Numerical Continuation on a Graphical Processing Unit for Kinematic Synthesis

Jeffrey Glabe
Robotics and Automation Laboratory,
University of California,
Irvine, CA 92697
e-mail: jglabe@uci.edu

J. Michael McCarthy
Robotics and Automation Laboratory,
University of California,
Irvine, CA 92697
e-mail: jmmccart@uci.edu

This paper presents an implementation of a homotopy path tracking algorithm for polynomial numerical continuation on a graphical processing unit (GPU). The goal of this algorithm is to track homotopy curves from known roots to the unknown roots of a target polynomial system. The path tracker solves a set of ordinary differential equations to predict the next step and uses a Newton root finder to correct the prediction so the path stays on the homotopy solution curves. In order to benefit from the computational performance of a GPU, we organize the procedure so it is executed as a single instruction set, which means the path tracker has a fixed step size and the corrector has a fixed number iterations. This trade-off between accuracy and GPU computation speed is useful in numerical kinematic synthesis where a large number of solutions must be generated to find a few effective designs. In this paper, we show that our implementation of GPU-based numerical continuation yields 85 effective designs in 63 s, while an existing numerical continuation algorithm yields 455 effective designs in 2 h running on eight threads of a workstation.
[DOI: 10.1115/1.4047240]

Keywords: computational synthesis, computer aided design, GPU computing for design and manufacturing

1 Introduction

Freudenstein and Roth [1] described what they called a parameter-perturbation procedure to compute the solution of a set of non-linear equations starting from a known solution of another similar set of equations. They used this technique to solve the synthesis equations for a four-bar linkage that guides a coupler point through nine specified task positions [2]. Wampler et al. [3] returned to this nine point synthesis problem with what they called numerical polynomial continuation and identified Roth and Freudenstein's approach as a type of numerical continuation.

Zangwill and Garcia [4] described a wide range of problems that can be solved by tracking the paths of a homotopy, such as nonlinear programming, economic equilibria, and game theory. Morgan [5] applied this approach to finding all of the solutions of a system of polynomial equations. Tsai and Morgan [6] used numerical polynomial continuation to solve for the inverse kinematics of a general 6R robot, which was an important outstanding problem at the time.

Since then numerical continuation has improved in capabilities, see Ref. [7]. Now a variety of software packages for numerical continuation are available, such as Bertini [8], PHCpack [9], HOM4PS [10], and POLSYS_GLP [11].

Homotopy solution of a system of polynomials follows the known roots of a starting polynomial system as its coefficients are smoothly changed into the coefficients of the target polynomial system [7]. The process of following the transformation of these known roots into the roots of the target system is known as path tracking and can be parallelized for distributed computation [11].

Graphics processing units (GPUs) were developed to accelerate rendering calculations in computer graphics [12]. These devices execute the same instruction set for each pixel in a display at very high speed. This capability has been deployed in other applications where identical sets of instructions are executed for a large number of cases. Examples are computational fluid mechanics [13], robot motion planning [14,15], and deformable body modeling for computer graphics [16].

In contrast to parallelization on a cluster of central processing units (CPUs), parallelization on a GPU requires the execution of identical instruction sets on a collection of threads known as a warp to ensure maximum performance. Verschelde and Yoffe [17] introduced the use of a GPU for polynomial homotopy, using it to evaluate the polynomials and their derivatives with extended precision mathematics. The result was speeds of almost 20 times the speed of computation on a single CPU.

In order to increase the performance for path tracking on a GPU in a numerical polynomial continuation solver, a strategy to manage changes in step size is needed, because the execution of a conditional statement in one thread can pause the computation in other threads of a warp until the conditional is completed. Verschelde and Yu [18,19] manage this by providing three levels of adaptive step sizes for tracking the tens of thousands solution paths in GPU-based polynomial homotopy solver as well as using higher precision to more accurately track paths.

This paper presents a different strategy for path tracking using a GPU in a polynomial continuation solver, which avoids changes in step size. Paths for which the desired accuracy has not yet been achieved are grouped and computed with a smaller step size. This approach is well-adapted to the need for high-speed path tracking of many thousands of paths needed to solve the design equations for the kinematic synthesis of mechanisms, see for example Ref. [20].

2 Path Tracking

Numerical polynomial continuation obtains the roots of a system of n polynomials, $\mathcal{P}(\mathbf{x}) = (p_1, p_2, \ldots, p_n)$ in n unknowns $\mathbf{x} \in \mathbb{C}^{nx1} = (x_1, x_2, \ldots, x_n)$ by starting with a system $\mathcal{S}(\mathbf{x})$ that has the same total degree, M. The start system $\mathcal{S}(\mathbf{x})$ is constructed so that these roots have known values $\mathbf{y}_k$, $k = 1, \ldots, M$, that is

$$\mathcal{S}(\mathbf{y}_k) = \begin{Bmatrix} s_1(\mathbf{y}_k) \\ s_2(\mathbf{y}_k) \\ \vdots \\ s_n(\mathbf{y}_k) \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}, \quad k = 1, \ldots, M \tag{1}$$

Manuscript received September 24, 2019; final manuscript received May 9, 2020; published online June 12, 2020. Assoc. Editor: Anurag Purwar.

The M roots $\bar{\mathbf{x}}_k$ of $\mathcal{P}(\mathbf{x})$ are obtained by smoothly transforming the roots of $\mathcal{S}$ into those of $\mathcal{P}$ and tracking the paths $\mathbf{v}_k(t) = (v_{1k}, v_{2k}, \ldots, v_{nk})$ as t varies from 1 to 0; the initial positions of the roots are $\mathbf{y}_k = \mathbf{v}_k(1)$ and their final positions are $\bar{\mathbf{x}}_k = \mathbf{v}_k(0)$. Note that while $\mathcal{S}(\mathbf{x})$ starts with M roots, the target system $\mathcal{P}(\mathbf{x})$ may have fewer than M roots. The number of roots is $\mathcal{P}(\mathbf{x})$ is upper bounded by M.

