Root-tracking methods for the simulation of constrained multi-body systems

Author 1, Author 2, Author 3, Sandipan Bandyopadhyay*

Department of Engineering Design, Indian Institute of Technology Madras, Chennai 600 036, India.

Abstract

This paper presents a comparative study of three root-tracking methods used for tracking solutions of a system of non-linear equations. The first method, termed the nearest neighbour method, tracks the roots by comparing them based on a measure of distance. The second method, named the Davidenko method or the integration method reformulates the set of equations and poses it as an initial value problem to track the roots. The third, Newton-Raphson based method, iteratively uses the Newton-Raphson technique at each step to track the roots of the equations. The scope, computational time and accuracy of the methods are presented in the comparison. The implementation of the proposed methods is illustrated using the semi-regular Stewart platform manipulator (SRSPM) following a given path. Numerical studies show that all three methods are useful based on the context of their application. Further, the utility of these methods is demonstrated using a path following problem of a 3-3 cable driven parallel robot (CDPR).

Keywords: Root-tracking, parallel manipulators, cable driven parallel robot (CDPR)

1. Introduction

(sc:intro) Unlike linear equations, non-linear equations produce multiple sets of feasible solutions or roots. The problem of keeping track of a particular root of interest is essential not only in the kinematics of multi-body systems but also in the problems of speech transmission and direction of arrival estimation [1](starer1992high). There are several methods in the literature to obtain the roots of non-linear equations arising in the kinematics of manipulators, [2, 3, 4, 5](wampler2005numerical, raghavan,Manocha,merlet2009interval). Such equations are generally parametric in terms of the manipulator configuration, and their roots vary as its pose changes. In the context of parallel manipulators,

*Corresponding author, Phone: +91 44 2257 4733, Fax: +91 44 2257 4732.
Email addresses: email 1 (Author 1), email 2 (Author 2), email 3 (Author 3), sandipan@iitm.ac.in

their closed-loop architecture gives rise to the loop-closure constraints. Feasible configurations of the system are obtained by solving these non-linear loop-closure equations. The procedure to solve these equations is not straightforward and is computationally expensive [6](ghosal2006robotics). In practical scenarios, the problem of tracking the roots are resolved either by using sensor-based methods [7, 8](stoughton1991optimal, dallej2012vision) or are avoided by using alternate strategies [9, 10](tempel2015modelling, miermeister2010modelling). Despite these techniques, Merlet et al. [11](merlet2017simulation) emphasises the need for ways to solve and track the roots of these equations in developing software platforms for computer simulations of manipulators.

Tracking the branches becomes essential for the simulation of systems like CDPRs, where additional constraints are imposed along with the kinematic constraints as discussed in [12](borgstrom2007discrete). In the context of dynamics, an actuator-space formulation is essential for real-time control applications as illustrated in [13](abdellatif2009computational). In such cases, tracking methods enable the computation of the passive joint variables given the actuator variables at each time step using the constraint equations. Further, these methods are not limited to actuator and configuration space variables, any different sets of variables related via the constraint equations can be tracked.

Parameter homotopy [14](bates2018paramotopy) is one of the popular methods used in tracking the roots of parametrised polynomials. The techniques presented in this paper differ from homotopy continuation or parameter homotopy as the later are limited to polynomials, whereas the former deals with a set of generic non-linear equations. The use of trigonometric identities is one of way to convert the loop-closure equations into polynomials. However, such conversion leads to a set of equations with high Bézout's number, thereby increasing the computational burden. Further, such transformations bring in additional parametric singularities which need to be carefully handled. The objective of this paper is to formulate tracking algorithms for a generic set of non-linear equations based on speed and accuracy requirements. To the authors best knowledge, a comparative study of such root-tracking methods is not reported extensively in the literature.

The layout of the rest of the paper is: problem setting and the algorithms for root-tracking are described in Section 2(sc:meth). A discussion on the methods is presented in Section 3(sc:disc). Demonstration of the techniques to solve a path following problem of an SRSPM and a 3-3 CDPR is illustrated in Section 4(sc:impl). Finally, the conclusion and future work constitute Section 6(sc:conc) of the paper.

(Sandipan Bandyopadhyay)

2. Root-tracking algorithms

(sc:meth) This section explains the details of the above mentioned root-tracking algorithms Consider a set of n non-linear equations in variables $\boldsymbol{x}$ and $\boldsymbol{y}$ of the form,

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0}, \tag{1}$$

where $\boldsymbol{x}$ is a set of m known and $\boldsymbol{y}$ is a set of n unknown variables implicitly dependent on time. For the system of equations in Eq. (1), there are more than one set of feasible solutions and each set corresponds to a particular *branch* of solutions. The branches represent the evolution of the roots of the equations ($\boldsymbol{y}$) with the variation of the known variables ($\boldsymbol{x}$). In the context of the kinematics of parallel manipulators, the equations correspond to the loop-closure constraints obtained in terms of the active and passive variables and their solutions represent the forward kinematic (FK) branches which vary with the manipulators pose.

As explained in the Section 1(sc:intro), it is often necessary to keep track of the required branch of roots at each instant as the equations evolve with time, from $t = 0$ to $t = t_f$, i.e., for the entire duration of the simulation. As cited earlier, several methods in the literature for solving such equations exploit the specific form and nature of the equations. However, in this work, we only deal with tracking the roots, given at least one set of solutions, to begin with. For further discussion, it is assumed that roots of the equations are known at time $t = 0$. The following are a few techniques for root-tracking under these assumptions. Of the n sets of solutions at time t, let the jth solution set represented as $\boldsymbol{y}_j^t$, belong to the required branch. Then the problem is to find the solution belonging to the same branch at time $t = t + \delta t$ amongst all the n branches, $\boldsymbol{y}_1^{t+\delta t}, \boldsymbol{y}_2^{t+\delta t}, \boldsymbol{y}_3^{t+\delta t} \ldots \boldsymbol{y}_n^{t+\delta t}$.

