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

[a], Sandipan Bandyopadhyay[a,*]

[a]*Department of Engineering Design, Indian Institute of Technology Madras, Chennai-36, Tamil Nadu, India*

Abstract

This paper presents a comparative study of three methods used to track the solutions of a system of non-linear equations. The computational time and accuracy of the methods are studied and are presented. Moreover, an *event identifier* works in tandem with the root-tracking methods to identify and estimate the singularities of the equations. The implementation of the proposed methods is illustrated using the semi-regular Stewart platform manipulator (SRSPM) following a given path. Further, the utility of the root-tracking methods together with the singularity event identifier is demonstrated using a path following problem of a 3-3 cable driven parallel robot (CDPR).

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

2010 MSC: 70B10, 70B20, 70C20, 70E60, 70Q05

Nomenclature

DoF	Degree(s)-of-freedom
CDPR	Cable-driven parallel robot
DM	Davidenkos' method
NN	Nearest neighbour
NR	Newton-Raphson
SEI	Singularity event identification
SRSPM	Semi-regular Stewart platform manipulator
$\boldsymbol{f}\ (\boldsymbol{x},\boldsymbol{y}) = \mathbf{0}$	Constraint equations

1. Introduction

Unlike linear equations, solving non-linear equations produce multiple sets of feasible solutions or roots. Root-tracking deals with identifying the branch to which a particular root belongs when

*Corresponding author

Email addresses: (), sandipan@iitm.ac.in (Sandipan Bandyopadhyay)

the parameters of the system of non-linear equations are changed. 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 as given in [1](starer1992high). There are several methods in the literature to obtain the roots of non-linear equations arising in the kinematics of manipulators, as illustrated in [2, 3](wampler2005numerical, raghavan). Such equations are generally parametric in terms of the manipulator configuration, and hence their roots vary as the manipulator‘s pose changes. In the context of parallel manipulators, the 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, as discussed in [4](ghosal2006robotics). In practical scenarios, the problem of tracking the roots are resolved either by using sensor-based methods as done in [5, 6](stoughton1991optimal, dallej2012vision) or are avoided by using alternate strategies as in [7, 8](tempel2015modelling, miermeister2010modelling). Despite these techniques, Merlet et al. in [9](merlet2017simulation) emphasise 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 cable-driven parallel robots (CDPRs), where additional constraints are imposed along with the kinematic constraints as discussed in [10](borgstrom2007discrete), to compute their feasible poses. In the context of dynamics, an actuator-space formulation is essential for real-time control applications as illustrated in [11](abdellatif2009computational). In such cases, tracking methods enable the computation of the passive joint variables, given the active joint (actuated joints) variables at each time step, using the constraint equations. Moreover, these methods are not limited to actuator and configuration (passive joint variables) space variables, and any sets of variables related via the given set of non-linear equations can be tracked using the methods described in the current work.

Parameter homotopy, described extensively in [12](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 the ways to convert the loop-closure equations into polynomials. However, such conversion leads to a set of equations with a high Bézout's number, thereby increasing the computational burden. Further, such transformations bring in additional parametric singularities, which need to be handled carefully. 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.

Moreover, as will be discussed in the subsequent sections, the proposed root-tracking methods fail at a singularity of non-linear equations as two or more branches merge, rendering the tracking of their roots infeasible. In case of parallel manipulators, singularities can be avoided using a *degree of redundancy* or an *additional degree of freedom* coupled with methods like potential functions described in [13]. However, this may require either sacrificing one of the existing *degrees of freedom* or redesigning of the manipulator. Another way to avoid singularities is if the path is checked for singularities by prior computations and is found to be "safe", however such methods may require the singularity manifold to be precomputed in closed form, which is either infeasible or computationally expensive[1].

The layout of the rest of the paper is: the problem setting and the algorithms for root-tracking are described in Section 2(sc:meth). An illustrative example demonstrating the working and implementation of the proposed methods is presented in Section 4(sc:impl). A discussion on the methods, scope and limitations is presented in Section 5(sc:disc). The utility of the root-tracking methods in the simulation of CDPRs is emphasised in Section 6(sc:CDPR). Finally, the conclusion and future work constitute Section 7(sc:conc) of the paper.

2. Root-tracking algorithms

(sc:meth) This section explains the details of the problem setting of the above mentioned root-tracking algorithms. Consider a set of n independent 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. Assuming the equations to be consistent, a finite number of unique solutions, N, can be computed solving Eq. (1). For the system of equations in Eq. (1), as mentioned earlier, there are more than one set of feasible solutions, both real and complex, 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 the solutions of these loop-closure constraints represent the forward kinematic (FK) branches which vary with the varying of the active variables.

[1]For example the singularity manifold of 3-RRR planar parallel manipulator for a given fixed orientation of the moving platform has not been derived in closed-form yet.

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 a simulation. As cited earlier, several methods in the literature for solving such equations exploit the specific form and nature of the equations. However, this work deals with tracking both the real and complex roots, given at least one set of solutions. For further discussion, it is assumed that the 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 solutions at time t, let the jth solution represented by $\boldsymbol{y}_j^t$, belong to the required branch. Then, the objective of a root-tracker is to find the solution belonging to the same branch at time $t = t + \delta t$ amongst all the N branches, $\boldsymbol{y}_1^{t+\delta t}, \boldsymbol{y}_2^{t+\delta t}, \boldsymbol{y}_3^{t+\delta t} \dots \boldsymbol{y}_N^{t+\delta t}$.

2.1. Nearest neighbour method

The nearest neighbour method further abbreviated as NN method, relies on a distance metric for identifying the roots belonging to the required branch. In the NN 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* to which the computed variables belong. For example, let the variable set involve an angle, say $\theta \in \mathbb{S}^1$. 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 NN procedure assumes the existence of a solver, `Solve`, such as the methods described in [14, 15, 16], 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 proceeds to track roots at each discrete step using the roots computed by the solver at the current step and the solution belonging to the branch in the previous step.

In Method 1(algo:NN), `Solve` returns all the roots of the input equations. The variables, $\boldsymbol{x}^i$ represents the known variables at the i^{th} step and $\boldsymbol{y}_j^i$ is the value of the j^{th} root at the i^{th} step. The function `selectMin` selects the the index s, corresponding to the minimum value element of the list. Further, $\boldsymbol{y}_s^i$ represents the root belonging to the selected branch at the ith iteration.

The major drawback in this technique is the necessity to compute all the roots up to the required accuracy, which is often memory and computationally intensive. Such a method is feasible for

Method 1 Root-tracking using the nearest neighbour method

Input: Initiate the branch with the intial solution at time $k = 0$, as $\boldsymbol{y}_s^0 \in \mathbb{C}$

Output: A list of solutions belonging to the required branch at each step

1: **procedure** NNTracker ▷ For the ith iteration

2: $\quad \boldsymbol{y}_j^i \leftarrow \texttt{Solve}(\boldsymbol{f}(\boldsymbol{x}^i, \boldsymbol{y}) = \boldsymbol{0})$ ▷ where $j = 1 \dots N$, where N is the number of solutions

3: $\quad \boldsymbol{y}_s^i \leftarrow \texttt{selectMin}(\|\boldsymbol{y}_j^i - \boldsymbol{y}_s^{i-1}\|_\infty)$ ▷ `selectMin` is as described below

4: $\quad \texttt{Append}(\boldsymbol{y}_s^i)$ ▷ Appends $\boldsymbol{y}_s$ to a list of solutions

5: $\quad \boldsymbol{y}_s^{i-1} \leftarrow \boldsymbol{y}_s^i$

6: **end procedure**

manipulators with the loop-closure constraints to leading to non-linear equations or polynomials of small order, as is the case in [13](nasa2011trajectory). Agarwal et al. in [17](agarwal2016dynamic), have also used the NN method based root-tracking strategy in a dynamic singularity avoidance control scheme of a planar 3-RRR manipulator. 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. Moreover, in reality, computing and updating only the necessary root corresponding to the required branch is sufficient. Suppose the information of all the branches is available, the NN method is the easiest to implement[2] even though it might not be computationally the fastest algorithm.