The homotopy $\mathcal{H}(\mathbf{x}, t)$ that transforms $\mathcal{S}$ into $\mathcal{P}$ is given by the polynomial system,

$$\mathcal{H}(\mathbf{x}, t) = \mathcal{S}(\mathbf{x})\, t + \mathcal{P}(\mathbf{x})\,(1 - t) \tag{2}$$

A path $\mathbf{v}_k(t)$ exists for each root of the start system and is a solution of the polynomial system $\mathcal{H}(\mathbf{x}, t)$, that is

$$\mathcal{H}(\mathbf{v}_k, t) = \begin{Bmatrix} s_1(\mathbf{v}_k)t + p_1(\mathbf{v}_k)(1-t) \\ s_2(\mathbf{v}_k)t + p_2(\mathbf{v}_k)(1-t) \\ \vdots \\ s_n(\mathbf{v}_k)t + p_n(\mathbf{v}_k)(1-t) \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}, \quad k = 1, \ldots, M \tag{3}$$

In order to describe a strategy to track these paths, consider the Taylor series expansion of the homotopy $\mathcal{H}(\mathbf{x}, t)$ in $\mathbf{x}$ and t, given by

$$\mathcal{H}(\mathbf{x} + \Delta\mathbf{x}, t + \Delta t) = \mathcal{H}(\mathbf{x}, t) + \frac{\partial \mathcal{H}}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial \mathcal{H}}{\partial t}\Delta t + \text{higher order terms} \tag{4}$$

Introduce the notation,

$$\mathbf{J}_\mathbf{x} = \frac{\partial \mathcal{H}}{\partial \mathbf{x}} \quad \text{and} \quad \mathbf{H}_t = \frac{\partial \mathcal{H}}{\partial t} \tag{5}$$

where $\mathbf{J}_\mathbf{x}$ is an $n \times n$ matrix and $\mathbf{H}_t$ an $n \times 1$ vector with elements J_{ij} and H_i, respectively, given by

$$J_{ij} = \frac{\partial s_i}{\partial x_j}\, t + \frac{\partial p_i}{\partial x_j}\,(1-t), \quad H_i = s_i - p_i, \qquad i, j = 1, \ldots, n \tag{6}$$

Now consider a point $(\mathbf{x}, t)$ that is sufficiently close to a path $\mathbf{v}(t)$ of the homotopy, such that $\mathcal{H}(\mathbf{x}, t) \approx 0$. Then, we can predict a new point $\mathbf{p} = \mathbf{x} + \Delta\mathbf{x}$ at $t + \Delta t$ by setting $\mathcal{H}(\mathbf{x} + \Delta\mathbf{x}, t + \Delta t) = 0$ and solving for the first-order terms of Eq. (4) to obtain

$$\mathbf{p} = \mathbf{x} - \mathbf{J}_\mathbf{x}^{-1}(\mathbf{x}, t)\mathbf{H}_t(\mathbf{x})\Delta t \tag{7}$$

This is the Davidenko equation and the solution is known as the "prediction" step of the path tracker and is achieved using a solver for ordinary differential equations.

If $\mathcal{H}(\mathbf{x}, t)$ is not sufficiently close to a path, then $\mathcal{H}(\mathbf{x}, t) \neq 0$, and we compute $\mathbf{c} = \mathbf{x} + \Delta\mathbf{x}$ for $\Delta t = 0$ such that $\mathcal{H}(\mathbf{x} + \Delta\mathbf{x}, t) \approx 0$. From Eq. (4), we obtain

$$\mathbf{c} = \mathbf{x} - \mathbf{J}_\mathbf{x}^{-1}(\mathbf{x}, t)\mathcal{H}(\mathbf{x}, t) \tag{8}$$

This is called the "correction" step of the path tracker and is an example of Newton's method for root finding. Path tracking executes a sequence of these prediction and correction steps to track the paths from $\mathcal{S}(\mathbf{x})$ to $\mathcal{P}(\mathbf{x})$.

3 Numerical Path Tracking

In this section, we present the series of computations that perform path tracking for numerical polynomial continuation that will be executed on a GPU. Because the initial step of the path tracker begins with a known root of the start system, we begin by predicting the next value using the Runge–Kutta–Fehlbreg method, see Ref. [21]. For convenience, introduce the notation $\mathbf{f}(\mathbf{x}, t)$ for the vector function in Eq. (7), so we have,

$$\Delta\mathbf{x} = -\mathbf{J}_\mathbf{x}^{-1}(\mathbf{x}, t)\mathbf{H}_t(\mathbf{x})\Delta t = \mathbf{f}(\mathbf{x}, t) \tag{9}$$

For step size Δt, the next prediction point $\mathbf{p}$ of a path can be calculated using the Runge–Kutta fourth-order formulas,

$$\mathbf{p}(t + \Delta t) = \mathbf{x}(t) + \frac{1}{6}(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4) \tag{10}$$

where

$$\begin{aligned} \mathbf{K}_1 &= \Delta t\, \mathbf{f}(\mathbf{x}, t) \\ \mathbf{K}_2 &= \Delta t\, \mathbf{f}\left(\mathbf{x} + \frac{1}{2}\mathbf{K}_1, t + \frac{1}{2}\Delta t\right) \\ \mathbf{K}_3 &= \Delta t\, \mathbf{f}\left(\mathbf{x} + \frac{1}{2}\mathbf{K}_2, t + \frac{1}{2}\Delta t\right) \\ \mathbf{K}_4 &= \Delta t\, \mathbf{f}(\mathbf{x} + \mathbf{K}_3, t + \Delta t) \end{aligned} \tag{11}$$

This calculation of $\mathbf{p}$ involves four evaluations of $\mathbf{f}$ for different arguments, each of which requires finding the inverse of the $n \times n$ Jacobian matrix $\mathbf{J}_\mathbf{x}$.

3.1 LU Decomposition. An effective algorithm for calculating the inverse of $\mathbf{J}_\mathbf{x}$ is known as Lower-Upper (LU) decomposition [21]. This is achieved by permuting $\mathbf{J}_\mathbf{x}$, so that it can be factored into the product of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$, that is

$$\mathbf{P}\mathbf{J}_\mathbf{x} = \mathbf{L}\mathbf{U} \tag{12}$$

where $\mathbf{P}$ is an $n \times n$ matrix that permutes the rows of $\mathbf{J}_\mathbf{x}$. Write Eq. (7) in the form,

$$\mathbf{J}_\mathbf{x}\Delta\mathbf{x} = -\mathbf{H}_t\,\Delta t \tag{13}$$

and substitute the LU decomposition to obtain,

$$\mathbf{P}\mathbf{J}_\mathbf{x}\Delta\mathbf{x} = \mathbf{L}\mathbf{U}\Delta\mathbf{x} = -\mathbf{P}\mathbf{H}_t\,\Delta t \tag{14}$$

We solve this equation by introducing $\mathbf{z} = \mathbf{U}\Delta\mathbf{x}$ and use sequential elimination by rows to solve

$$\mathbf{L}\mathbf{z} = -\mathbf{P}\mathbf{H}_t\,\Delta t \tag{15}$$

for $\mathbf{z}$. Then, back-substitution is used to solve

$$\mathbf{U}\Delta\mathbf{x} = \mathbf{z} \tag{16}$$

for $\Delta\mathbf{x}$.