2.1. Nearest neighbour method

As explained earlier the nearest neighbour method relies on a distance metric for identifying the roots belonging to the required branch. Upon solving the loop-closure equations, multiple feasible configurations of the parallel manipulator are obtained, and the one corresponding to the physical robot is chosen as the actual pose of the manipulator. In the nearest neighbour method, all the roots obtained at time $t = t_i$ are compared with the chosen solution at the previous instant $t = t_{i-1}$ and the root *closest* to it is then selected as the solution at time t_i. Since this method employs the notion of distance for comparison, attention should be given to the *space* the computed variables belong to. For example, let the variable set involve an angle, say $\theta \in \mathbb{S}^1$. Since the L_2 norm fails to capture the *distance* between any two elements in $\mathbb{S}^1$, and therefore fails to be a useful metric for comparison. In such cases, the variables are mapped to suitable sub-spaces where the distance measure locally holds. In the above example, the L_2 norm can be

used if, all the angles are mapped into the sub-space of $[0, 2\pi)$. Then, the computed length of the minor arc between the two angles serves as a valid distance measure.

The entire simulation duration, from $t = 0$ to t_f, is discretised into k finite steps. The nearest neighbour procedure assumes the existence of a solver (Solve$(\cdot)$) capable of computing all the solutions of the required set of equations. From the known initial configuration of the system at time $t = 0$, the algorithm proceed to track roots at each discrete step using the roots computed by the solver at that step.

[AS: *Ignore the formatting and offset. This is due to the change of template*] [AS: *Not changing the changes noted in the print version*]

Method 1 Root-tracking using the nearest neighbour method

Input: Initiate the branch with the intial solution at time $t = 0$, as $\boldsymbol{y}_0$
Output: A list of solutions belonging to the required branch at each instant

1: **procedure** NEARESTNEIGHBOUR
2: **for** $i = 1 \rightarrow k$ **do**
3: $\boldsymbol{y}_j \leftarrow \texttt{Solve}(\boldsymbol{f}(\boldsymbol{x}_j, \boldsymbol{y}) = \boldsymbol{0})$ ▷ where $j = 1 \ldots n$ and $\texttt{Solve}(\cdot)$ returns all the roots of the input equations and $\boldsymbol{x}_j$ is the value of the known variables at jth instant
4: $\boldsymbol{y}_s \leftarrow \texttt{Select}(\min|\max(\boldsymbol{y}_j - \boldsymbol{y}_0)|)$ ▷ where $\max(\cdot)$ and $\min(\cdot)$ return the maximum and minimum value among all values of an input list and $|\cdot|$ returns the absolute values of all the elements of the input list. $\texttt{Select}(\cdot)$ selects the element at the index corresponding to the minimum value element of the list
5: $\texttt{Append}(\boldsymbol{y}_s)$ ▷ Appends $\boldsymbol{y}_s$ to a list of solutions
6: $\boldsymbol{y}_0 \leftarrow \boldsymbol{y}_s$
7: **end for**
8: **end procedure**

The major drawback in this procedure is the necessity to compute all the roots up to the required accuracy. Often it is a memory and computationally intensive process to obtain all the solutions. Moreover, in reality, computing and updating only the required set of solutions corresponding to the required branch is sufficient. Therefore, such a method is feasible for manipulators with the loop-closure constraints to leading to non-linear equations or polynomials of small order as in the case in [15, 16](agarwal2016dynamic, nasa2011trajectory).

However, in the case of manipulators like SRSPM and 6-RSS where solving the FK problem is not straightforward; this method is not the best choice. However, if the information of all the branches is available, then the nearest neighbour method is both easy to implement and computationally faster as the time required corresponds only to the time taken in comparison of the roots.

2.2. Newton-Raphson-based root-tracking

Unlike the nearest neighbour method, only the root corresponding to the required branch is computed in the Newton-Raphson based root-tracking method. The current method uses the Newton-Raphson technique to compute the required root at time $t = t_i$, utilising the solution at time $t = t_{i-1}$ as its initial guess. Therefore, this method requires the computation of the Jacobian, defined as,

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}, \tag{2}$$

at each instant. The assumption of working within the safe working zone (SWZ) explained later, ensures the continuous existence of the Jacobian and therefore guarantees the convergence of the Newton-Raphson method.

Method 2 Root-tracking using Newton-Raphson method

Input: Initiate the branch with the intial solution at time $t = 0$, as $\boldsymbol{y}_0$
Output: A list of solutions belonging to the required branch at each instant

procedure NEWTONRAPHSON
  for $i = 2 \rightarrow k$ **do**
    $\boldsymbol{f}(\boldsymbol{y}) \leftarrow \boldsymbol{f}(\boldsymbol{x}^i, \boldsymbol{y})$ ▷ Substituting $\boldsymbol{x}^i$, value of the known variables $\boldsymbol{x}$ in the *i*th iteration
    $\boldsymbol{J}(\boldsymbol{y}) \leftarrow \boldsymbol{J}(\boldsymbol{x}^i, \boldsymbol{y})$
    $\boldsymbol{f} \leftarrow \boldsymbol{f}(\boldsymbol{y}^{i-1})$
    while $\max(|\boldsymbol{f}|) \geq \epsilon$ **do** ▷ ϵ is the numerical zero
      $\boldsymbol{J} \leftarrow \boldsymbol{J}(\boldsymbol{y}^{i-1})$
      $\delta\boldsymbol{y} \leftarrow \texttt{Solve}(\boldsymbol{J}\delta\boldsymbol{y} = \boldsymbol{f})$
      $\boldsymbol{y}^i \leftarrow \boldsymbol{y}^{i-1} - \delta\boldsymbol{y}$
      $\boldsymbol{f} \leftarrow \boldsymbol{f}(\boldsymbol{y}^i)$
    end while
  end for
  $\texttt{Append}(\boldsymbol{y}_i)$ ▷ Appends $\boldsymbol{y}_i$ to a list of solutions
end procedure

Where ϵ is a predefined numerical zero, i.e., any value a is considered to be zero if $|a| \leq \epsilon$, and $\epsilon \in \mathbb{R}^+$. An inherent advantage of the current method is the explicit use of the loop-closure equations, which allows the computation of the roots up to a required precision by adjusting the ϵ value. Moreover, since the solutions satisfy the loop-closure equations, they are always physically feasible. Owing to its advantages, the NR method is the most popular method used in the literature.

