i(k, m, p) = \det(M_i) = 0$.

2. **Null space extraction:** Form the difference system $\Delta_i = R_{i+1} - R_1$ and extract the null space $\mathbf{y} \propto \text{Null}(A)$ to obtain parameterized solutions for $X_1, \ldots, X_4$. Note: the matrix is rank 3 (not 4), requiring adjugate-based extraction rather than standard Gaussian elimination.

3. **High-order substitution (G_7):** Substitute the null space expressions back into the primary constraint C_1 to obtain $G_7(k, m, p) = 0$.

4. **Resultant cascades:** Employ Sylvester resultants pairwise to eliminate k:

$$H_{ij}(m, p) = \text{Res}(G_i, G_j, k) = 0 \tag{9}$$

then eliminate m:

$$K_l(p) = \text{Res}(H_{ab}, H_{cd}, m) = 0 \tag{10}$$

5. **Polynomial extraction:** Compute the GCD of the resulting K_l polynomials to isolate the 40^{th}-degree characteristic polynomial $\mathcal{P}_{40}(p) = 0$.

The 40 roots of $\mathcal{P}_{40}(p)$ over the complex field define the algebraic candidates for assembly configurations; real, physically valid platform poses are obtained only after back-substitution and constraint checking. Back-substitution through the elimination

TABLE 4: All 20 SS dyad solutions (sorted by chain length L)

Sol	Length L	Moving pivot $\mathbf{p}$	Fixed pivot $\mathbf{B}$
1	2.862	(1.361, 0.085, −1.028)	(−1.453, −0.413, −1.178)
2	3.379	(2.144, −0.927, −0.102)	(−0.376, −0.269, −2.255)
3	3.435	(1.629, 1.837, −1.746)	(−0.405, −0.884, −1.240)
4	3.721	(2.357, 0.664, 0.246)	(0.810, −0.974, −2.716)
5	4.287	(−1.556, 1.152, 0.233)	(0.261, 2.459, −3.424)
6	4.395	(−0.978, 1.062, 0.436)	(0.974, 2.907, −3.042)
7	4.401	(0.161, −0.635, 2.478)	(−0.148, 2.679, −0.401)
8	4.465	(−0.646, 4.142, 0.906)	(1.280, 0.716, −1.214)
9	4.636	(0.703, −0.322, −0.656)	(−3.421, −0.294, 1.462)
10	7.945	(0.876, 2.477, 2.777)	(−3.820, −3.726, 4.385)
11	9.250	(−0.303, 0.205, −0.922)	(−2.561, −4.158, −8.760)
12	10.295	(−1.325, −7.207, 5.071)	(−4.071, −2.560, −3.697)
13	10.984	(3.459, 3.352, −9.664)	(0.899, −0.907, 0.131)
14	13.402	(0.367, −0.588, −0.934)	(−7.839, −0.089, 9.649)
15	15.815	(−0.471, 1.584, −1.981)	(−7.735, −9.633, −10.438)
16	25.709	(−5.393, 3.202, 25.079)	(0.073, −0.561, 0.241)
17	38.078	(5.484, −5.092, 14.718)	(−3.244, −34.968, −7.218)
18	75.962	(−0.068, 5.145, −4.517)	(−48.953, −37.551, −43.981)
19	86.022	(51.331, 26.929, −62.155)	(−7.967, 2.518, −4.817)
20	193.612	(−43.310, −113.557, −109.956)	(75.542, 37.613, −87.432)

chain recovers the Study parameters and the platform pose. The full polynomial is given in Appendix B.

The working implementation preserves Husty's elimination order and uses exact integer/rational arithmetic through the determinant, resultant, and GCD stages. Floating-point CAS arithmetic may be adequate for evaluating the final polynomial and refining roots, but it is not reliable for the symbolic elimination itself because small roundoff perturbations can create spurious factors, obscure common divisors, or change the recovered degree. Exact arithmetic, or at minimum controlled high-precision arithmetic with independent degree and root checks, is therefore required for reliable reproduction of the degree-40 polynomial.