This solution $\Delta\mathbf{x}$ is used to calculate each of the four terms $\mathbf{K}_i$, $i = 1, 2, 3, 4$ in the Runge–Kutta calculation for $\mathbf{p}$.

3.2 Newton's Correction. We assume the calculation of the point $\mathbf{p}(t + \Delta t)$ along a path $\mathbf{v}(t)$ takes the point away from the homotopy hypersurface $\mathcal{H}(\mathbf{x}, t) = 0$, so we use Newton's method to find the nearby root $\mathbf{c}(t + \Delta t)$. Write Eq. (8) in the form,

$$\mathbf{J}_\mathbf{x}(\mathbf{p}, t + \Delta t)\Delta\mathbf{x} = \mathcal{H}(\mathbf{p}, t + \Delta t) \tag{17}$$

This equation can be solved using LU decomposition to calculate $\Delta\mathbf{x}$, which yields the correction,

$$\mathbf{c} = \mathbf{p} + \Delta\mathbf{x} \tag{18}$$

The usual implementation of a path tracker for numerical polynomial continuation iterates Newton's method to ensure convergence to the homotopy hypersurface. The correction step can be repeated multiple times until a desired level of convergence is achieved. The computation flow of the path-tracking method is shown in Fig. 1. Whether running on a CPU or a GPU, the path trackers use the same predictor corrector methods. However, executing a path tracker on a GPU presents additional challenges.

3.3 Graphical Processing Units Implementation of Path Tracking. A GPU consists of an array of streaming multiprocessors (SMs), each of which is analogous to the core of a CPU.

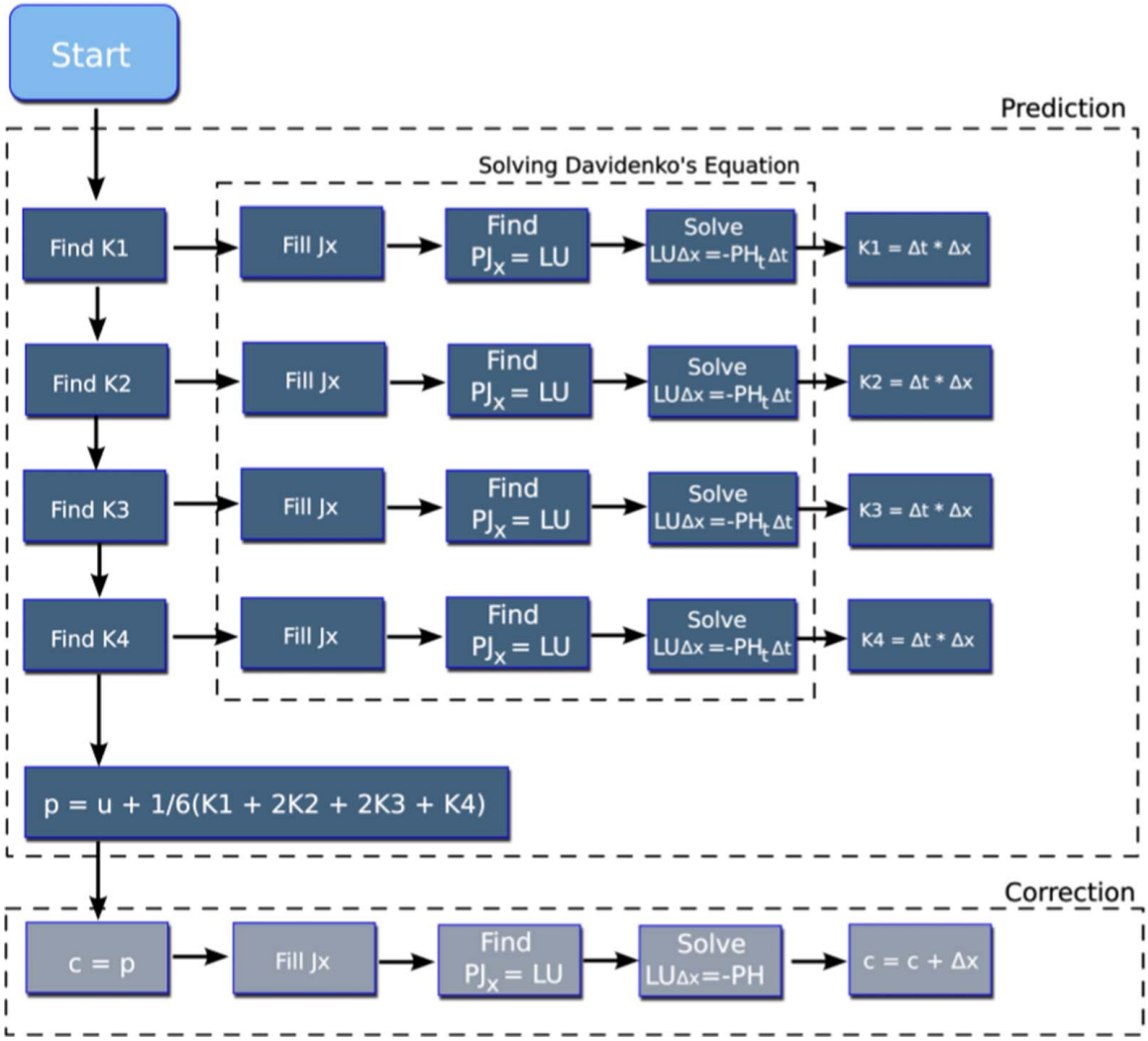

Fig. 1 The solution procedure of the ordinary differential equations for prediction and Newton's method for correction at each path tracking step

However, unlike a CPU that can execute different instruction sets concurrently, the SMs of a GPU execute the same instruction set concurrently. An important consequence of this aspect of SM computation is that conditional instructions degrade the performance on a GPU. This can occur wherever there is an IF statement or a WHILE statement, because when the instructions of some of the threads in a warp are different from the rest, the SM must execute the two different instruction sets one at a time, which results in stalled threads and under-utilization of the GPU. The amount of under-utilization depends on the size of the instruction sets to be used. IF statements are quite useful so they should not be avoided all together, rather the amount of divergence should be considered when implementing code on a GPU, particularly if maximum performance is desired.