2.3. Davidenko method or integration of the first order form of the equations

The loop-closure equations mentioned in Eq. (1), do not have an explicit dependency on time and hence are scleronomic in nature [17]. Therefore, their time derivative can be written as,

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}\,\dot{\boldsymbol{y}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\,\dot{\boldsymbol{x}} = \boldsymbol{0}, \tag{3}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ are known and unknown variables respectively. From Eq. (3), a relation between the variables $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{y}}$ can be established as.

$$\dot{\boldsymbol{y}} = \boldsymbol{J}_{\boldsymbol{yx}}\dot{\boldsymbol{x}}, \quad \text{where } \boldsymbol{J}_{\boldsymbol{yx}} = -\boldsymbol{J}_{\boldsymbol{fy}}^{-1}\boldsymbol{J}_{\boldsymbol{fx}},\ \det(\boldsymbol{J}_{\boldsymbol{fy}}) \neq \boldsymbol{0}, \tag{4}$$

$$\text{and } \boldsymbol{J}_{\boldsymbol{fx}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}},\ \boldsymbol{J}_{\boldsymbol{fy}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}. \tag{5}$$

The Eq. (4) is nothing but the Pfaffian form of the constraint equations in Eq. (1). This differential form of the Newton equations are also called the Davidenko differential equations [AS: *cite*]. Values of the unknown variables $\boldsymbol{y}$ are computed by solving Eq. (4) as an initial value problem using numerical integrator.

Method 3 Root-tracking by Davidenok's method

Input: Initiate the branch with the intial solution at time $t = 0$ as $\boldsymbol{y}_0$
Output: A list of solutions belonging to the required branch at each instant

1: **procedure** INTEGRATECONSTRAINT
2: **for** $i = 1 \rightarrow k$ **do**
3: $\Delta\boldsymbol{x} = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$
4: $\boldsymbol{y}_{i+1} = \boldsymbol{y}_i + \boldsymbol{J}_{\boldsymbol{yx}}\Delta\boldsymbol{\theta}$ ▷ Assuming an Euler step for integration
5: `Append`$(\boldsymbol{y}_{i+1})$ ▷ Appends $\boldsymbol{y}_i + 1$ to a list of solutions
6: **end for**
7: **end procedure**

As seen from Method 3, similar to the NR method, only the root corresponding to the required branch is tracked. Reddy et al. [18](reddy2016comprehensive) have used such method in tracking roots of a 65-degree polynomial equation obtained from solving the FK problem of an automotive suspension system for continuous steering and road profile input. Method 3 illustrates the implementation of Davidenko's method using the explicit Euler integration scheme. The inversion of the constrained Jacobian involved in the derivation of the Davidenkos form is done numerically. Hence, care must be taken if the manipulator is working outside the SWZ. As explained in [19](hejase1993use), the convergence of the Davidenko equations is more reliable than the Newton-Raphson method. Although this method relies on the error monitoring and the step size control of the ODE solver, the solutions generally diverge from the *constraint*

manifold due to the errors in the numerical integration method used.

3. Discussion

(sc:disc) This section contains a discussion on computational aspects of various root-tracking methods. At a singularity, the branches of solutions merge and hence cannot be resolved using the discussed root-tracking methods. Therefore, all the methods fail at or close to the location of the singularity. Singularity effects all the techniques and hence cannot be avoided by choice of any particular technique. While dealing with simulations in discrete steps, it is possible to *skip* the singularity due to the choice of step size used. For example, in Method 1, there is a possibility of *jumping over* the singularity without being noticed. The same is shown in Fig. 1. Moreover, in some cases, even at the singular configuration, Jacobian matrix can still be computed using a pseudo inverse and hence care should be taken during the implementation. Such a phenomenon is dangerous in case of CDPRs where the active set of cables for a redundant manipulator change at the singularity, significantly changing the position of the moving platform as discussed in [11].

[AS: *Redo image based on suggestions*]

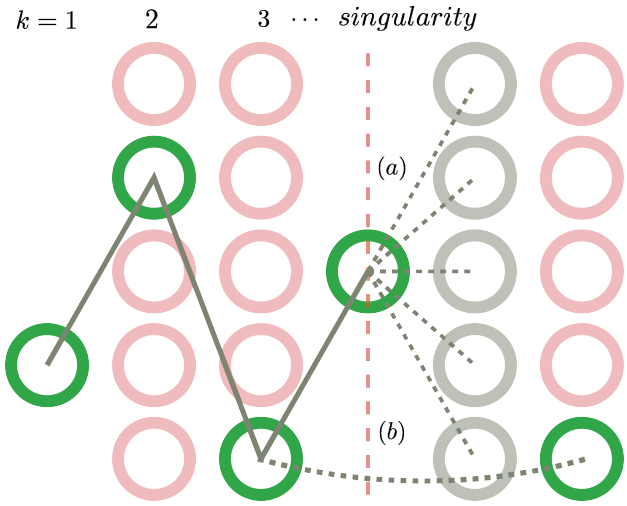

Figure 1: Pictorial representation of the nearest neighbour method. (a) Issue of resolving the branches at a singularity, (b) Possibility of jumping ahead of a singularity

At a singularity, the individual forward kinematic branches do not carry any significance as two or more of the branches merge and tracking the roots loses meaning. The assumption of remaining within the SWZ ensures the unique existence of solutions and hence be tracked at all times. The SWZ as described in [20](karnamcomputation) can be summarised as the *connected* subset of the workspace free from loss-type, gain-type singularities where the manipulator moves within its joint constraints and without self-interference.

As mentioned earlier, for Methods 2 and 3 the size of the discretisation step plays a critical role in determining the convergence and the quality of the solutions obtained. For a given time interval, $[t, t+\epsilon]$, the implicit function theorem allows a unique solution for a system of non-linear equations in the neighbourhood of the solution at time t, $\boldsymbol{y}^t$, given the Jacobian of the system is regular. Therefore, the implicit function theorem ensures that the Newton-Raphson method converges to a unique solution throughout the SWZ for a chosen small step size as elaborated in [11](merlet2017simulation).