Algorithm 2 Forward Kinematics of Stewart–Gough Platform [2]

Require: 6 leg lengths $L_1, \ldots, L_6$ and platform geometry
Ensure: Algebraic candidates for up to 40 assembly configurations
1: Formulate 6 constraint equations $C_1, \ldots, C_6$ on Study quadric S_6^2
2: Introduce parameterization: $x_1 = kx_4, x_2 = mx_4, x_3 = px_4$
3: Group terms into 7 constraint vectors $R_1, \ldots, R_7$
4: Eliminate $[X_1, X_2, X_3, X_4, \Phi]^T$ to find null space vector L_x
5: Substitute L_x into C_1 to obtain polynomial $G_7(k, m, p) = 0$
6: Compute $H_{ij}(m, p) = \mathrm{Res}(G_i, G_j, k)$
7: Compute $K_l(p) = \mathrm{Res}(H_{ab}, H_{cd}, m)$
8: Extract 40th-degree polynomial $\mathcal{P}_{40}(p) = \mathrm{GCD}(K_1, K_2)$
9: Compute up to 40 complex roots for p and retain real, physically valid candidates
10: Back-substitute p through resultant cascades to find (m, k)
11: Recover (X_1, X_2, X_3, X_4) and reconstruct full platform pose
12: **return** All real assembly configurations

5.4. Code Generation by LLM

This problem represents a substantial challenge for classical algebraic geometry, requiring extensive symbolic computation in Mathematica. The prompt specified: (1) the Study quadric parameterization and leg length constraints, (2) the step-by-step elimination strategy via Sylvester resultants, (3) the expectations for the degree-40 polynomial, and (4) validation using Husty's published numerical example in [2] with the known planted root $p = -1$.

For this problem, the AI agent succeeded when the prompt closely followed Husty's published elimination sequence. It generated a functionally complete Mathematica script that derived the 40th-degree polynomial and verified the known planted root $p = -1$. The entire process of crafting the prompt and generating the working code took approximately 30 minutes, indicating that LLMs can translate a well-specified symbolic algorithm into CAS code effectively. However, this should not be interpreted as robustness to arbitrary reformulations. Trial prompts that changed the elimination order caused substantially larger intermediate expressions and did not reliably recover the degree-40 polynomial within the same workflow, even though the underlying algebraic problem is equivalent. The final Mathematica script generated by the LLM is provided in Appendix B. All prompts, generated scripts, and test reports are publicly available in the GitHub repository [13]. Table 5 summarizes the FK solver results.

TABLE 5: Stewart–Gough FK solver evaluation

Metric	Value
Polynomial degree	40
Known root $p = -1$ recovered	Yes
CAS used	Mathematica
Arithmetic during elimination	Exact integer/rational
Alternative elimination order	Expression growth; not reliable in this workflow
Derivation runtime	∼30 min

6. CONCLUSIONS

This paper presented a practical guide for using large language models to assist in generating analytical and numerical solvers for selected classical kinematics problems, specifically planar and spherical linkages, spatial SS dyads, and the forward kinematics of Stewart–Gough platforms. The examples reveal distinct levels of useful agent assistance based on mathematical complexity, but they should be read as case studies rather than a statistically controlled benchmark.

Planar and spherical linkage problems were solved with minimal iteration when the prompts supplied the relevant source mate-

rial, expected solution counts, and residual checks. The Stewart–Gough FK problem demonstrated the LLM's ability to translate an established symbolic algorithm into computer algebra logic when the elimination order and exact-arithmetic requirements were specified.

Conversely, the spatial SS dyad synthesis problem illustrated a scenario requiring expert intervention. While the LLM produced draft solvers quickly, an expert was needed to diagnose catastrophic cancellation, recognize the matrix-polynomial structure, and direct the LLM toward a companion-matrix generalized eigenvalue formulation.

The key takeaways for practitioners are:

1. **LLMs accelerate code generation**: LLMs can generate working kinematics solver code quickly when guided by structured prompts encoding problem-specific strategies and validation tests.

2. **Problem complexity dictates interaction**: Simpler and symbolically straightforward problems may be solved in a single structured prompt, while complex polynomial systems involving nuanced numerical methods require multi-step prompting and iterative expert correction.

3. **Numerical method selection is critical**: Practitioners should explicitly prompt the LLM to use robust methods, such as eigenvalue-based solvers or exact-arithmetic elimination, and should verify the result using residuals, known roots, solution counts, and precision-sensitivity checks.