In polynomial numerical continuation, branch divergence can occur when the path tracking step size Δt is changed, and when the number of correction steps is changed to achieve convergence. The path tracking algorithms of numerical polynomial continuation solvers implement adaptive step size and convergence checking to ensure the accuracy of each path. Unfortunately, these features can have a large performance hit when implemented on a GPU, because when path tracking step is changed in one thread all the other threads in the warp are paused until the computation is completed. This happens when the number of correction steps is changed as well.

Our approach is to introduce a new path tracking algorithm that is a better match to the SIMT requirements of a GPU, Algorithm 1. Our algorithm is has a fixed number N of equal steps Δt along the path, as well as a fixed number of iterations C of the Newton correction.

Algorithm 1 New path tracking algorithm with fixed step size Δt.

Data: x_0, a solution to $\mathcal{S}(\mathbf{x})$;
Δt, the fixed step size;
C, the number of iterations of newtons method to execute;
N, the number of steps to take;
Result: x, a solution to $\mathcal{P}(\mathbf{x})$
begin
  $x \leftarrow x_0, t \leftarrow 1$
  for $i = 1{:}N$ **do**
    $t \leftarrow t - \Delta t$
    $p \leftarrow predict(\mathcal{H}, \mathbf{x}, \mathbf{t}, \mathbf{\Delta t})$
    $c \leftarrow correct(\mathcal{H}, \mathbf{p}, \mathbf{t}, \mathbf{\Delta t}, \mathbf{C})$
  $x \leftarrow c$
  end
end

It is possible to implement a path tracker with an adaptive step size on a GPU; however, this causes the all the threads of the GPU to run at the speed of the most difficult path. Rather than

parallelize the path tracking, it is also useful to use the GPU to speed up the computation of the elements of both the polynomial system and its Jacobian matrix, see Refs. [18,19]. Our goal is to use the GPU to minimize computation time, so we fix the step size to minimize conditional branching.

Tracking paths with a fixed step size can mean that we cannot avoid the zones around areas singularities which would cause the Jacobian matrix, $\mathbf{J_x}$, to become ill-conditioned. In these zones, the path will almost certainly not result in a solution to the target system of polynomials. Similarly, if the number of iterations of Newton's correction is not enough to assure convergence, the path will not result in a solution.

Both of these issues can be mitigated by an evaluation kernel call that checks all of the roots computed for $\mathcal{P}(\mathbf{x})$ and identifying those paths that do not yield accurate roots. Those that fail can be recalculated using a smaller step size and larger values for N and C. This strategy sacrifices individual path tracking accuracy for increased GPU computational performance associated with a single instruction set.

4 Four-Bar Linkage Synthesis

In this section, we formulate the synthesis equations for a four-bar linkage that we solve using our GPU implementation of numerical polynomial continuation. The goal is to compute the dimensions of a four-bar linkage that guides its coupler through five task positions. We formulate the design problem following Glabe and McCarthy [22] using the loop equations of the linkage. This is different from the usual approach known as Burmester theory that uses the constraint equations of a crank [23]. We use this approach because it can be generalized to design more complex linkage systems [20].

All code was implemented in the Compute Unified Device Architecture (CUDA) language and executed on an Nvidia Quadro M2000 GPU. The sections are broken up into a series of individual functions known as kernel calls. A kernel call is a CUDA term for a function that is called from the CPU, but executed on the GPU.

4.1 Loop Equations.

The synthesis equations that we will solve are obtained from the loop equations of the four-bar linkage. Let the coordinates of the fixed pivots of the linkage be denoted $O = O_x + iO_y$ and $C = C_x + iC_y$ and let the moving pivot coordinates be $A = A_x + iA_y$ and $B = B_x + iB_y$.

The loop equation defines the relationship that is preserved among these variables throughout the movement of linkage linkage. They obtained as the complex vector equations,

$$P = O + Q_\phi(A - O) + T_\theta(P_1 - A) = C + S_\psi(B - C) + T_\theta(P_1 - B) \tag{19}$$

where

$$Q_\phi = e^{i\phi}, \quad S_\psi = e^{i\psi}, \quad T_\theta = e^{i\theta} \tag{20}$$

This loop equation can be used to formulate synthesis equations for the dimensions of the four-bar linkage. First, identify five positions that are to be achieved by the end-effector of the linkage, denoted, $\Gamma_j = (\theta_j, P_j)$, $j = 1, \ldots, 5$, where P_j is the position of the origin and θ_j is the orientation of a desired end effector pose with respect to the x-axis. Then, evaluate the loop equations and their conjugates for each of these task positions. The result is

$$\begin{aligned} O + Q_j(A - O) + T_j(P_1 - A) &= P_j \\ C + S_j(B - C) + T_j(P_1 - B) &= P_j, \quad j = 1, \ldots, 5 \end{aligned} \tag{21}$$

and

$$\begin{aligned} \bar{O} + \bar{Q}_j(\bar{A} - \bar{O}) + \bar{T}_j(\bar{P}_1 - \bar{A}) &= \bar{P}_j \\ \bar{C} + \bar{S}_j(\bar{B} - \bar{C}) + \bar{T}_j(\bar{P}_1 - \bar{B}) &= \bar{P}_j, \quad j = 1, \ldots, 5 \end{aligned} \tag{22}$$

Then, introduce the normal conditions for the angles Q_j and T_j,

$$Q_j\bar{Q}_j = 1 \quad \text{and} \quad S_j\bar{S}_j = 1, \quad j = 1, \ldots, 5 \tag{23}$$

The result is a set of polynomial equations in the coordinates O, A, B, and C and their conjugates that define the coordinates of the pivots of the linkage in the reference position and the relative angles ϕ_j and ψ_j, $j = 1, \ldots, 5$, that define the movement of the linkage through the five task positions.