2.2. Newton-Raphson method based root-tracking

Unlike the NN method, only the root corresponding to the required branch is computed in the Newton-Raphson method based root-tracking abbreviated further as the NR method. Here, the NR method is used 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 matrix, defined as,

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

at each instant. The continued existence of the inverse of the Jacobian matrix is essential for guaranteeing the convergence of the NR method. However, the behavior of the NR method when close to a singularity is discussed in detail the context of singularity event identification.

In Method 2(algo:NR), ϵ is a predefined numerical zero, i.e., any value a is considered to be zero if $|a| \leq \epsilon$, and $\epsilon \in \mathbb{R}^+$. The description of the variables is the same as described in

[2]the NN method is easiest to implement when the variables involved, $\boldsymbol{x} \in \mathbb{C}^n$ i.e. when the L_2 norm is applicable without mapping the variables

Method 2 Root-tracking using Newton-Raphson method

Input: Initiate the branch with the initial solution at time $t = 0$, as $\boldsymbol{y}_s^0 \in \mathbb{C}$

Output: A list of solutions belonging to the required branch at each instant

procedure NRTracker ▷ For the ith iteration
  $\boldsymbol{J}(\boldsymbol{y}) \leftarrow \boldsymbol{J}(\boldsymbol{x}^i, \boldsymbol{y})$
  $\boldsymbol{f} \leftarrow \boldsymbol{f}(\boldsymbol{x}^i, \boldsymbol{y}^{i-1})$ ▷ Evaluation of the equations, $\boldsymbol{f}$
  while $\|\boldsymbol{f}\|_\infty \geq \epsilon$ **do** ▷ ϵ is the predefined numerical zero
    $\boldsymbol{J} \leftarrow \boldsymbol{J}(\boldsymbol{y}^{i-1})$
    $\delta\boldsymbol{y} \leftarrow \texttt{Solve}(\boldsymbol{J}\delta\boldsymbol{y} = \boldsymbol{f})$
    $\boldsymbol{y}_s^i \leftarrow \boldsymbol{y}^{i-1} - \delta\boldsymbol{y}$ ▷ Directly computes the roots belonging to the required branch
    $\boldsymbol{f} \leftarrow \boldsymbol{f}(\boldsymbol{y}_s^i)$
  end while
  $\texttt{Append}(\boldsymbol{y}_s^i)$ ▷ Appends $\boldsymbol{y}_s^i$ to a list of solutions
end procedure

Method 1(algo:NN). 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, NR method is the most popular method used in the literature. Since the algorithm is initiated with a known solution, only the root corresponding to this branch are calculated, eliminating the need for computing all the roots at each simulation step. This method forms the part of the root-tracking strategy used in [18](vyankatesh2018) for the dynamic simulation of the double wishbone suspension.

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

Davidenko's method reformulates the non-linear equations as an initial value problem to track the required roots. The set of equations mentioned in Eq. (1) do not have an explicit dependency on time and hence are scleronomic in nature as cited in [19]. 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 the known and unknown variables respectively. From Eq. (3), a relation between the variables $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{y}}$ can be established as,

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

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

The Eq. (4) is nothing but the Pfaffian form of the constraint equations in Eq. (1). Values of the unknown variables $\boldsymbol{y}$ are computed by solving Eq. (4) as an initial value problem using available numerical integration schemes.

Method 3 Root-tracking by Davidenko's method

Input: Initiate the branch with the intial solution at time $t = 0$ as $\boldsymbol{y}_s^0 \in \mathbb{C}$

Output: A list of solutions belonging to the required branch at each instant

1: **procedure** DMTracker ▷ For the ith iteration

2: $\Delta \boldsymbol{x} = \boldsymbol{x}^{i+1} - \boldsymbol{x}^i$

3: $\boldsymbol{y}_s^{i+1} = \boldsymbol{y}_s^i + \boldsymbol{J_{yx}}(\boldsymbol{y}_s^i, \boldsymbol{x}^i)\Delta \boldsymbol{x}$ ▷ Using an explicit Euler step for integration

4: `Append`$(\boldsymbol{y}_s^{i+1})$ ▷ Appends $\boldsymbol{y}_s^{i+1}$ to a list of solutions

5: **end procedure**

Similar to the NR method, Method 3 computes only the root corresponding to the required branch. Reddy et al. in [20](reddy2016comprehensive) have used such a technique in tracking roots of a 64 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 matrix involved in the derivation of the first order form Eq. (4) is done numerically. Hence, care must be taken if the manipulators' configuration lies outside the SWZ. As explained in [21](hejase1993use), the convergence of the first-order equations is more reliable than the NR method. Although this method relies on the error monitoring and step-size control of the ODE solver, the solutions generally diverge from the *constraint manifold* due to the errors in the numerical integration scheme used.

3. Singularity event identification

This section explains in detail the proposed *singularity event identifier* (SEI) algorithm. A system of non-linear equations is said to be singular when they have atleast one repeated root, i.e. atleast two of the solutions coincide (i.e., the two branches meet or cross-over). In the context of parallel manipulators, the degeneracy of the loop closure equations produce singularities termed as gain type or type two singularities. The singular configurations of the constraint equations manifests physically as locking of the manipulator, which might cause damage to the actuators or, in worse cases, lead to catastrophic accidents. Hence, having knowledge of the singularities is essential for

the safe continuous simulation of the system. Several methods in the literature, deal with the computation of singular configurations of parallel manipulators. At a singular configuration, as the constraint equations becomes degenerate, the $\boldsymbol{J}_{\boldsymbol{\eta\phi}}$ matrix looses its rank and hence the matrix becomes non-invertible. Methods dealing with the computation of the singular configuration using the determinant of the $\boldsymbol{J}_{\boldsymbol{\eta\phi}}$ matrix as a metric are prone to the problem of the choice of ϵ, the numerical zero to determine how close the current solution is to the singularity. Moreover, the value of the determinant does not directly translate to the distance from the singularity.

Given a path parametrised in a single variable, typically in terms of time or a geometric parameter of the path, the current singularity event identification algorithm identifies and estimates the singular configurations of a particular branch based on the distance between the roots. The SEI method is purely numerical does not require the knowledge of the singularity manifold. The working of the SEI algorithm is explained in Method 4(algo:SEI).