The workflow can generalize beyond polynomial and closed-form kinematics, but the validation target changes. For non-polynomial models, compliant mechanisms, contact-rich mechanisms, or systems described only by simulation, the LLM should be prompted to build numerical residual functions, continuation or homotopy sweeps, interval or multi-start searches, automatic-differentiation Jacobians, and regression tests against known configurations rather than to derive a closed-form characteristic polynomial. In such settings the evidence should emphasize reproducibility, convergence behavior, constraint residuals, sensitivity to initial guesses and prompts, and failure modes. The present paper does not claim that LLMs can discover globally complete solution sets for these broader problems without additional numerical analysis.

These results demonstrate that LLMs are useful tools for kinematics researchers when embedded in a validation-centered human–AI workflow. Future work could compare multiple agents under identical prompts, report runtime and prompt-sensitivity statistics more systematically, and extend the templates to problems where exact algebraic elimination is unavailable.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN THE MANUSCRIPT PREPARATION PROCESS

During the preparation of the reported solver results, the authors used OpenAI Codex 5.3 as the primary coding agent to generate solver code and assist with symbolic derivations. Anthropic Claude Opus 4.6 and Google Gemini 3.1 Pro (Thinking) were considered as available coding-agent interfaces for the workflow, but their outputs are not reported as a comparative evaluation in this paper. The authors supplied the problem formulations, source materials, prompts, and validation criteria; reviewed and edited all generated content; and take full responsibility for the published article.

REFERENCES

[1] Raghavan, Madhavan. "The Stewart platform of general geometry has 40 configurations." *Journal of Mechanical Design* Vol. 115 No. 2 (1993): pp. 277–282.

[2] Husty, Manfred L. "An algorithm for solving the direct kinematics of general Stewart-Gough platforms." *Mechanism and Machine Theory* Vol. 31 No. 4 (1996): pp. 365–380.

[3] OpenAI. "GPT-4 Technical Report." Technical report no. OpenAI. 2023. URL https://cdn.openai.com/papers/gpt-4.pdf.

[4] OpenAI. "OpenAI GPT-5 System Card." *arXiv preprint arXiv:2601.03267* (2025)URL https://arxiv.org/abs/2601.03267.

[5] Su, Hai-Jun. "Large language model assisted human-AI collaborative development of analytical inverse kinematics solvers for robot manipulators." *Mechanism and Machine Theory* Vol. 222 (2026): p. 106392. DOI https://doi.org/10.1016/j.mechmachtheory.2026.106392. URL https://www.sciencedirect.com/science/article/pii/S0094114X26000431.

[6] McCarthy, J Michael and Soh, Gim Song. *Geometric design of linkages*. Springer Science & Business Media (2010).

[7] Liao, Q. and McCarthy, J. M. "Seven-position synthesis of a spatial SS chain." *Journal of Mechanical Design* Vol. 123 No. 1 (2001): pp. 74–79.

[8] OpenAI. "OpenAI Codex 5.3." (2025). https://openai.com/codex.

[9] Anthropic. "Claude 4.6 Opus." (2025). https://www.anthropic.com.

[10] Google. "Gemini 3.1 Pro (Thinking)." (2025). https://deepmind.google/technologies/gemini/.

[11] Su, Haijun et al. "LLM Reproduction of Geometric Design of Linkages." https://github.com/haijunsu-osu/gdl_llm_reproduction (2026).

[12] Su, Haijun et al. "LLM Code Generation for Seven-Position Synthesis of Spatial SS Chains." https://github.com/haijunsu-osu/llm_ss_synthesis_liao_2001 (2026).

[13] Su, Haijun et al. "LLM Code Generation for Forward Kinematics of Stewart Platform." https://github.com/haijunsu-osu/llm_FK_Stewart_Husty_1996 (2026).