We can simplify these equations to eliminate the unknowns Q_j, $\bar{Q}_j$, S_j, and $\bar{S}_j$ by solving Eq. (21) for Q_j and S_j and Eq. (22) for $\bar{Q}_j$ and $\bar{S}_j$ and then substitute the results into Eq. (23). This yields

$$\begin{aligned} &(P_j - T_jP_1 + T_jA - O)(\bar{P}_j - \bar{T}_j\bar{P}_1 + \bar{T}_j\bar{A} - \bar{O}) - (A - O)(\bar{A} - \bar{O}) = 0 \\ &(P_j - T_jP_1 + T_jB - C)(\bar{P}_j - \bar{T}_j\bar{P}_1 + \bar{T}_j\bar{B} - \bar{C}) - (B - C)(\bar{B} - \bar{C}) = 0 \\ &\quad j = 1, \ldots, 5 \end{aligned} \tag{24}$$

which form a system of ten quadratic equations.

These equations can be further simplified by selecting the first task position as the reference frame, such that $P_1 = (0, 0)$ and $\theta_1 = 0$ and measuring the remaining four task positions relative to this frame. To do this, compute the five homogeneous transformation matrices H_j associated with the given task positions, $\Gamma_j = (\theta_j, P_j)$, $j = 1, \ldots, 5$,

$$\mathbf{H}_j = \begin{bmatrix} \cos\theta_j & -\sin\theta_j & Px_j \\ \sin\theta_j & \cos\theta_j & Py_j \\ 0 & 0 & 1 \end{bmatrix}, \quad j = 1, \ldots, 5 \tag{25}$$

Then, transform these matrices to the first task frame of Γ_1

$$\mathbf{K}_{1j} = \mathbf{H}_1^{-1}\mathbf{H}_j, \quad j = 2, \ldots, 5 \tag{26}$$

Obtain the new relative task positions $\Gamma_{1i} = (\theta_{1i}, W_{1i})$ as

$$\theta_{1i} = \arctan(k_{21}/k_{11}), \quad W_{1i} = (k_{13} + ik_{23}), \quad i = 2, \ldots, 5 \tag{27}$$

This yields two sets of loop equations relative to P_1,

$$\begin{aligned} \mathcal{P}: \quad &(W_{1i} + T_{1i}A - O)(\bar{W}_{1i} + \bar{T}_{1i}\bar{A} - \bar{O}) - (A - O)(\bar{A} - \bar{O}) = 0, \\ &\quad i = 2, \ldots, 5 \\ &(W_{1i} + T_{1i}B - C)(\bar{W}_{1i} + \bar{T}_{1i}\bar{B} - \bar{C}) - (B - C)(\bar{B} - \bar{C}) = 0, \\ &\quad i = 2, \ldots, 5 \end{aligned} \tag{28}$$

where $T_{1i} = e^{i\theta_{1i}}$. Figure 2 outlines the geometry of the relative displacement of the four-bar linkage. Equation (28) is a system of eight polynomials in eight unknowns $(O, \bar{O}, A, \bar{A}, B, \bar{B}, C, \bar{C})$. Each solution of this set of equations is a candidate for a four-bar linkage that guides its coupler link through the given set of task positions. This polynomial system has a Bezout degree of $2^8 = 256$. We used the numerical continuation solver Bertini [8] to compute a start system for this polynomial system and found that it had a multi-homogeneous degree of 25.

Finding the solutions to these equations can be divided into three separate kernel calls: Path Tracker, Task Generator, and Solution Filter.

4.2 Path Tracker.

The synthesis equations $\mathcal{P}$ in Eq. (28) are linear combinations of monomials formed from the variables $\mathbf{x} = (O, \bar{O}, A, \bar{A}, B, \bar{B}, C, \bar{C})$. These equations include the 16 parameters

$$\begin{aligned} \mathbf{p} = (&T_{12}, T_{13}, T_{14}, T_{15}, \bar{T}_{12}, \bar{T}_{13}, \bar{T}_{14}, \bar{T}_{15}, W_{12}, W_{13}, W_{14}, \\ &W_{15}, \bar{W}_{12}, \bar{W}_{13}, \bar{W}_{14}, \bar{W}_{15}) \end{aligned} \tag{29}$$

which are constants that define the task for the linkage to be designed.

We use the numerical continuation software Bertini to solve start system $\mathcal{S}$ that can be used for parameter continuation to solve a polynomial system $\mathcal{P}(\mathbf{p}, \mathbf{x})$ for different values of the parameters $\mathbf{p}$.

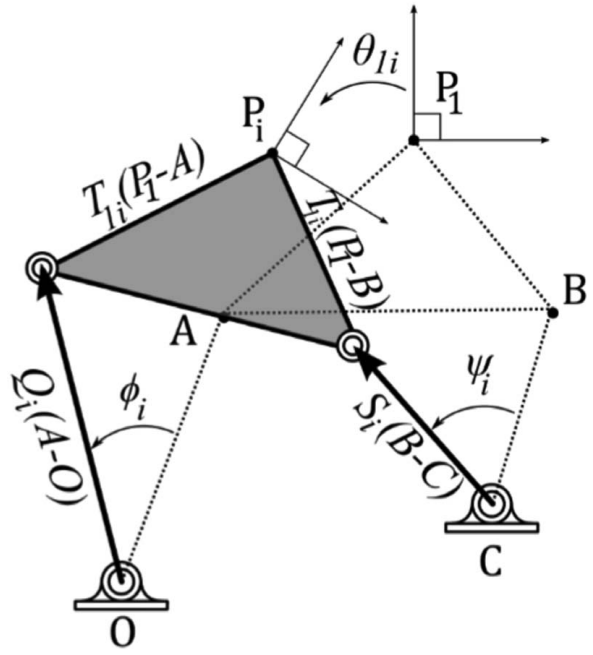

Fig. 2 A four-bar linkage moving from position P_1 to P_i, showing the complex vectors that form the loop equations

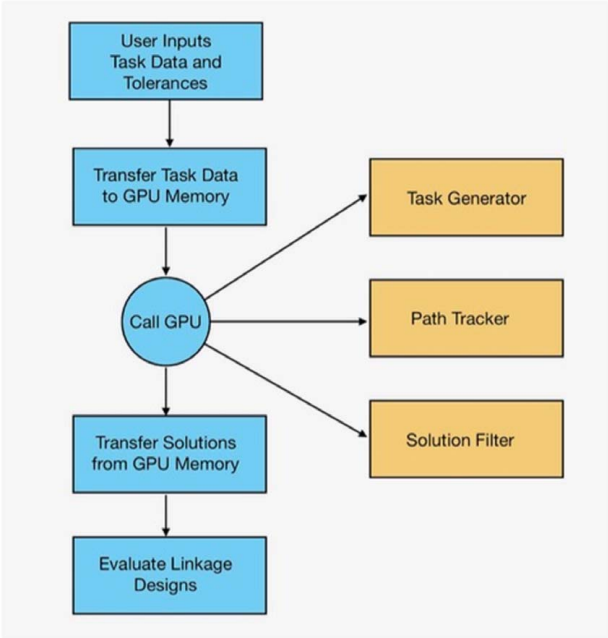

Fig. 3 Flow of calculations on the CPU (on the left) with calls to the GPU on the right

Bertini chooses a generic set of parameters $\mathbf{q}$ so that the start system is $\mathcal{S}(\mathbf{q}, \mathbf{x})$ has known roots $\mathbf{y}_k$.