The numerical integration step used in Method 3 presents a trade-off between accuracy and the time taken to compute the solutions. Due to the numerical errors introduced during integration, the calculated values of the solutions diverge from the actual, violating the constraint equations. These errors are depicted in Fig 1 as the deviation of the solutions from the constraint manifold. Whenever the drift exceeds a particular threshold value, the Newton-Raphson stem is employed to compute solutions that satisfy the constraint manifold. Therefore, using an explicit Euler scheme would lead to fewer functional evaluations and faster computational times at the cost of the accuracy of the solution obtained.

Despite working within the SWZ, numerical artefacts mentioned above might still affect the root-tracking procedure. In scenarios where the above methods fail to converge to a feasible solution, Vyankatesh et al. [21](vyankatesh2018), have used the analytical FK formulation to compute all the sets of roots at the current step and resume the use of root-trackers from the next instant. This strategy ensures a robust working of the simulation. The nearest neighbour method only involves the comparison of roots; therefore, the accuracy of the roots obtained is determined by the solver `(Solve(·))` used for the computation of the roots.

4. Implementation

(sc:impl) The current section discusses the implementation details of all the algorithms and is demonstrated by solving a path following problem of two different manipulators, i.e., an SRSPM and a 3-3 CDPR.

4.1. Semi-Regular Stewart Platform Manipulator (SRSPM)

The Gough-Stewart or the Stewart platform manipulator is a six degree-of-freedom parallel manipulator. Its construction consists of a fixed platform connected to six linear actuators (legs) through universal or spherical joints and a moving platform attached to the other ends of the linear actuators via spherical joints, as shown in Fig. 2(fig:srspm1). Since its introduction in [22](stewart1965platform), this manipulator has attracted a large amount of research on the topics of its kinematics, dynamics and control. The inter-

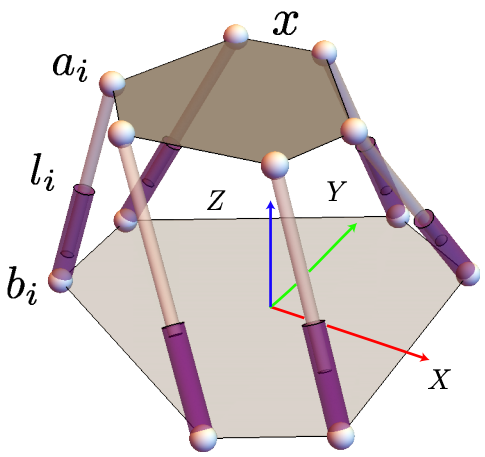

Figure 2: Architecture of a Semi-Regular Stewart Platform Manipulator

est in this particular manipulator stems from its wide range of applications including automotive simulators [23, 24](freeman1995iowa, park2001development), flight simulators [25](pradipta2013development), machine tools [26](lebret1993dynamic), etc.

4.1.1. Constraint equations and the forward kinematic problem

The FK problem of the SRSPM deals with obtaining all the feasible poses of the manipulator given the lengths of the prismatic links. As mentioned earlier, there are several prior works in solving the forward kinematic problem, e.g., [27, 28, 29](lee2001forward, dasgupta1994canonical, lee2003improved). Nag et al. [30](nag2019comparative) have shown that addressing the FK problem of an SPM involves deriving and solving a 20-degree polynomial equation called the forward kinematic univariate (FKU). FKU is parametrised in terms of the actuated variables; therefore, roots of the polynomial changes as the manipulator moves. The 18 dimensional configuration space of an SRSPM consists of,

$$\boldsymbol{q} = [\boldsymbol{\phi}^\top, \boldsymbol{\theta}^\top]^\top, \tag{6}$$

where $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are variables representing the 6 leg lengths and 12 passive joint angles, respectively. Following the procedure mentioned in Bandyopadhyay et al. [31](bandyopadhyay2006geometric), a set of 12 constraint equations are obtained by enforcing the conditions of the rigidity of the top platform. These constraints, called the loop-closure equations, are non-linear with its terms containing trigonometric entities.

The computational speed and accuracy of the methods are evaluated by simulating an SRSPM following a given path. The path followed by the centroid of the manipulator is ensured to be within the SWZ and its complete description is given in Appendix A. Further, the path is discretised into 50 points in the Cartesian space, and the leg lengths are obtained by solving the IK problem at each discretised location. The physical parameters corresponding to the manipulator used in the simulations can be found in Ap-

pendix A. All the data corresponding to the root-tracking algorithms are obtained through simulations in `C++` on an AMD Ryzen 7, 8 core CPU with a clock speed of 3.6 GHz, 16 RAM machine installed with Ubuntu 18.04.1 LTS. All the execution times reported are the average values of 1000 trails run only on a single core.

5. Comparison

This section presents the comparison between various algorithms in terms of their computational speed and accuracy.

5.1. Nearest neighbour method

Since this method relies on `Solve(·)` function to compute the roots, the accuracy of the solutions obtained correspond to the accuracy of the solver used. The clock-time taken for simulating the system using the nearest neighbour method is 0.004 milliseconds. The time taken in computing the roots is not included in the mentioned root-tracking time. If the roots at all disctretisation points are computed a priory, then the Nearest Neighbour method would be the most computationally efficient, as the only operation performed is the comparison between the different roots.