APPENDIX A. LLM-GENERATED MATHEMATICA SCRIPT FOR SS DYAD SYNTHESIS

The following Mathematica script was generated by the LLM to solve the 7-position SS chain synthesis problem via the generalized eigenvalue companion matrix formulation.

```
(* SS Chain 7-Position Synthesis Solver *)
(* Based on: Liao & McCarthy (2001) *)
ClearAll["Global`*"];

(* Elementary rotation matrices *)
rotY[t_] := {{Cos[t],0,Sin[t]},{0,1,0},{-Sin[t],0,Cos[t]}};
rotX[t_] := {{1,0,0},{0,Cos[t],-Sin[t]},{0,Sin[t],Cos[t]}};
rotZ[t_] := {{Cos[t],-Sin[t],0},{Sin[t],Cos[t],0},{0,0,1}};

(* Paper Table 1 data *)
dxs = Rationalize[{0,1,0.3182,-0.1788,-1.258,-3.5939,-0.0497},0];
dys = Rationalize[{0,-0.7423,-0.5085,-1.7842,0.8362,2.7283,0.57},0];
dzs = Rationalize[{0,-0.1337,-0.7922,-1.0429,-1.4992,-2.0334,-1.4858},0];
thetas = Rationalize[{0,6.1802,-83.2605,-170.031,-84.7359,
  -8.30385,118.266},0]*Pi/180;
minusphis = Rationalize[{0,4.27866,-18.2345,39.5378,-29.1752,
  5.0361,-33.7953},0]*Pi/180;
psis = Rationalize[{0,-97.9255,73.6069,-50.9407,150.331,
  -68.2528,139.022},0]*Pi/180;

A = Table[rotY[thetas[[i]]].rotX[-minusphis[[i]]].rotZ[psis[[i]]],
  {i,1,7}];
d = Table[{dxs[[i]],dys[[i]],dzs[[i]]},{i,1,7}];
A = N[A,30]; d = N[d,30];

(* STEP 1: Build 6 design equations M_i^T p + N_i = 0 *)
Bvec = {Bx,By,Bz};
Mvecs = Table[-2 Transpose[A[[i]]-A[[7]]].Bvec +
  2(Transpose[A[[i]]].d[[i]]-Transpose[A[[7]]].d[[7]]),{i,1,6}];
Nvals = Table[-2(d[[i]]-d[[7]]).Bvec +
  d[[i]].d[[i]]-d[[7]].d[[7]],{i,1,6}];

(* STEP 2: Build 6x4 augmented matrix C, extract 15 minors *)
Cmat = Table[Append[Mvecs[[i]],Nvals[[i]]],{i,1,6}];
polys = Flatten[Minors[Cmat,4]];

(* STEP 3: Build 15x15 coefficient matrix M(Bz) *)
monomialDegrees = {{4,0},{3,1},{2,2},{1,3},{0,4},
  {3,0},{2,1},{1,2},{0,3},{2,0},{1,1},{0,2},{1,0},{0,1},{0,0}};
getCoef[poly_,{px_,py_}] :=
  Coefficient[Coefficient[Expand[poly],Bx,px],By,py];
matrixM = Table[getCoef[polys[[i]],monomialDegrees[[j]]],
  {i,1,15},{j,1,15}];

(* STEP 4: Solve det(M(Bz))=0 via 60x60 eigenvalue pencil *)
M0=matrixM/.Bz->0; M1=Coefficient[matrixM,Bz,1];
M2=Coefficient[matrixM,Bz,2]; M3=Coefficient[matrixM,Bz,3];
M4=Coefficient[matrixM,Bz,4];
id=IdentityMatrix[15]; z15=ConstantArray[0,{15,15}];
matA=ArrayFlatten[{{z15,id,z15,z15},{z15,z15,id,z15},
  {z15,z15,z15,id},{-M0,-M1,-M2,-M3}}];
matB=ArrayFlatten[{{id,z15,z15,z15},{z15,id,z15,z15},
  {z15,z15,id,z15},{z15,z15,z15,M4}}];
vals=Eigenvalues[{matA,matB}];
realBz=Sort[Re[Select[Select[vals,
  NumericQ[#]&&Abs[#]<10^8&],Abs[Im[#]]<10^-5&]]];

(* STEP 5: Back-substitute Bz -> (Bx,By) -> p via SVD *)
results={};
For[k=1,k<=Length[realBz],k++, bz=realBz[[k]];
  Mnun=matrixM/.Bz->bz;
  {u,s,v}=SingularValueDecomposition[N[Mnun]];
  vnull=v[[All,-1]]; vnull=vnull/vnull[[-1]];
  bx=vnull[[13]]; by=vnull[[14]];
  Cnun=Cmat/.{Bx->bx,By->by,Bz->bz};
  {uc,sc,vc}=SingularValueDecomposition[N[Cnun]];
  pnull=vc[[All,-1]]; pnull=pnull/pnull[[-1]];
  pvec={pnull[[1]],pnull[[2]],pnull[[3]]};
  Bpt={bx,by,bz};
  len=Norm[Bpt-(A[[1]].pvec+d[[1]])];
  AppendTo[results,{len,pvec,Bpt}]];
Print[Length[results]," solutions found."];
```