We use the parameters $\mathbf{q}$ computed by Bertini to construct the parameter homotopy,

$$\mathcal{H} = \mathcal{P}(\mathbf{q}\,t + \mathbf{p}\,(1-t), \mathbf{x}) \tag{30}$$

This is computed symbolically in MATHEMATICA [24], where we also compute symbolic equations for 8×8 Jacobian matrix $\mathbf{J_x}$ and the 8×1 vector $\mathbf{H}_t$; see Ref. [5]. These symbolic equations are copied into the CUDA kernel call, which is labeled PATH TRACKER.

PATH TRACKER includes the algorithm for LU decomposition, Runge–Kutta prediction, and Newton correction. Organizing the calculations in this way uses the advantages of the GPU for rapid computation; however, it means that some of the paths may not converge to roots of our target system. Our formulation of the linkage synthesis problem reduces the importance of finding any particular root. This is discussed in Sec. 4.3.

4.3 Task Generator and Solution Filter. It is the nature of this linkage design problem that for a given task $T{:}\Gamma_j = (\theta_j, P_j)$, $j = 1, \ldots, 5$, the resulting four-bar linkage may have one or more of a set of various defects [25,26]. To address this Plecnik and McCarthy [27] introduced tolerance zones around the specified task positions and randomly selected small variations within these zones. The result is a successful set of linkages that reach task positions close to the originally specified positions, see also Ref. [28].

We implement this strategy by introducing the TASK GENERATOR kernel call. This algorithm reads the specified task positions $\Gamma_j = (\theta_j, P_j)$, $j = 1, \ldots, 5$, and a set of tolerance zones $(\Delta\theta, \Delta x, \Delta y)_i$, $i = 1, \ldots, 5$, and writes L new tasks,

$$T_m: \quad \Gamma_{jm} = (\theta_j + \rho_{jm}\Delta t,\, P_j + \sigma_{jm}\Delta x + i\tau_{jm}\Delta y)_m, \qquad j = 1, \ldots, 5, \quad m = 1, \ldots, L \tag{31}$$

where ρ_{jm}, σ_{jm}, and τ_{jm} are randomly generated constants between -1 and 1. In this example, we set L, the number of task iterations, to be 200. Our PATH TRACKER computes up to 25 roots for each of these 200 paths for 5000 possible linkage designs.

We use a third kernel call SOLUTION FILTER to evaluate each of the 5000 roots to determine if (O, A, B, C) and $(\bar{O}, \bar{A}, \bar{B}, \bar{C})$ are complex conjugates. This ensures that the root yields a physical linkage. It is known that this synthesis problem can have at most six solutions, which means of the 25 roots that exist for each task at most six define physical linkages [23]. It is important to mention that while SOLUTION FILTER does contain IF statements, the branch divergence caused by these statements is quite small and thus has minimal impact on performance.

4.4 Outline of Execution. A block diagram of our GPU-based numerical continuation algorithm for four-bar linkage synthesis is shown in Fig. 3. The equations for the start system, its roots, the homotopy equations, and the derivatives $\mathbf{J_x}$ and $\mathbf{H}_t$ are programed for execution on the GPU.

Execution starts with user input of task positions, tolerance zones, and the number of task iterations L. The number of steps N for the path tracker and the number C of iterations of the Newton correction are also user-specified. We set the number of step $N = 100$, step size $\Delta t = 1/N$, and set the number of Newton corrections to $C = 2$. It reads the coefficients of the 16 parameters for both the start system ($\mathbf{q}$) and the target system ($\mathbf{p}$), along with each of the roots $\mathbf{x}_k$, $k = 1, \ldots, 25$.

The TASK GENERATOR reads this data and writes L different tasks to the GPU memory. The PATH TRACKER computes the roots for the synthesis equations for each task in the GPU memory. The SOLUTION FILTER evaluates each of the roots obtained for all of the tasks to determine those that define physical linkages.

The physical linkages identified in the GPU must be evaluated to determine they are defect-free, which we call effective solutions. Linkages with branch and circuit defects are rejected, but those with order defects are allowed. Examples of this analysis can be found in references such as [29] or [23]. The output of this algorithm is a list of linkage designs that reach the specified task positions within the given tolerance zones.

5 Demonstration

In order to demonstrate this algorithm, we use a Lenovo workstation with an Intel Xeon 2.10 GHz CPU, running Windows 10 with

Table 1 Task position coordinates in the global frame

j	θ_j (deg)	P_j
1	80	(2.7, 4.9)
2	55	(2.8, 4.6)
3	0	(2.9, 4.4)
4	−35	(3.0, 4.2)
5	−60	(3.0, 4.0)