Method 4 Singularity event identification algorithm

Inputs: Initial solutions of all branches ($\boldsymbol{y}_j$), parametrisation of the path, (α), index of the selected root (s), parametrised path $\mathcal{P}$

Output: Singularity predicted on the path (bool), estimate of the predicted singularity (y^*)

procedure SEI ▷ For the ith iteration

  $\boldsymbol{\alpha}_{\text{stack}} \leftarrow \alpha^i$ ▷ Push the given α^i to a global stack of $\boldsymbol{\alpha}_{\text{stack}} = [\alpha^i, \alpha^{i-1}, \alpha^{i-2}]$

  $\boldsymbol{x}^i \leftarrow \mathcal{P}(\alpha^i)$ ▷ Compute the current input variables from the input parameter

  $\boldsymbol{y}^i_j \leftarrow \texttt{NRTracker}(\boldsymbol{x}^i, \boldsymbol{y}^{i-1}_j)$ ▷ where $j = 1 \dots N$, where N is the number of solutions

  $\boldsymbol{d}^i_{sj} \leftarrow \|\boldsymbol{y}^i_s - \boldsymbol{y}^i_j\|$ ▷ Computing the distance between the selected and the other roots

  $\boldsymbol{d}_{\text{stack}} \leftarrow \boldsymbol{d}^i$ ▷ Push the given $\boldsymbol{d}^i$ to a global stack of $\boldsymbol{d}_{\text{stack}} = [\boldsymbol{d}^i, \boldsymbol{d}^{i-1}, \boldsymbol{d}^{i-2}]$

  $\nabla \boldsymbol{d}^i = \frac{\Delta \boldsymbol{d}^i}{\Delta \alpha^i} = \frac{\boldsymbol{d}^i - \boldsymbol{d}^{i-1}}{\alpha^i - \alpha^{i-1}}$ ▷ Computing the gradient of the distances

  $\boldsymbol{d}^{i+1} = \boldsymbol{d}^i + \nabla \boldsymbol{d}^i \Delta \alpha^i$ ▷ Predicting distance to one step in the future

  if $\boldsymbol{d}^{i+1}_j < 0$ **then** ▷ Checking for a negative distance

    $f = \texttt{fitPoly}(\boldsymbol{d}_{stack}, \alpha_{stack})$ ▷ Extrapolation function between $\boldsymbol{d}$ and α, i.e., $d = f(\alpha)$

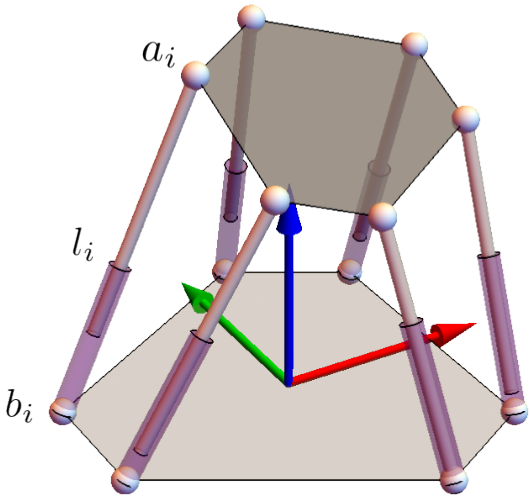
    $f(\alpha_{est}) = 0$ ▷ Estimating the α for which the $\boldsymbol{d}$ is zero

    $\boldsymbol{x}_{est} \leftarrow \mathcal{P}(\alpha_{est})$ ▷ Compute the corresponding manipulator configuration

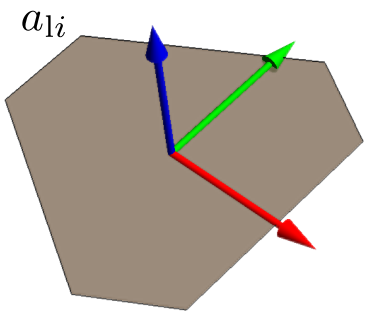
    return(s, j, $\boldsymbol{x}_{est}$) ▷ Intersection between branches s, j

  end if

end procedure



(a) Architecture of an SRSPM



(b) Local axes of the top platform

Figure 1: Description of the SRSPM ({red,green,blue}$\equiv \{x, y, z\}$ axes)

4. Illustrative examples

(sc:impl) The current section discusses the implementation details of all the algorithms and is demonstrated by solving a path following problem of an SRSPM.

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. 1(fig:srspm1). Since its introduction in [22](stewart1965platform), this manipulator has attracted a large amount of research on its kinematics, dynamics and control. The interest in this particular manipulator stems from its wide range of applications including automotive simulators as shown in [23, 24](freeman1995iowa, park2001development), flight simulators described in [25](pradipta2013development), machine tools as illustrated in [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 dealing with the FK problem, e.g., in [14, 15](lee2001forward, lee2003improved) and more recently in [16](NAG2021104090). During the process of solving the FK problem, a *forward kinematic univariate* (FKU) is formulated. The FKU is parametrised in terms of the actuated variables, therefore, roots of the polynomial changes as the manipulator moves. One such instance of deriving a 20 de-

gree polynomial FKU in the context of FK of an SPM is described in [27](nag2019comparative). Root-tracking methods can also be used for tracking roots of FKU obtained during the FK process.

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. In the case of an SRSPM the constraints may be formulated by equating the positions of the vertices of the moving platform as traversed through each individual serial chain present along the legs,

$$\boldsymbol{\eta}(\boldsymbol{X}, \boldsymbol{q}) = \boldsymbol{0}, \text{where } \boldsymbol{X} = [x, y, z, c_1, c_2, c_3]^\top, \tag{7}$$

$$\boldsymbol{x} + \boldsymbol{R}_{tp}\boldsymbol{a}_{li} - \boldsymbol{a}_i = \boldsymbol{0}, \tag{8}$$

$$\boldsymbol{a}_i = \boldsymbol{b}_i + \boldsymbol{R}_y(\boldsymbol{\phi}_i)\boldsymbol{R}_x(\boldsymbol{\psi}_i)\boldsymbol{l}_i, \quad \text{where } i = 1, 2, \ldots 6, \tag{9}$$

$\boldsymbol{x}$ is the position of the geometric centre $\{x, y, z\}$ and $\boldsymbol{R}_{tp}$ is the rotation matrix corresponding to the orientation of the moving platform; $\boldsymbol{a}_i$, $\boldsymbol{a}_{li}$ are the vertices of the moving platform described in the global reference frame and a local reference frame fixed to the moving platform respectively. A pictorial representation of the formulation of the constraint equations is shown in Fig. 2(fig:constraints).

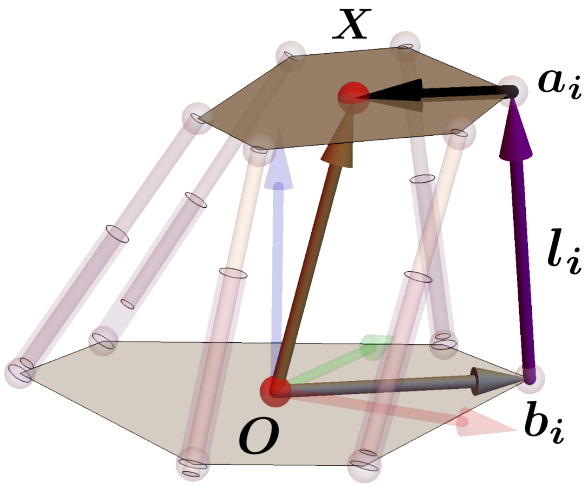


Figure 2: Illustration of the formulation of constraint equations of the SRSPM in 24 variables.

The computational speed and accuracy of the methods are evaluated by simulating an SRSPM following a given path. The physical parameters of the SRSPM used is given in Table 1(tab:srspm).