5.2. Integration of the first order of the equations

On the other hand, the root to be tracked is computed as a part of the root tracking process itself in the case of Methods 2 and 3. In Method 3 an explicit Euler integration scheme is used to integrate the obtained ODE equations Eq. (4). Method 3 took 0.452 milliseconds to track the required root through all the 50 discretisation steps. As a measure of the quality of root-tracking, an error metric (e_1) is introduced, which is a measure of the *drift* of constraint equations, i.e. the measure of how much the constraints are violated, and is defined as,

$$e_1 = \max |\boldsymbol{\eta}(\boldsymbol{q})|, \tag{7}$$

where $\max(\cdot)$ selects the maximum among all the values of a list of elements. The error e_1 (Eq. (7)) plotted against time instants is shown in Fig. 3. Moreover, starting from the initial estimate of the configuration variables, the current method tracks only the roots belonging to the required branch. Since the Euler integration step consists of a few operations, it is generally computationally faster than the Newton-Raphson method. However, with time the solutions start to drift from the actual values violating the loop-closure

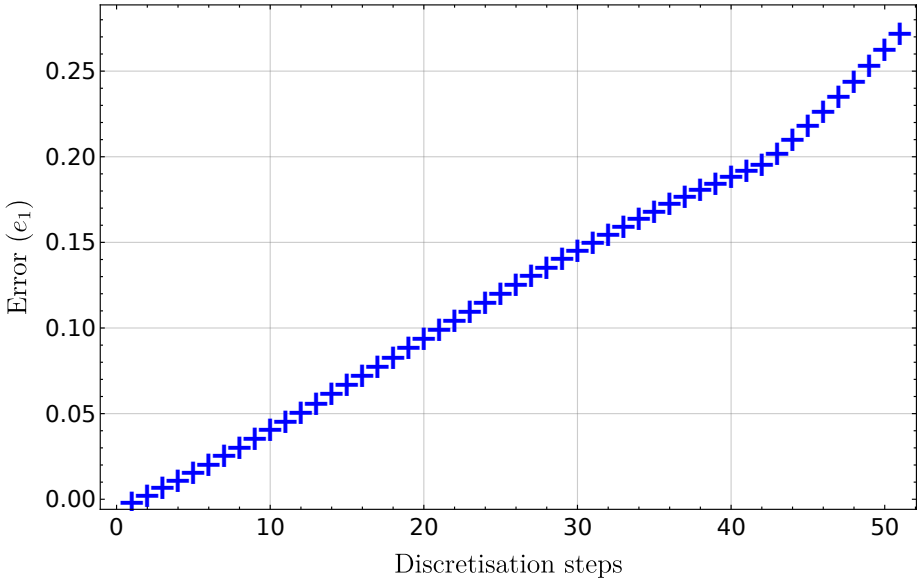

Figure 3: Error measure (e_1) as given in Eq. (7), representing the drift of the solutions obtained from the constraint manifold at each step of integration

constraints as shown in Fig. 3. A Newton-Raphson correction step is executed whenever the errors exceed a predefined tolerance value to correct the drift. This ensures that the computed solutions satisfy the constraint manifold again, as shown in Fig. 4.

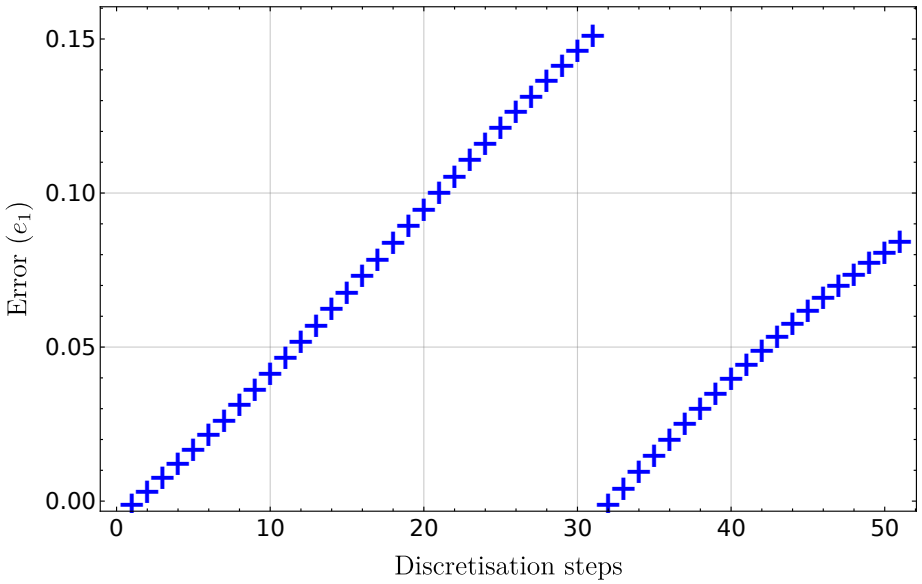

Figure 4: Depiction of correction of the drift in the solutions, using a Newton-Raphson step, bringing them onto the constraint manifold

5.2.1. Newton-Raphson method for root-tracking

For the same path described earlier, Newton-Raphson based root-tracking method takes 0.777 milliseconds to track the required root through the entire simulation. The error e_1 for the solutions computed using the Newton-Rasphson method using a step size of [AS: *Fill*], is 10^{-10}. This error value can be set using the tolerance value for convergence of Method 2. The Table 1 summarises the computational time and accuracy of all the three methods.

Table 1: Computational time taken and the accuracy of the solutions in various methods of root-tracking

Root-tracking method	**Computational time (ms)**	**Accuracy (Error e_1)**	**Tunable parameter**
Nearest neighbour	0.004	10^{-10}	FK algorithm used
Constraint integration	0.452	10^{-1}	Integration step used(Euler)
Newton-Raphson	0.777	10^{-10}	Accuracy goal used

To obtain highly accurate solutions to the forward kinematic problem, special care must be taken like the multi-precision representation of variables ([29](lee2003improved)), which are generally time and resource intensive computations. Although the solvers are computationally expensive the time taken to compute the roots is not included in the tracking time of the Method 1.

5.3. Applications in simulation of cable-driven parallel robots (CDPR)

Cable-driven parallel manipulators (CDPR) are a class of parallel manipulators with the moving platform connected to the stationary platform via cables. Although CDPRs have a low mass and better load carrying characteristics similar to parallel manipulators, cables only support tension, unlike their rigid body counterparts, making analysis and design of CDPRs much more challenging. The interest in CDPR's is inspired from their potential applications in material handling [32](bostelman1994applications), exoskeletons [33](garrec2008able), rehabilitation [34](cablerehab) and re-configurable robotics [35](nguyen2014analysi

The forward kinematic problem of CDPR deals with computing the position and orientation of the moving platform given the lengths of all the cables. The importance of calculating the FK solutions at each iteration, especially for the Cartesian space control is illustrated in [36, 37](yamamoto1999inverse, gallina2001planar). Unlike parallel robots with rigid membered links, the loop closure equations must be solved along with the conditions of static equilibrium called the geometrico-static problem, to obtain the FK solutions for a CDPR. Further, additional constraints on unilaterality of the cable forces should be imposed to obtain physically feasible configurations amongst all the computed solutions.