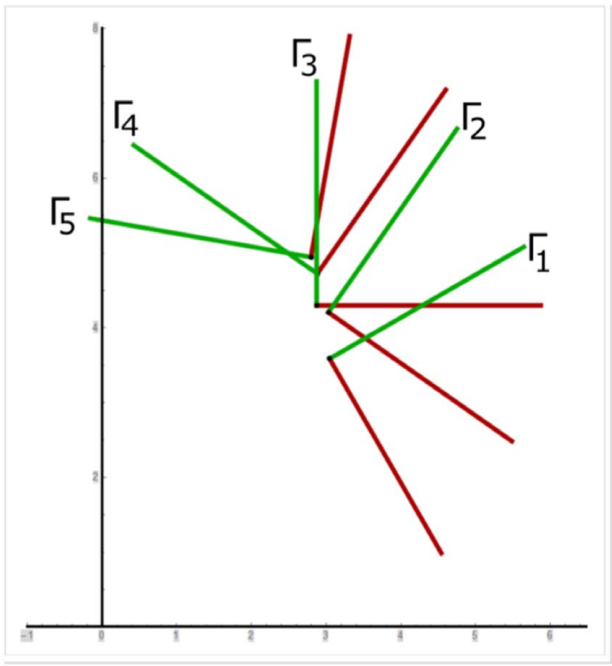

Fig. 4 The five task positions in the global frame

Table 2 Task positions relative to the first task frame

j	θ_j (deg)	P_j
1	0	(0, 0)
2	−25	(−0.28, −0.15)
3	−80	(−0.46, −0.28)
4	−115	(−0.64, −0.42)
5	−140	(−0.83, −0.45)

an NVIDIA Quadro M2000 GPU. The five task positions together are listed in Table 1 and shown in Fig. 4. The tolerance zones chosen were $\Delta\theta = 0.5$ deg, $\Delta x = \Delta y = 0.1$. The original transformed task positions that are used with our synthesis equations are listed in Table 2. Because the randomization can produce two different sets of task positions with differing numbers of solutions to analyze, we generated one standard randomized set and ran both the CPU and GPU code on it. The time to calculate 200 iterations or 5000 threads, with the GPU is shown in Table 3 to be 63 s.

For comparison, we used MATHEMATICA 11.1 to define the synthesis equations for 200 randomized tasks for computation using Bertini v1.5.1 on the Lenovo workstation. The Lenovo workstation has multiple cores which allows Bertini to be ran in parallel using eight CPU threads. Bertini by default performs path tracking using an adaptive step size. The speed up using the GPU was 120 times compared with the eight CPU thread computation. One example solution computed by the GPU is given by the coordinates in Table 4 and shown in each of the task positions in Fig. 5.

A comparison of the results of the two calculations shows the impact of adaptive step size and convergence test for the Newton corrector in Bertini as opposed to the fixed step size and fixed number of Newton iterations. Bertini calculated fewer physical solutions as our GPU code, but more effective designs. This is likely due to the fact that Bertini checks for paths crossing, whereas the GPU does not. If two paths cross, it is possible for one path to jump to the other, resulting in a repeated solution. When calculating the number of effective solutions in the GPU we removed duplicate solutions. Thus, the GPU calculation provides 85 effective designs in 63 s compared to 455 effective designs in just over 2 h of computation.

Table 3 Comparison of kinematic synthesis of eight parallel threads on a workstation CPU with our algorithm on 5000 threads on a GPU

Hardware	Time (s)	Solutions	Physical sols.	Effective sols.
CPU	7588	3774	1510	455
GPU	63	4425	2382	85

Table 4 Joint coordinates in global frame for a selected solution

Point	(x, y) coordinates
O	(2.37, 4.43)
A	(2.54, 4.13)
B	(2.46, 4.58)
C	(2.73, 4.29)

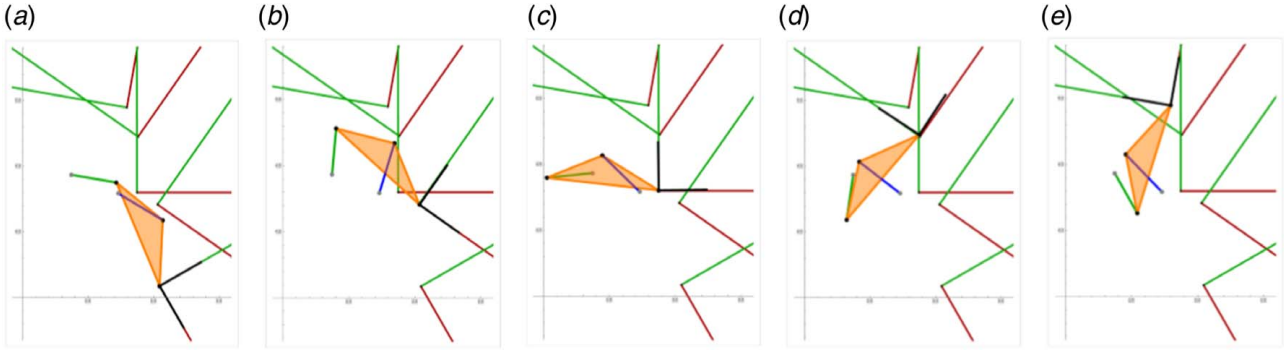

Fig. 5 An example four-bar linkage design moving the end-effector through the five task positions

While it might be tempting to assume using 5000 threads should be 625 times faster than eight threads, it is important to note that GPU and CPU threads are not the same. The base clock speed for an Nvidia M2000 GPU is 1126 MHz while the clock speed for an Intel Xeon CPU is 2.10 GHz. Additionally, CUDA requires threads to be grouped up into blocks, where each block is processed by an SM, one warp at a time. The current GPU algorithm implementation uses only 25 threads per block, whereas on the Maxwell architecture 1024 threads can be executed simultaneously per block. This results in under-utilization of the GPU. More research into this area should be performed to further increase GPU performance.

Additionally, it is important to note that Bertini is not optimized to be executed on instance of the same problem with different parameters. Bertini as a program has many file input/output operations that invariably slows the computation time down. It is important to mention that there is a program being developed called Paramotopy which is intended to be ran on the same problem with different parameters, but the program was unavailable at the time of this paper.

6 Conclusion

This paper presents an algorithm for numerical continuation on a GPU for the solution of the polynomial systems that arise in kinematic synthesis. In order to obtain the benefits of increased computational speed of a GPU the algorithm must run a single instruction set. This lead to a CUDA implementation of Runge–Kutta integration and LU decomposition corrector algorithms for the path tracker for execution on a GPU. In order to eliminate conditional statements that degrade the performance of the GPU, our algorithms use a constant step-size for prediction and a fixed number of iterations for correction. This trades accuracy of the path tracker for speed of computation. This is well-adapted to our application to numerical kinematic synthesis, where a large number of solutions must be generated to find a few effective designs.

A comparison of the performance of our algorithm on 5000 GPU threads with the execution of the software Bertini on eight threads of Lenovo workstation shows a speed up of 120 times. The impact of the trade-off between speed and accuracy can be seen in the fewer number of effective solutions found by the GPU 85 compared with 455 by Bertini. It seems further research is needed to increase speed using a GPU and maintain accuracy.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request. The authors attest that all data for this study are included in the paper. Data provided by a third party are listed in Acknowledgments.