APPENDIX B. STEWART–GOUGH FORWARD KINEMATICS

B.1. 40th-Degree Characteristic Polynomial

The resultant elimination cascades yield the following 40th-degree polynomial $\mathcal{P}_{40}(p) = 0$, corresponding to Husty's numerical example (Eq. 23 of [2]). The polynomial is presented with exact integer coefficients:

$$\mathcal{P}_{40}(p) = c_0 + c_1 p + c_2 p^2 + \cdots + c_{40} p^{40} = 0$$

where the coefficients c_i are:

$$\begin{aligned}
c_0 &= 551863002346251447441190477490269750822902072981061\\
c_1 &= 955242753801639574466167697472402349003645364069978 8\\
c_2 &= 805819544871669691046861227996008872132848594564010 64\\
c_3 &= 431189931236119711364249046798947555141839421741150004\\
c_4 &= 163470328202608538763666264760291883422505113729513 5234\\
c_5 &= 469702960014419757074510244183751492131925248368445 3276\\
c_6 &= 107518595976532657181001414008772304366147258352541768 88\\
c_7 &= 204267602304866722393838820628236807968217770052064042 44\\
c_8 &= 332885549383538450366196452670908444287789393560991781 41\\
c_9 &= 475056218759498735833505292893745731373182245913822721 12\\
c_{10} &= 595779285311232613501660139743077253496261611841838144 16\\
c_{11} &= 648996678782763930830813622717513178357865382148882599 84\\
c_{12} &= 605572585153314189900941706530496655012407443358976998 96\\
c_{13} &= 491616820880633148516162052718674091784825679053804369 44\\
c_{14} &= 389157181442266673251760213167751174536872116006179787 68\\
c_{15} &= 373423446236963966148159616920166341771291262482085063 52\\
c_{16} &= 439875550282732547084299964766613822619656761086898460 26\\
c_{17} &= 504557419005180568506171093024568700094530867775975953 04\\
c_{18} &= 484035880316655777046111490518167984523566061400485764 16\\
c_{19} &= 369151843298803428131303133984688819246690082660897748 08\\
c_{20} &= 219557408589722703829953220949232907327005848108875508 92\\
c_{21} &= 100459005713667140063629386648918493744545504324655197 84\\
c_{22} &= 356556783607281028659649298270349601307046955875763808 0\\
c_{23} &= 114493753791739577533103870569633282184388143864306508 0\\
c_{24} &= 529537062618969837385020125131389278204869445785093154\\
c_{25} &= 368202118785263954590411745279572511387964969608181552\\
c_{26} &= 245174056263421431450333504278264415670901008913627120\\
c_{27} &= 129206387273840734949141656780161168639896549834400976\\
c_{28} &= 483316331750819875087982079387153441428089398359598 00\\
c_{29} &= 966921348843016362560556249754804228965795184423444 8\\
c_{30} &= -150632389349832812940579279051242505297678581396240\\
c_{31} &= 771929045738980443971945980003467850732615405339120\\
c_{32} &= 202293000275613980586220646441009092640290148715336 1\\
c_{33} &= 169530519922066718530276453083990400779973706394270 0\\
c_{34} &= 913459756655347881431549040458270272574071838336184\\
c_{35} &= 372133115268542194673635230322975653714920509596564\\
c_{36} &= 113106818692008598970794074255654810918177731939874\\
c_{37} &= 221321171702186461122120177818551438603320558978 84\\
c_{38} &= 202032573666664380729760461364479083983003943508 0\\
c_{39} &= 483730556277193198454230495885061864567754613 2\\
c_{40} &= 117579862251147904747865456218984043461615184 1
\end{aligned}$$

The root $p = -1$ can be verified as an exact root of this polynomial, confirming one of the 40 assembly configurations of the Stewart–Gough platform.