5.3.1. The 3-3 cable-driven parallel manipulator

Consider the under-constrained spatial 3-3 cable-driven manipulator presented in [38](carricato2010geometrico), where the mobile platform is connected to the fixed base via 3 cables, as shown in Fig. 5. It is assumed that the cables are non-deformable. Carricato et. al. [38](carricato2010geometrico) have illustrated the complexity of the geometrico-static problem to obtain feasible poses.

For the treatment of a general redundant manipulator dealing with change of the operating cables during the motion, one should refer to [11]. Three loop-closure equations are derived by enforcing the con-

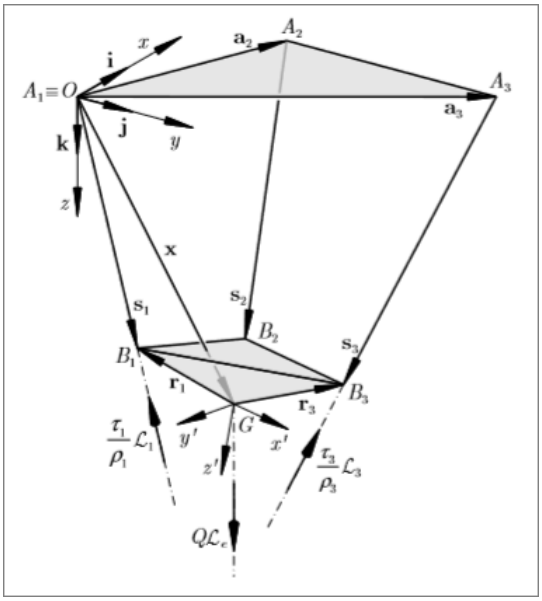

Figure 5: Description of the 3-3 CDPR [38]

straints on the lengths of the three cables, A_iB_i. Following the methodology described in [38](carricato2010geometrico further, a set of 12 equations are obtained from the static equilibrium condition of the manipulator. In total they form a set of 15 equations in 6 variables ($\boldsymbol{X}$), describing the position and orientation of the moving platform. where $\boldsymbol{X}$ is,

$$\boldsymbol{X} = [x, y, z, e_1, e_2, e_3]^\top. \tag{8}$$

The 12 equations obtained by the above procedure are linearly independent. The selection of any of these equations with the three loop-closure constraints forms a valid system of six equations with six unknowns. Abbasnejad et al. [39](abbasnejad2012real) have illustrated the introduction of spurious roots while solving the FK problem using the above-formulated set of equations. However, this is not true in the case of root-tracking methods initiated with a known initial configuration. Since the choice of the equations does not change the solutions, equations with the smallest degree in $\boldsymbol{X}$ are selected. Further, it should be noted that irrespective of the chosen equations, the obtained roots would also satisfy all the 15 constraint equations.

5.3.2. Root-tracking problem

All the dynamic effects of the CDPR are ignored by assuming its motion to be quasi-static. The physical parameters of the manipulator used for simulation are given in Appendix B. The FK problem of a 3-3 CDPR formulated in Section [AS: *ref*] yields 156 solutions. All the obtained solutions are checked to satisfy the unilaterality constraints, and only the poses which admit tension in all the cables are selected. This check is necessary to ensure the feasibility of the computed FK solutions. However, this additional

step at each iteration increases the computational burden. Moreover, the accuracy of the solution is critical to ensure safe operation of the manipulator. Therefore root-tracking methods have a clear advantage in their implementation in problems dealing with CDPRs.

5.3.3. Newton-Raphson method based root-tracking

It is observed that out of all 156 computed roots, only 10 of them are real-valued solutions, of which only 6 admit positive tensions in all the cables. Therefore all the six solutions are physically feasible and correspond to a possible configuration of the system. Starting from any of such settings, let the CDPR follow a given path *P*, within the SWZ. The numerical details of the initial configuration and the path followed by the manipulator are given in Appendix A. Further, the path followed by the CDPR is given in Fig. 6. Moreover the accuracy of the computed solutions described in terms of error e_1 are of the order, 10^{-11}, which is comparable to the order of the roots reported in [40], which were obtained by solving the FK problem using high precision computations. For the discretisation of 1000 steps, the time taken to track is 0.0619 seconds, for required precision set to 10^{-10}. The red curve is the path followed by the manipulator shown in Fig. 6.

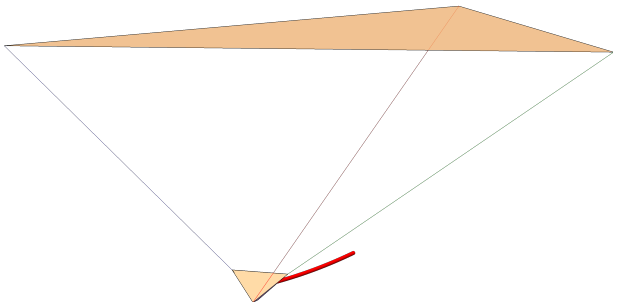

Figure 6: The path followed by the moving platform of the CDPR represented by a red curve

6. Conclusion

(sc:conc) Three methods for the application of root-tracking are discussed and compared. The drawbacks and the scope of the methods are presented. Their implementation on SRSPM for a path following problem is illustrated. As the number of solutions for the non-linear equations increase, it is difficult to solve the FK solution iteratively as in the case of CDPRs. A path following application is demonstrated for the 3-3 CDPR. Such methods allow one to simulate the dynamics of the system with any set of variables used to represent the system. Fast computations allow higher control bandwidths and hence better tracking performance of high-frequency motions. [AS: *Add references, add content here after running final runs*] [AS: *re-emphasize the importance of tracking in change of variables and control especially model predictive type where error computation is necessary*]

7. Acknowledgement

[AS: *What should this section contain?*] [AS: *Verify all the references, heading positions, heading names legends and plots etc., complete references*]