Acknowledgment

This material is based on work supported by the National Science Foundation under Grant No. 1636017.

References

[1] Freudenstein, F., and Roth, B., 1963, "Numerical Solution of Systems of Nonlinear Equations," J. ACM, **10**(4), pp. 550–556.
[2] Roth, B., and Freudenstein, F., 1963, "Synthesis of Path-Generating Mechanisms by Numerical Methods," ASME J. Eng. Ind., **85**(3), pp. 298–304.
[3] Wampler, C. W., Morgan, A. P., and Sommese, A. J., 1992, "Complete Solution of the Nine-Point Path Synthesis Problem for Four-Bar Linkages," ASME J. Mech. Des., **114**(1), pp. 153–159.
[4] Zangwill, W. I., and Garcia, C. B., 1981, *Pathways to Solutions, Fixed Points and Equilibria*, Prentice-Hall, Inc., Upper Saddle River, NJ.
[5] Morgan, A. P., 1983, "A Method for Computing All Solutions to Systems of Polynomials Equations," ACM Trans. Math. Software, **9**(1), pp. 1–17.
[6] Tsai, L. W., and Morgan, A. P., 1985, "Solving the Kinematics of the Most General Six- And Five-Degree-of-Freedom Manipulators by Continuation Methods," J. Mech., Transm. Autom. Des., **107**(2), p. 189.
[7] Sommese, A. J., and Wampler, C. W., 2005, *The Numerical Solution of Systems of Polynomials: Arising in Engineering And Science*, World Scientific Pub Co., Singapore.
[8] Bates, D. J., Hauenstein, J. D., Sommese, A. J., and Wampler, C. W., 2013, *Numerically Solving Polynomial Systems with Bertini*, Society for Industrial and Applied Mathematics, Philadelphia, PA.
[9] Verschelde, J., 2010, "Polynomial Homotopy Continuation With Phcpack," ACM Commun. Comput. Algebra, **44**(3/4), pp. 217–220.
[10] Li, T. Y., and Tsai, C.-H., 2009, "HOM4PS-2.0 Para: Parallelization of HOM4PS-2.0 for Solving Polynomial Systems," Parallel Comput., **35**(4), pp. 226–238.
[11] Su, H.-J., McCarthy, J. M., Sosonkina, M., and Watson, L. T., 2006, "Algorithm 857: POLSYS GLP: A Parallel General Linear Product Homotopy Code for Solving Polynomial Systems of Equations.," ACM Trans. Math. Software, **32**(4), pp. 561–579.
[12] Parker, M., 2017, *Digital Signal Processing 101: Everything You Need to Know to Get Started*, Elsevier Inc., Amsterdam, Netherlands.
[13] Hori, C., Gotoh, H., Ikari, H., and Khayyer, A., 2011, "GPU-Acceleration for Moving Particle Semi-Implicit Method," Comput. Fluids, **51**(1), pp. 174–183.
[14] Pan, J., Lauterbach, C., and Manocha, D., 2010, "g-Planner: Real-Time Motion Planning and Global Navigation Using GPUs," Twenty-Fourth AAAI Conference on Artificial Intelligence, Atlanta, GA, July 11–15, pp. 1245–1251.
[15] Ichter, B., Schmerling, E., Agha-Mohammadi, A., and Pavone, M., 2017, "Real-Time Stochastic Kinodynamic Motion Planning Via Multiobjective Search on GPUs," IEEE International Conference on Robotics and Automation (ICRA), Singapore, Singapore, May 29–June 3, pp. 5019–5026.
[16] Dick, C., Georgii, J., and Westermann, R., 2011, "A Real-Time Multigrid Finite Hexahedra Method for Elasticity Simulation Using CUDA," Simul. Modell. Pract. Theory, **19**(2), pp. 801–816.
[17] Verschelde, J., and Yoffe, G., 2012, "Evaluating Polynomials in Several Variables and Their Derivatives on a GPU Computing Processor," 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, Shanghai, May 21–25, pp. 1397–1405.
[18] Verschelde, J., and Yu, X., 2015, "Tracking Many Solution Paths of a Polynomial Homotopy on a Graphics Processing Unit in Double Double and Quad Double Arithmetic," 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), New York, Aug. 24–26, pp. 371–376.
[19] Verschelde, J., and Yu, X., 2015, "Accelerating Polynomial Homotopy Continuation on a Graphics Processing Unit With Double Double and Quad Double Arithmetic," *CoRR, **abs/1501.06625***.
[20] Plecnik, M. M., and McCarthy, J. M., 2016, "Computational Design of Stephenson Ii Six-Bar Function Generators for 11 Accuracy Points," ASME J. Mech. Rob., **8**(1), p. 011017.
[21] Cheney, W., and Kincaid, D. R., 2012, *Numerical Mathematics and Computing*, Brooks Cole Pub. Co., Pacific Grove, California.
[22] Glabe, J., and McCarthy, J. M., 2019, "Advances in Mechanism and Machine Science," IFToMM World Conference, Krakow, Poland, June 3–July 4.
[23] McCarthy, J. M., and Soh, G. S., 2011, *Geometric Design of Linkages*, Springer, New York.
[24] Wolfram, S., 2003, *The Mathematica Book*, 5th ed, Wolfram Media, Champaign, IL.
[25] Chase, T. R., and Mirth, J. A., 1993, "Circuits and Branches of Single Degree-of-Freedom Planar Linkages," ASME J. Mech. Des., **115**(2), pp. 223–230.
[26] Beloiu, A. S., and Gupta, K. C., 1997, "A Unified Approach for the Investigation of Branch and Circuit Defects," Mech. Mach. Theory, **32**(4), pp. 539–557.
[27] Plecnik, M. M., and McCarthy, J. M., 2011, "Five Position Synthesis of a Slider-Crank Function Generator," Proceedings of the ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Washington, DC, Aug. 28–31, pp. 317–324.
[28] Tsuge, B. Y., and McCarthy, J. M., 2016, "Homotopy Directed Optimization to Design a Six-Bar Linkage for a Lower Limb With a Natural Ankle Trajectory," ASME J. Mech. Rob., **8**(6), p. 061009.
[29] Uicker, J. J., Pennock, G. R., and Shigley, J. E., 2016, *Theory of Machines and Mechanisms*, 5th ed, Oxford University Press, Oxford, UK.