The path followed by the centroid of the manipulator is ensured to be within the SWZ. The chosen path is a circle parametrised by a path variable α; its complete description is given in Eq. (10).

$$\boldsymbol{X} = \mathrm{k} \times [\cos(\alpha), \sin(\alpha), 6.4, \cos(\alpha), \sin(\alpha), \sin(2\alpha)]^\top, \tag{10}$$

Table 1: Parameters of the SRSPM used for the demonstration of the root-tracking algorithms

Parameter	Symbol	Value	Unit
Circumradius of the base platform	r_{b}	1.000	m
Circumradius of the moving platform	r_{t}	0.580	m
Angular spacing between the adjacent pair of legs of the fixed platform	γ_{b}	0.298	rad
Angular spacing between the adjacent pair of legs of the moving platform	γ_{t}	0.657	rad
Length of the sliding link of the prismatic joint	$l_{\mathrm{b_i}}, i = (1, \ldots, 6)$	0.500	m
Length of the fixed link of the prismatic joint	$l_{\mathrm{a_i}}, i = (1, \ldots, 6)$	1.500	m

where the parameter k, a measure of scale, is set to 0.2 and the variable $\alpha \in [0, 2\pi]$ is discretised into 50 steps. The leg lengths corresponding to each steps are obtained by solving the inverse kinematic (IK) problem. To illustrate the repeatability and root independence of the methods, all the FK roots corresponding to the initial leg values are tracked. The tracked paths corresponding to all the real roots are illustrated in Fig. 4(fig:allroots). A list of all the 14 roots tracked can be found at `https://github.com/akhilsathuluri/root_tracking/tree/master/assets/data/roots_tracked_real_and_complex`.

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 GB RAM machine installed with Ubuntu 18.04.1 LTS using `clang` compiler with version 10.0.0. All the execution times reported are the average values of 1000 executions which were run only on a single core. The `C++` implementations and the corresponding `doxygen` documentation is made available and can be accessed at `https://github.com/akhilsathuluri/root_tracking` .

4.1.2.

$$\mathcal{S}_{\boldsymbol{p}} \;=\; 0.007\; y - 0.010\; xy + 0.036\; z + 0.067\; xz - 0.032\; y^2 z + 0.235\; yz^2 - 0.381\; z^3 \qquad (11)$$

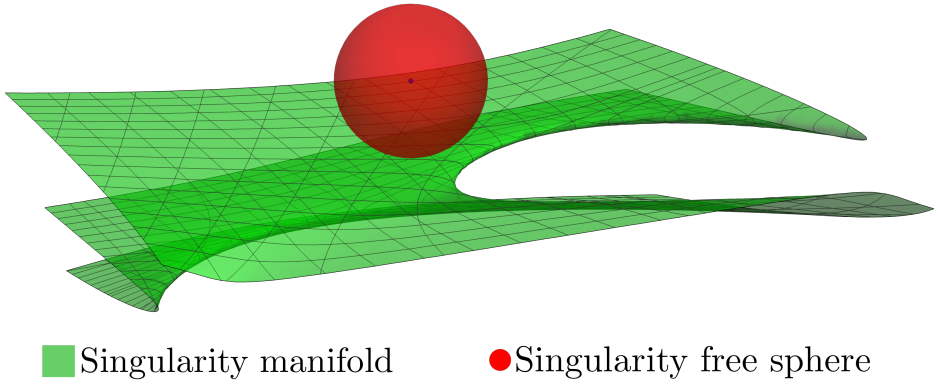
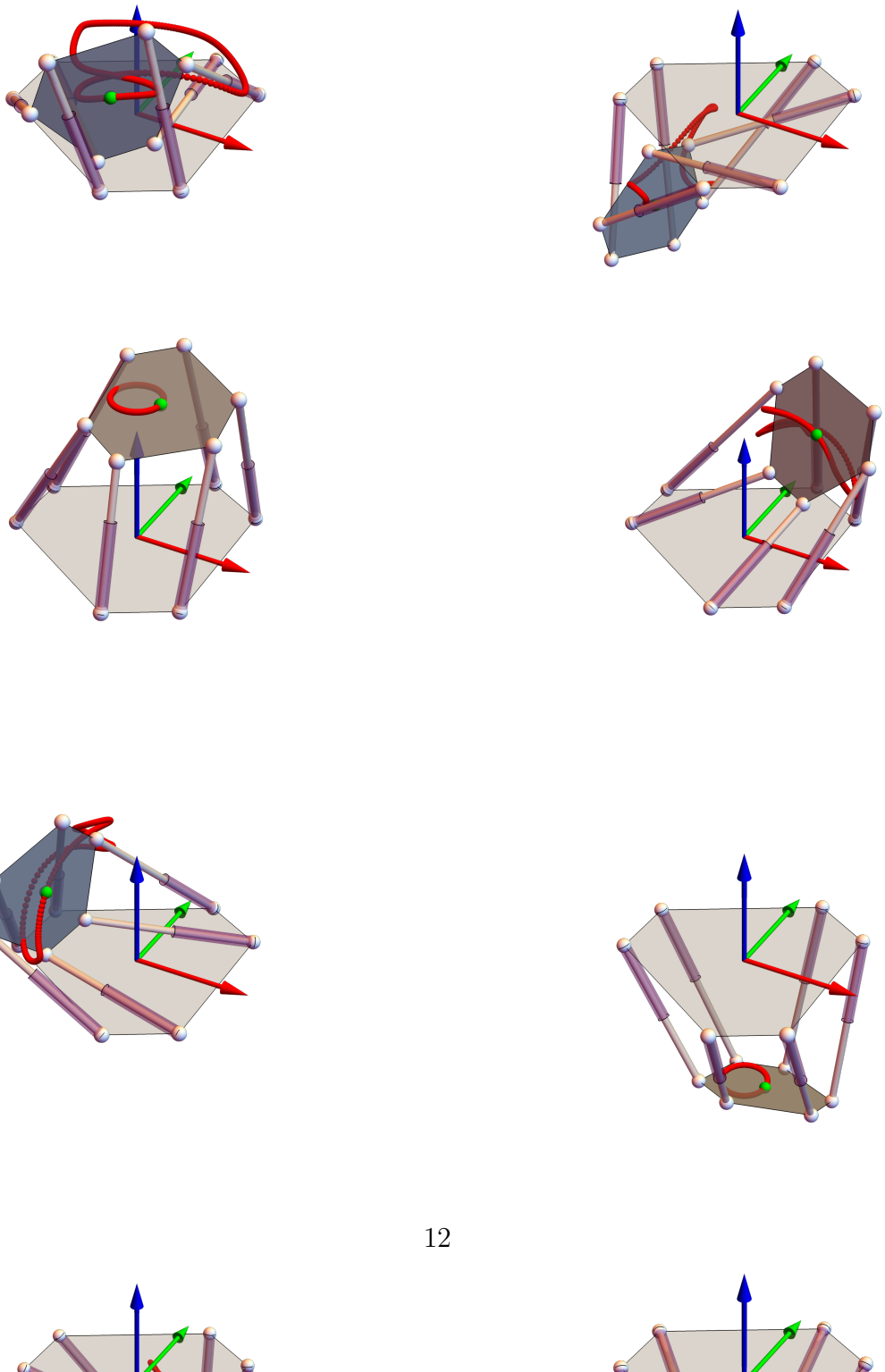


Figure 3: The singularity free region represented as a sphere (red) and the constraint manifold (green) of an SRSPM.



4.2. Example 1: Comparison between root-tracking algorithms

This subsection presents a comparison between various algorithms in terms of their computational speed and accuracy. The known solution used to initiate the root-tracking methods is given in Eq. (12).

$$\begin{aligned}\boldsymbol{\phi} = \boldsymbol{y}_0 = [&0.003, -0.067, 0.457, 0.547, -0.095, 0.003,\\ &0.336, 0.287, -0.021, 0.025, -0.395, -0.380]^\top\\ \boldsymbol{\theta} = \boldsymbol{x}_0 = [&1.446, 1.567, 1.579, 1.339, 1.152, 1.287]^\top\end{aligned} \tag{12}$$

The tracked task-space coordinates of the selected branch are illustrated in Fig. 5(fig:circlemotion).