Appendix A. Physical parameters of the SRSPM

Table A.2: Parameters of the SRSPM used to solve the FK problem

Parameter	Symbol	Value	Unit
Circumradius of the base platform	r_b	1	m
Circumradius of the moving platform	r_t	0.5803	m
Angular spacing between the adjacent pair of legs of the fixed platform	γ_b	0.2985	rad
Angular spacing between the adjacent pair of legs of the moving platform	γ_t	0.6573	rad
Length of the sliding link of the prismatic joint	$l_{bi}, i = (1, \ldots, 6)$	0.5	m
Length of the fixed link of the prismatic joint	$l_{ai}, i = (1, \ldots, 6)$	1.5	m

Appendix B. Details of the 3-3 CDPR

Table B.3: Physical parameters used for the FK problem of the CDPR

Parameter	Symbol	Value (m)
Coordinates of the vertices of the fixed platform	$\boldsymbol{a}_1$	$[0, 0, 0]^\top$
	$\boldsymbol{a}_2$	$[10, 0, 0]^\top$
	$\boldsymbol{a}_3$	$[0, 12, 0]^\top$
Coordinates of the vertices of the moving platform	$\boldsymbol{b}_1$	$[1, 0, 0]^\top$
	$\boldsymbol{b}_2$	$[0, 1, 0]^\top$
	$\boldsymbol{b}_3$	$[0, 0, 1]^\top$

Appendix C. Moved back

For the sake of demonstration, the centroid of the moving platform of the SRSPM is required to follow a circular path for which the pose of the moving platform is parametrised as,

$$\boldsymbol{X} = [0.2\cos\alpha, 0.2\sin\alpha, 1.1, 0.2\sin\alpha, 0.2\cos\alpha, 0]^\top.$$

The path is discretised in terms of the parameter α into 50 discrete points at which the leg lengths are obtained by solving the inverse kinematics problem. Given all the set of roots apriori, then this method

would be computationally efficient as only comparison is done with no other operations done on the solutions.

One of these solutions is taken as a starting pose for the root-tracking.

$$\boldsymbol{X} = [-3.3554,\ 0.542536,\ 1.71102,\ 2.93133,\ 4.07689,\ 6.04519]^\top. \tag{C.1}$$

The path to be followed quasi-statically by the moving platform is obtained by interpolating the cable lengths between the initial and the final configurations.

$$\boldsymbol{\rho}_i = \frac{1}{2}[15,\ 20,\ 19]^\top, \tag{C.2}$$

$$\boldsymbol{\rho}_f = [10,\ 11,\ 8]^\top, \tag{C.3}$$

where the $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_f$ denote the vectors of the initial and final lengths of the cables. Following the method described in Method 2(NRalgo), the six constraint equations and its corresponding Jacobian matrix is computed.

References

[1] D. Starer, A. Nehorai, High-order polynomial root tracking algorithm, in: ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 4, 1992, pp. 465–468 vol.4. doi:10.1109/ICASSP.1992.226335.

[2] A. J. Sommese, C. W. Wampler, The Numerical Solution of Systems of Polynomials Arising in Engineering and Science, WORLD SCIENTIFIC, 2005. arXiv:https://www.worldscientific.com/doi/pdf/10.1142/5763, doi:10.1142/5763.

[3] M. Raghavan, B. Roth, Solving Polynomial Systems for the Kinematic Analysis and Synthesis of Mechanisms and Robot Manipulators, Journal of Mechanical Design 117 (B) (1995) 71–79. arXiv:https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/117/B/71/5920678/71_1.pdf, doi:10.1115/1.2836473.

[4] D. Manocha, S. Krishnan, Solving algebraic systems using matrix computations, SIGSAM Bull. 30 (4) (1996) 4–21. doi:10.1145/242961.242965.

[5] J.-P. Merlet, Interval analysis for certified numerical solution of problems in robotics, International Journal of Applied Mathematics and Computer Science 19 (3) (2009) 399–412.

[6] A. Ghosal, Robotics: Fundamental Concepts and Analysis, Oxford University Press, New Delhi,

2006.

[7] R. Stoughton, T. Arai, Optimal sensor placement for forward kinematics evaluation of a 6-DOF parallel link manipulator, in: IEEE/RSJ International Workshop on Intelligent Robots and Systems, IEEE, 1991, pp. 785–790.

[8] T. Dallej, M. Gouttefarde, N. Andreff, R. Dahmouche, P. Martinet, Vision-based modeling and control of large-dimension cable-driven parallel robots, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2012, pp. 1581–1586.

[9] P. Tempel, P. Miermeister, A. Lechler, A. Pott, Modelling of kinematics and dynamics of the IPAnema 3 cable robot for simulative analysis, in: Progress in Production Engineering, Vol. 794 of Applied Mechanics and Materials, Trans Tech Publications Ltd, 2015, pp. 419–426. doi:10.4028/www.scientific.net/AMM.794.419.

[10] P. Miermeister, A. Pott, Modelling and real-time dynamic simulation of the cable-driven parallel robot IPAnema, in: D. Pisla, M. Ceccarelli, M. Husty, B. Corves (Eds.), New Trends in Mechanism Science, Springer Netherlands, Dordrecht, 2010, pp. 353–360.

[11] J.-P. Merlet, Simulation of discrete-time controlled cable-driven parallel robots on a trajectory, IEEE Transactions on Robotics 33 (3) (2017) 675–688.

[12] P. H. Borgstrom, N. P. Borgstrom, M. J. Stealey, B. Jordan, G. Sukhatme, M. A. Batalin, W. J. Kaiser, Discrete trajectory control algorithms for NIMS3D, an autonomous underconstrained three-dimensional cabled robot, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2007, pp. 253–260.

[13] H. Abdellatif, B. Heimann, Computational efficient inverse dynamics of 6-DOF fully parallel manipulators by using the lagrangian formalism, Mechanism and Machine Theory 44 (1) (2009) 192 – 207. doi:https://doi.org/10.1016/j.mechmachtheory.2008.02.003.

[14] D. Bates, D. Brake, M. Niemerg, Paramotopy: Parameter homotopies in parallel, in: J. H. Davenport, M. Kauers, G. Labahn, J. Urban (Eds.), Mathematical Software – ICMS 2018, Springer International Publishing, Cham, 2018, pp. 28–35.