B.2. LLM-Generated Mathematica Script

The following Mathematica script was generated by the LLM to reproduce Husty's forward kinematics algorithm. It uses exact rational arithmetic throughout the elimination pipeline and validates the result against the published numerical example.

```
(* Husty 1996 Stewart-Gough FK reproduction *)
(* Uses exact integer/rational arithmetic through the elimination pipeline *)
ClearAll["Global`*"];

contentInt[poly_, vars_] :=
```

```
  PolynomialGCD @@ (Last /@ CoefficientRules[poly, vars]);

primitivePoly[poly_, vars_] := Module[{c, out, nz},
  c = contentInt[poly, vars];
  out = Expand[poly/c];
  nz = Select[Last /@ SortBy[CoefficientRules[out, vars], First],
              # =!= 0 &];
  If[nz =!= {} && First[nz] < 0, out = -out]; out];

Q = 1 + k^2 + m^2 + p^2;

(* Eq. (23) constraint vectors *)
R1 = {0, 0, 0, 0, -Q, 1};
R2 = {8+8*m+8*p, -8-8*k-8*p, 8-8*k+8*m, -8*k+8*m-8*p,
  11+11*k^2+16*m-16*k*m+11*m^2+16*p+16*k*p+11*p^2, 1};
R3 = {4-4*m+12*p, -4+4*k+4*p, 4-12*k-4*m, -4*k+4*m-4*p,
  -1-8*k+7*k^2-4*m+4*k*m-m^2+4*p+4*k*p-8*m*p+7*p^2, 1};
R4 = {4*m+4*p, 4-4*k, 4-4*k, -4*m-4*p,
  1-4*k+k^2+m^2+4*m*p+p^2, 1};
R5 = {-20*m+4*p, -4+20*k, 4-4*k, 4*m-4*p,
  1-12*k+25*k^2+25*m^2-12*m*p+p^2, 1};
R6 = {-8+12*m, -16-12*k-8*p, -4+8*m, 8*k+16*m+4*p,
  13+24*k+5*k^2-16*m+16*k*m+21*m^2+16*p+16*k*p+8*m*p-3*p^2, 1};
R7 = {k, m, p, 1, 0, 0};

RFull = {R1, R2, R3, R4, R5, R6, R7};
systems = {{1,2,3,4,5,7}, {1,2,3,4,6,7}, {1,2,3,5,6,7},
           {1,2,4,5,6,7}, {1,3,4,5,6,7}};

normalizeGi[poly_] := Module[{q, r},
  {q, r} = PolynomialQuotientRemainder[Expand[poly], Q, k];
  primitivePoly[q, {k, m, p}]];

(* Step 1: G1..G5 from determinants *)
G = normalizeGi /@ (Det[RFull[[#]]] & /@ systems);

(* Step 2: H12 -- eliminate k between G1,G2 *)
res12 = primitivePoly[Expand[Resultant[G[[1]], G[[2]], k]], {m, p}];
f12 = FactorList[res12];
h12F = Select[Rest[f12],
  Exponent[#[[1]], m]==10 && Exponent[#[[1]], p]==10 &];
H12 = primitivePoly[Times @@ (#[[1]]^#[[2]] & /@ h12F), {m, p}];

(* Step 3: Null space (adjugate) for Lx -- Eq. 26 *)
A1 = {R2[[1;;5]]-R1[[1;;5]], R3[[1;;5]]-R1[[1;;5]],
      R4[[1;;5]]-R1[[1;;5]], R5[[1;;5]]-R1[[1;;5]], R7[[1;;5]]};
LX = -Adjugate[A1][[All, 4]] // Expand;

(* Step 4: G7 from C1 substitution -- Eq. 27 *)
G7raw = Expand[(R1[[1;;5]].LX)*LX[[5]]
        + 4*Total[LX[[1;;4]]^2]];
{G7q, G7r} = PolynomialQuotientRemainder[G7raw, Q, k];
G7 = primitivePoly[G7q, {k, m, p}];

(* Step 5: Eliminate k from G1*G7, G2*G7 *)
H17 = primitivePoly[Expand[Resultant[G[[1]], G7, k]], {m, p}];
H27 = primitivePoly[Expand[Resultant[G[[2]], G7, k]], {m, p}];

(* Step 6: Eliminate m, extract degree-40 polynomial *)
K1 = primitivePoly[Expand[Resultant[H12, H17, m]], {p}];
K2 = primitivePoly[Expand[Resultant[H12, H27, m]], {p}];
LG = primitivePoly[PolynomialGCD[K1, K2], {p}];

fl = FactorList[LG];
F39 = Select[Rest[fl], Exponent[#[[1]], p]==39 &][[1, 1]];
FinalP40 = primitivePoly[Expand[(1+p)*F39], {p}];

(* Validation: real roots *)
realPRoots = Sort[Select[
  p /. NSolve[FinalP40==0, p, WorkingPrecision->60],
  Abs[Im[#]] < 10^-30 &] // Re];
Print["Real roots: ", N[realPRoots, 12]];
Print["Degree of FinalP40: ", Exponent[FinalP40, p]];
```