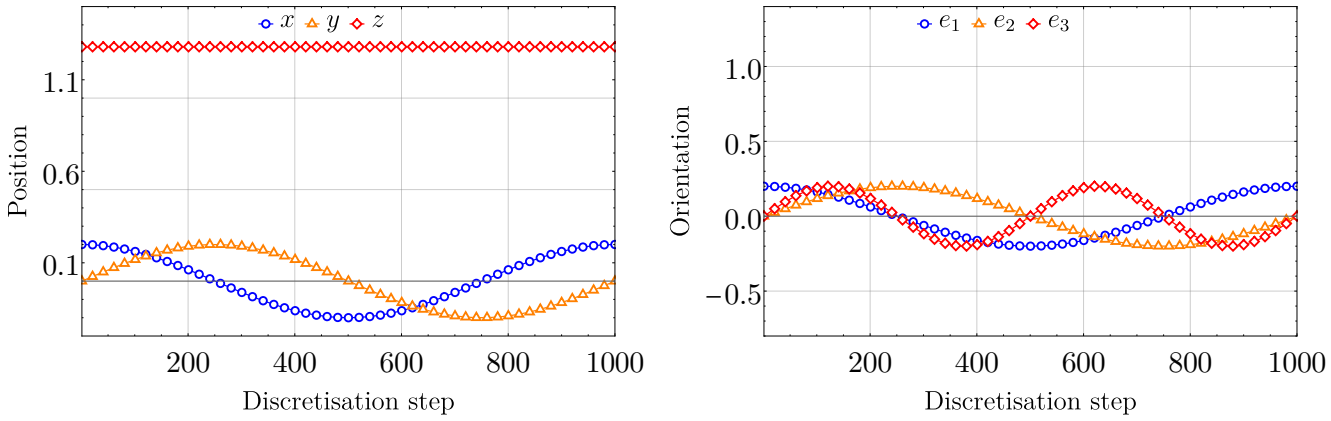


Figure 5: The task-space coordinates of the tracked selected branch

4.2.1. Nearest neighbour method

Since this method relies on the `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 6.437 ms. It is important to note that the time taken in computing the roots is included in the mentioned root-tracking time. If the roots at all discretisation points have been computed a priory, then the nearest neighbour method would be the easiest to implement, as explained earlier.

4.2.2. Davidenko's method or integration of the first order form 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, 3. In Method 3, an explicit Euler integration scheme is used to integrate the obtained ODE equations, Eq. (4). Method 3 took 0.452 ms 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 quantifies the *drift* in the value of the constraint equations, i.e. the measure of how much the constraints are violated by the computed roots, and is defined as,

$$e_1 = \|\boldsymbol{\eta}(\boldsymbol{q})\|_\infty, \tag{13}$$

where $\|\cdot\|_\infty$ is the L_∞ norm. The error e_1 given in Eq. (13), plotted against the discretisation step is shown in Fig. 6(e1NN).

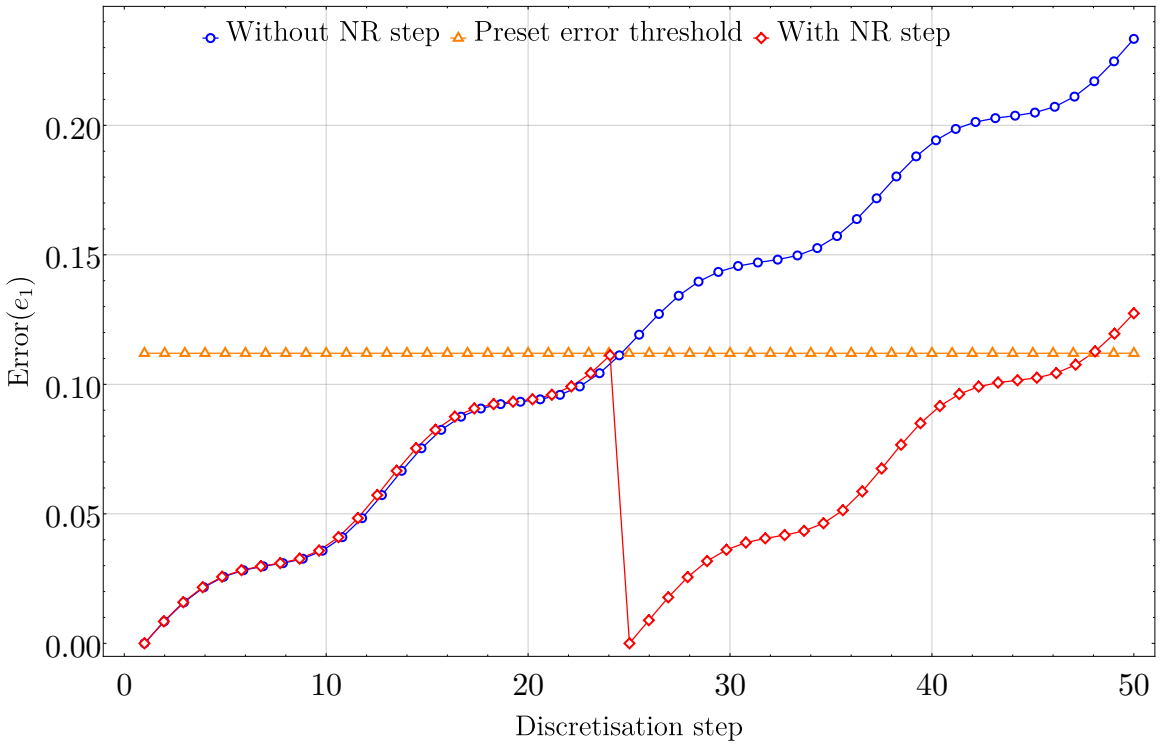


Figure 6: Illustration of variation in the error measure (e_1) given in Eq. (13), with and without the use of an intermediate step using the NR method along with the predefined tolerance bound denoted by the dashed line.

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 NR method. However, with time, the solutions start to drift from the actual values violating the loop-closure constraints as shown in Fig. 6(e1NN). The characteristics of the drift depend on the numerical integration technique used. A Newton-Raphson method based correction step is optionally 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. 6(e1NN).

4.2.3. Newton-Raphson method for root-tracking

For the same path described earlier, NR method takes 0.884 ms to track the required root through the entire simulation. The error e_1 for the solutions computed using the NR method is 10^{-10}. This error value can be set using the tolerance value for convergence, ϵ, used in Method 2. The Table 2(comparetable) summarises the computational time and accuracy of all three methods.

Table 2: Computational time taken and the accuracy of the solutions corresponding the root-tracking methods used to solve the path following problem of the SRSPM

Root-tracking method	**Computational time (ms)**	**Accuracy (Error e_1)**	**Tunable parameter**
Method 1(algo:NN)	6.437*	10^{-10}	FK algorithm used
Method 2(algo:NR)	0.884	10^{-10}	Accuracy goal used
Method 3(algo:I)	0.508	10^{-1}	Integration scheme used

* Includes the tracking time (time taken for comparison of the roots) plus the time taken by `Solve` to compute the FK solutions. The later is the bottleneck in using the nearest neighbour method. The time presented uses Method 2(algo:NR) as the `Solve` method.

To obtain highly accurate solutions to the FK problem, special care must be taken like the use of multi-precision representation of variables as detailed in [15](lee2003improved). However, such computations are generally time and resource intensive. Therefore, the time taken to compute the roots is included in the tracking time of the Method 1.

4.3.

(sec:SEIexample) The root-tracking algorithms work well for paths defined within the SWZ as the FK solvers behave well within this region. However, when the path nears the singular configuration, the behaviour of the root-tracking algorithm is unpredictable, as the constraint Jacobian matrix loses a rank, and therefore, its inverse is not defined. One such situation is illustrated in the Fig. 7(fig:linemanifold).