[15] A. Agarwal, C. Nasa, S. Bandyopadhyay, Dynamic singularity avoidance for parallel manipulators using a task-priority based control scheme, Mechanism and Machine Theory 96 (2016) 107 – 126. doi:https://doi.org/10.1016/j.mechmachtheory.2015.07.013.

[16] C. Nasa, S. Bandyopadhyay, Trajectory-tracking control of a planar 3-RRR parallel manipulator

with singularity avoidance, in: 13th World Congress in Mechanism and Machine Science, 2011, pp. 19–25.

[17] F. E. Udwadia, R. E. Kalaba, Analytical Dynamics: A New Approach, Cambridge University Press, Cambridge, 1996.

[18] K. V. Reddy, M. Kodati, K. Chatra, S. Bandyopadhyay, A comprehensive kinematic analysis of the double wishbone and MacPherson strut suspension systems, Mechanism and Machine Theory 105 (2016) 441–470.

[19] H. A. N. Hejase, On the use of Davidenko's method in complex root search, IEEE Transactions on Microwave Theory and Techniques 41 (1) (1993) 141–143. doi:10.1109/22.210241.

[20] M. K. Karnam, A. Baskar, R. A. Srivatsan, S. Bandyopadhyay, Computation of the safe working zones of planar and spatial parallel manipulators, Robotica (2019) 1–25.

[21] V. Ashtekar, S. Bandyopadhyay, Forward dynamics of the double-wishbone suspension mechanism using the embedded Lagrangian formulation, in: Asian MMS Conference, IFToMM Asian Mechanism and Machine Science, 2018, pp. 1–16.

[22] D. Stewart, A platform with six degrees of freedom, Proceedings of the Institution of Mechanical Engineers 180 (1) (1965) 371–386.

[23] J. S. Freeman, G. Watson, Y. E. Papelis, T. C. Lin, A. Tayyab, R. A. Romano, J. G. Kuhl, The Iowa driving simulator: An implementation and application overview, in: SAE Technical Paper, SAE International, 1995. doi:10.4271/950174.

[24] M. K. Park, M. C. Lee, K. S. Yoo, K. Son, W. S. Yoo, M. C. Han, Development of the PNU vehicle driving simulator and its performance evaluation, in: IEEE International Conference on Robotics and Automation, Vol. 3, IEEE, 2001, pp. 2325–2330.

[25] J. Pradipta, M. Klünder, M. Weickgenannt, O. Sawodny, Development of a pneumatically driven flight simulator Stewart platform using motion and force control, in: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, IEEE, 2013, pp. 158–163.

[26] G. Lebret, K. Liu, F. L. Lewis, Dynamic analysis and control of a Stewart platform manipulator, Journal of Robotic Systems 10 (5) (1993) 629–655.

[27] T.-Y. Lee, J.-K. Shim, Forward kinematics of the general 6–6 Stewart platform using algebraic elimination, Mechanism and Machine Theory 36 (9) (2001) 1073–1085.

[28] B. Dasgupta, T. Mruthyunjaya, A canonical formulation of the direct position kinematics problem for a general 6-6 Stewart platform, Mechanism and Machine Theory 29 (6) (1994) 819 – 827. doi:https://doi.org/10.1016/0094-114X(94)90081-7.

[29] T.-Y. Lee, J.-K. Shim, Improved dialytic elimination algorithm for the forward kinematics of the general Stewart–Gough platform, Mechanism and Machine Theory 38 (6) (2003) 563–577.

[30] A. Nag, S. Bandyopadhyay, A comparative study of the configuration-space and actuator-space forward dynamics of closed-loop mechanisms using the Lagrangian formalism, in: D. N. Badodkar, T. A. Dwarakanath (Eds.), Machines, Mechanism and Robotics, Springer Singapore, Singapore, 2019, pp. 95–106.

[31] S. Bandyopadhyay, A. Ghosal, Geometric characterization and parametric representation of the singularity manifold of a 6–6 Stewart platform manipulator, Mechanism and Machine Theory 41 (11) (2006) 1377–1400.

[32] R. Bostelman, J. Albus, N. Dagalakis, A. Jacoff, J. Gross, Applications of the NIST RoboCrane, in: Proceedings of the 5th International Symposium on Robotics and Manufacturing, Vol. 5, 1994, pp. 14–18.

[33] P. Garrec, J. P. Friconneau, Y. Measson, Y. Perrot, ABLE, an innovative transparent exoskeleton for the upper-limb, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 1483–1488. doi:10.1109/IROS.2008.4651012.

[34] H. Xiong, X. Diao, A review of cable-driven rehabilitation devices, Disability and Rehabilitation: Assistive Technology 0 (0) (2019) 1–13, pMID: 31287340. arXiv:https://doi.org/10.1080/17483107.2019.1629110,, doi:10.1080/17483107.2019.1629110.

[35] D. Q. Nguyen, M. Gouttefarde, O. Company, F. Pierrot, On the analysis of large-dimension reconfigurable suspended cable-driven parallel robots, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2014, pp. 5728–5735.

[36] M. Yamamoto, N. Yanai, A. Mohri, Inverse dynamics and control of crane-type manipulator, in: Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289), Vol. 2, 1999, pp. 1228–1233 vol.2. doi:10.1109/IROS.1999.812847.

[37] P. Gallina, A. Rossi, R. L. Williams II, Planar cable-direct-driven robots, part II: Dynamics and control, in: ASME Design Engineering Technical Conference, Vol. 2, 2001, pp. 1241–1247.

[38] M. Carricato, J.-P. Merlet, Geometrico-static analysis of under-constrained cable-driven parallel robots, in: Advances in Robot Kinematics: Motion in Man and Machine, Springer, 2010, pp. 309–319.

[39] G. Abbasnejad, M. Carricato, Real solutions of the direct geometrico-static problem of under-constrained cable-driven parallel robots with 3 cables: a numerical investigation, Meccanica 47 (7) (2012) 1761–1773.

[40] M. Carricato, Direct geometrico-static problem of underconstrained cable-driven parallel robots with three cables, Journal of Mechanisms and Robotics 5 (3) (2013) 031008.