The above example motivates the necessity to identify the singular configurations ahead of the path to prevent the manipulator from becoming singular which could be hazardous in practical scenario. Therefore, a *singularity event identifier* (SEI) algorithm is proposed to not only identify the upcoming singular configurations but also estimate the precise location of the singularity. To demonstrate the working of the SEI, a circular path is chosen that deliberately intersects the singularity manifold. The chosen circular path along with the manipulator is shown in Fig. 8(fig:circlemanifold). It is important to note that the analytical singularity manifold is used only for verification and illustration of the SEI algorithm and SEI does not require any knowledge about the singularity manifold.

This section illustrates the implementation of the singularity event identification algorithm to identify and estimate the singular configuration of the SRSPM as it moves along a given path.

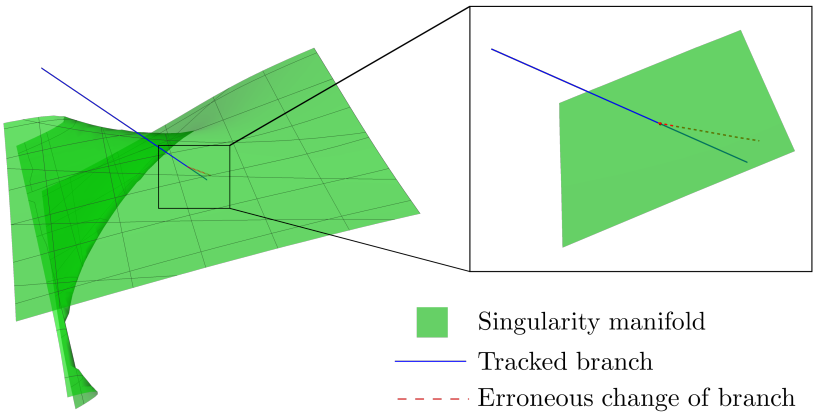


Figure 7: Illustration of the behaviour of the Method 2(algo:NR) when the tracked path intersects the singularity manifold

For the sake of completeness, the estimate of the singular configuration is compared with the analytical singular configuration using the singularity manifold derived using the procedure described in [28](bandyopadhyay2006geometric). The working of the singularity event identification for a chosen circular path deliberately intersecting the singularity manifold is shown in Fig. 8(fig:circlemanifold).

4.3.1. Example 2: Straight line path

4.3.2. Example 3: Circular path

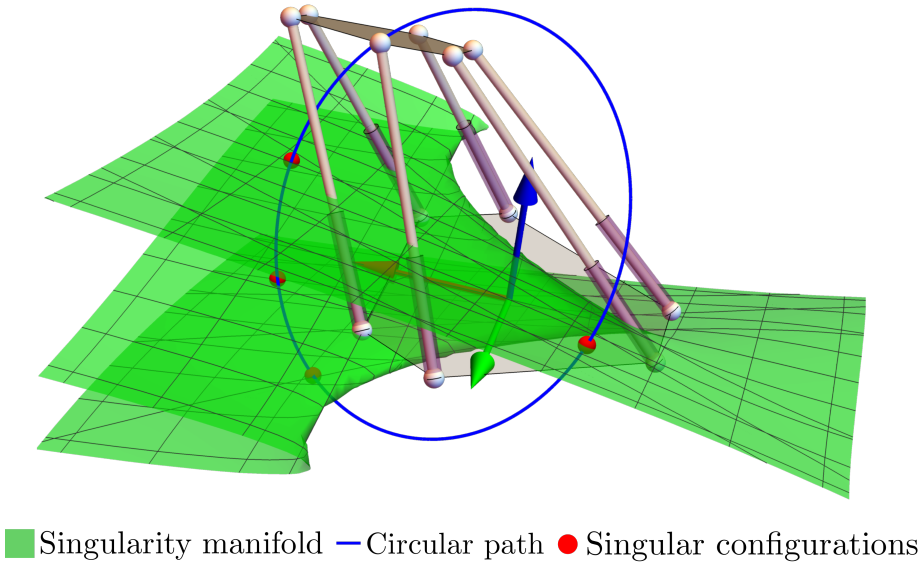


Figure 8: Illustration of the SRSPM and the circular path intersecting the singularity manifold

5. Discussion

(sc:disc) This section contains a discussion on the behaviour and the computational aspects of various root-tracking methods. At a singularity, the branches of the FK solutions merge and hence cannot be resolved using any of the previously discussed root-tracking methods. Therefore, all the methods fail at or close to the location of the singularity. Moreover, singularity affects all the techniques and cannot be avoided by choosing any particular method. 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 as shown in Fig. 9(nn). Such a similar instance is demonstrated in Section 4.3(sec:SEIexample) in the context of the NR method. Moreover, in some cases, even at the singular configuration, the inverse of the 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 redundantly actuated manipulator shift at the singularity, causing a significant change in the position and orientation of the moving platform as discussed in [9].

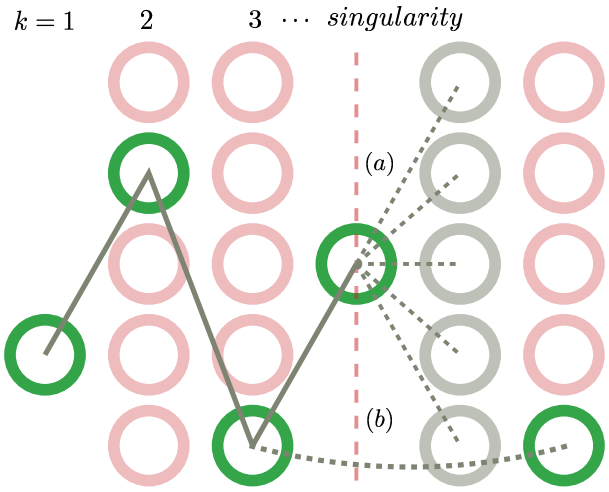


Figure 9: Pictorial representation of the nearest neighbour method, illustrating, (a) The issue of resolving the branches at a singularity, (b) The possibility of jumping ahead of a singularity.

The SWZ as described in [29](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. The assumption of remaining within the SWZ ensures the unique existence of solutions for all proposed methods. Therefore, the roots can be tracked at all times within the SWZ. However, the computation of the SWZ is non-trivial and requires extensive computational analysis. The proposed SEI algorithm identifies and estimates the singular configurations that locally lie within the selected path. The SEI algorithm can be generalised by incorporating additional constraints to handle the issues of joint constraints and self-

interference; however, this is not in the scope of this paper. Such a scheme would have an advantage over the SWZ in utilising a larger workspace as it is not confined by the choice of the convex region representing the SWZ. However, the inherent difference between the approaches should be noted; the SEI algorithm works locally, whereas SWZ globally establishes bounds on the workspace.

As mentioned earlier in Section 4.3(sec:SEIexample), for Methods 2(algo:NR), 3(algo:I) 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 matrix of the system is regular. Therefore, the implicit function theorem ensures that the NR method converges to a unique solution within the SWZ for a chosen, sufficiently small step size, as elaborated in [9](merlet2017simulation).

The numerical integration step used in Method 3(algo:I) 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. 6(e1NN) as the deviation of the solutions from the constraint manifold. Whenever the drift exceeds a particular threshold value, the NR method 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 solutions 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. in [18](vyankatesh2018) have suggested using the analytical FK formulation to compute all the sets of roots at the failed step and resume the use of root-trackers from the next instant. This strategy ensures the robust working of the simulation.

6. Applications in the simulation of cable-driven parallel robots

(sc:CDPR)

CDPRs 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 amongst the class of parallel manipulators, unlike their rigid body counterparts, cables only support tension, making analysis and design of CDPRs much more challenging. The interest in CDPRs is motivated from their potential applications in material handling [30](bostelman1994applications), exoskeletons [31](garrec2008able), rehabilitation [32](cablerehab) and re-configurable robotics [33](nguyen2014a

The FK 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 [34, 35](yamamoto1999inverse, gallina2001planar). Unlike parallel robots with rigid membered links, in CDPRs, the loop closure equations must be solved along with the conditions of static equilibrium, which leads to the formulation of a *geometrico-static* or *forward kineto-static problem* (FKS). Further, additional constraints on unilaterality of the cable forces should be imposed on the obtained solutions to compute the physically feasible configurations amongst all the computed solutions.

6.1. The 3-3 cable-driven parallel manipulator

(cdprfk) Consider the under-constrained spatial 3-3 CDPR presented in [36](carricato2010geometrico), where the mobile platform is connected to the fixed base via 3 non-deformable cables, as shown in Fig. 10(fig:carricato). Carricato et al. in [36](carricato2010geometrico) have illustrated the complexity of the FKS problem to obtain feasible poses.

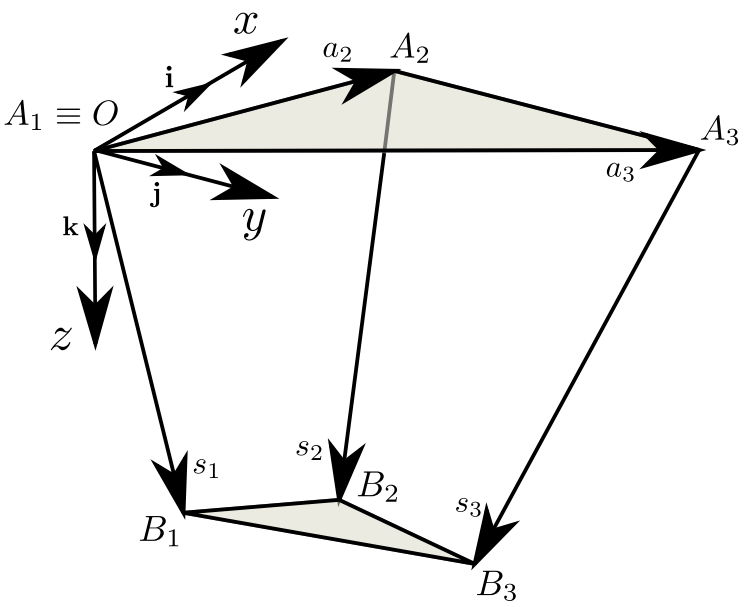


Figure 10: Reproduced architecture of the 3-3 CDPR as presented in [36]

For the treatment of a general redundant manipulator dealing with the change of the operating cables during the motion, one should refer to the discussion presented in [9]. Three loop-closure equations are derived by enforcing the constraints on the lengths of the three cables, A_iB_i. Following the methodology described in [36](carricato2010geometrico); further, a set of 12 equations are obtained by imposing 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,

$$\boldsymbol{X} = [x, y, z, e_1, e_2, e_3]^\top. \tag{14}$$

The detailed description of the formulated equations can be found in Eq. (4), (5) and Section 4 in [36](carricato2010geometrico). The selection of any 3 of these equations along with the 3 loop-closure constraints forms a valid system of 6 equations in 6 unknowns. Abbasnejad et al. in [37](abbasnejad2012real) have illustrated the introduction of spurious roots while solving the FKS problem using the above-formulated set of equations. However, since the root-tracking methods are initialised with a known configuration, they do not face such issues and provide an accurate estimate of the solution at all instants of time. 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 satisfy all the 15 constraint equations.

6.2. Root-tracking problem

For the demonstration of the root-tracking problem, the CDPR is assumed to be moving quasistatically and hence the dynamic effects are ignored. The physical parameters of the manipulator used for simulation are given in Table 3(tab:cdpr).

Table 3: Physical parameters used for the root-tracking problem of the Cable driven parallel manipulator

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	$\boldsymbol{b}_1$	$[1, 0, 0]^\top$
	$\boldsymbol{b}_2$	$[0, 1, 0]^\top$
	$\boldsymbol{b}_3$	$[0, 0, 1]^\top$

The FKS problem of a 3-3 CDPR formulated in Section 6.1(cdprfk) yields 156 solutions. All the obtained solutions are checked to satisfy the unilaterality constraints, and only the poses which admit positive tensions in all the cables are selected. This check is necessary to ensure the physical feasibility of the computed FKS solutions. However, this additional step at each iteration increases the computational burden. The accuracy of the solution is critical to ensure safe operation, especially in the case of a CDPR. Therefore root-tracking methods have a clear advantage in their implementation in problems dealing with CDPRs.

6.3. Method 2(algo:NR):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. One of such solutions is selected as the initial configuration, starting from which the CDPR follows a given path P. The tracked path is obtained through first order interpolation between initial and final cable lengths; the numerical details of which are given in Eq. (15),

$$\begin{aligned} \boldsymbol{\rho}_i &= \frac{1}{2}[15,\ 20,\ 19]^\top, \\ \boldsymbol{\rho}_f &= [10,\ 11,\ 8]^\top, \end{aligned} \tag{15}$$

where $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_f$ denote the vectors of the initial and final lengths of the cables. The resulting path followed by the CDPR is illustrated in Fig. 11(fig:trackedCDPR). The root-tracking problem is initiated with a feasible FKS solution reported in [38](carricato2013direct) and is also given in Eq. (16).

$$\boldsymbol{X} = [2.931, 4.077, 6.045, -3.355, 0.542, 1.711]^\top. \tag{16}$$

Moreover, the accuracy of the computed solutions described in terms of error e_1 are of the order 10^{-11}. The magnitude of error is comparable to the results reported in [38], which were obtained by solving the FKS problem using higher precision computations and there by requiring a higher computational overload than the root-tracking algorithms. For the discretisation of 1000 steps, the time taken to track is 0.0619 s, for a required precision of 10^{-10}. The red curve shown in Fig. 11(fig:trackedCDPR), is the path followed by the manipulator.

Moreover, the values of the tracked variables for initial position and orientation given by Eq. (16) is presented in Fig. 12(fig:trackedvars).

Notably the root-trackers not only have a computational advantage, all the computed roots belonging to the required branch also admit only positive tensions as shown in Fig. 13(fig:tensions). Therefore the computed solutions are feasible and realisable on a physical system.

7. 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 increases, it is difficult to solve the FK or FKS problems iteratively, as in CDPRs. A path following application is demonstrated for the 3-3 CDPR.

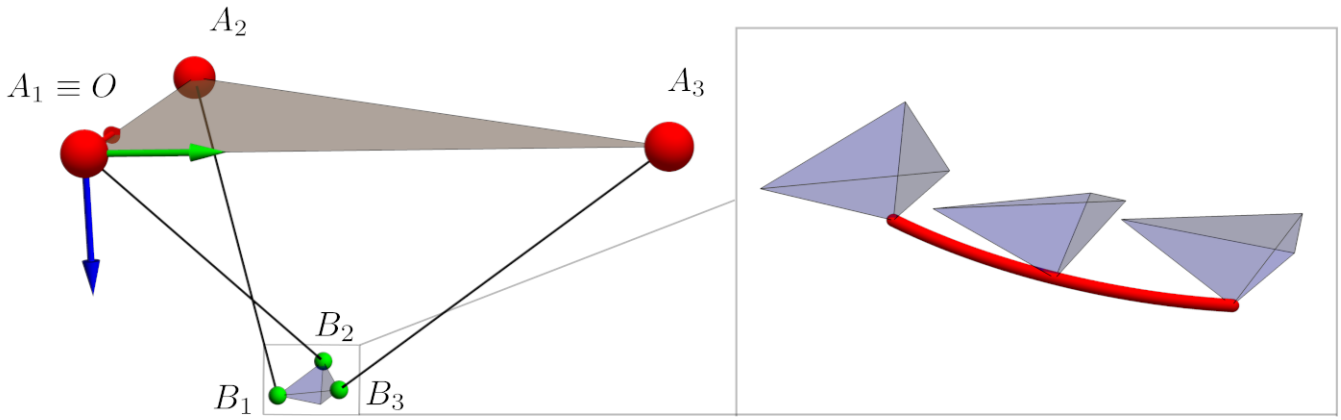


Figure 11: The construction of the 3-3 CDPR and the zoomed in view of the path followed by the moving platform.

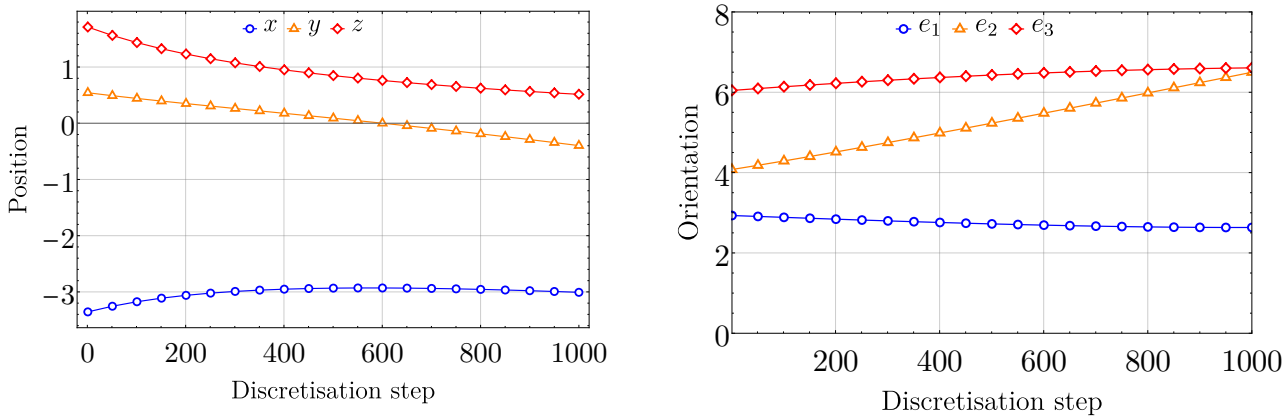


Figure 12: Variation of the task-space variables of the selected branch of the CDPR, tracked using the NR method

Moreover, a singularity event identification algorithm is proposed, and its utility is discussed in the context of SRSPM and CDPR. However, it should be noted that the proposed methods are general and are not limited to the scope of parallel manipulators. The proposed methods allow the simulation of the dynamics of parallel manipulators efficiently with any set of generalised coordinates used to represent the system. Faster computations allow higher control bandwidths and hence result in better tracking performance of high-frequency motions. In the case of reduced-order or sub-space mapped models, root-trackers find their advantage as state observers used to compute the full state values using the constraint equations, functioning as soft sensors.

8. Acknowledgement

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.

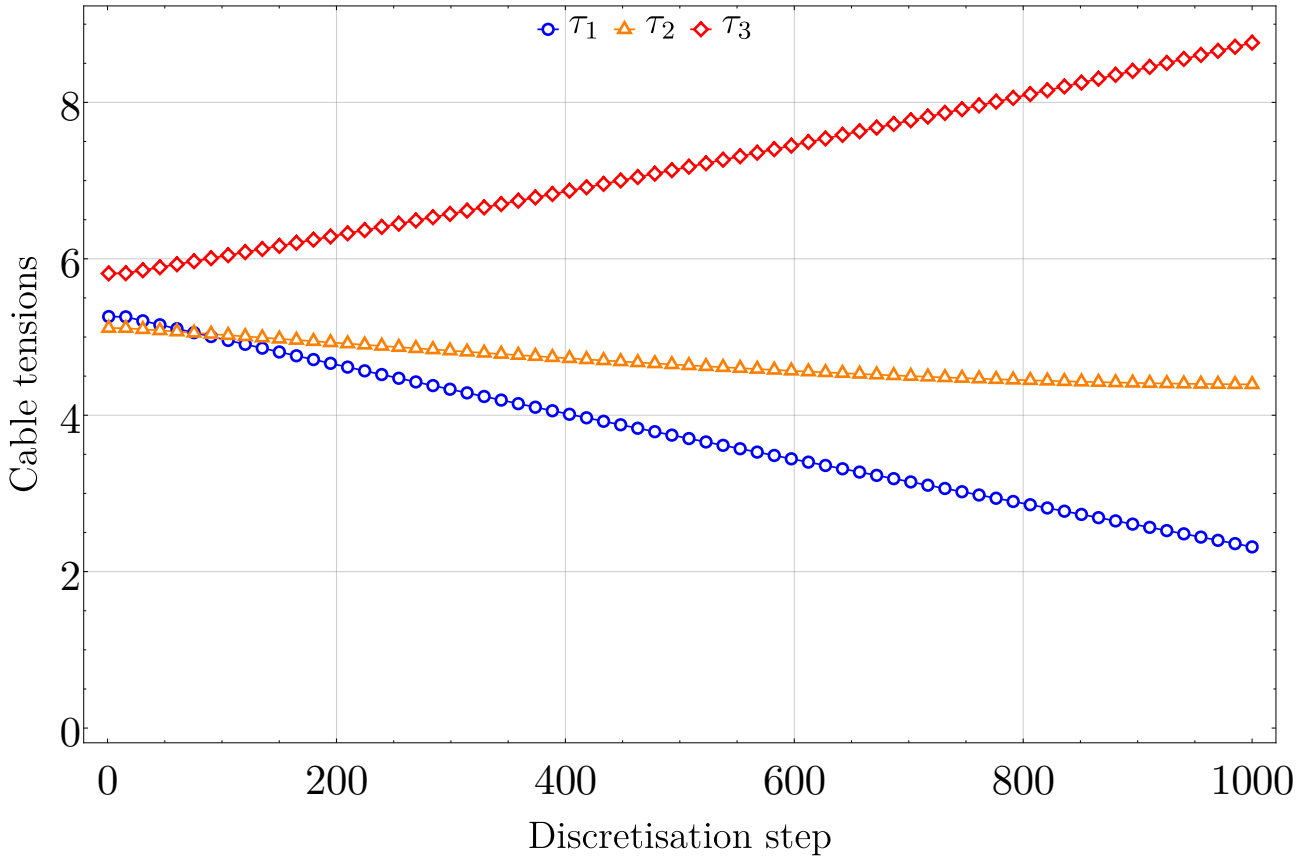


Figure 13: The variation of the positive tensions in the cables as the manipulator tracks the required path.

1992.226335.

[2] A. J. Sommese, C. W. Wampler, The Numerical Solution of Systems of Polynomials Arising in Engineering and Science, WORLD SCIENTIFIC, 2005. 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. doi:10.1115/1.2836473.

[4] A. Ghosal, Robotics: Fundamental Concepts and Analysis, Oxford University Press, New Delhi, 2006.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. Nag, S. V, S. Bandyopadhyay, A uniform geometric-algebraic framework for the forward kinematic analysis of 6-6 stewart platform manipulators of various architectures and other related 6-6 spatial manipulators, Mechanism and Machine Theory 155 (2021) 104090. doi:https://doi.org/10.1016/j.mechmachtheory.2020.104090.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[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] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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