ICINCO 2020

17th International Conference on Informatics in Control, Automation and Robotics

PROCEEDINGS

7 - 9 July, 2020

EDITORS

Oleg Gusikhin

Kurosh Madani

Janan Zaytoon

http://www.icinco.org

SPONSORED BY

INSTICC

PAPERS AVAILABLE AT

SCITEPRESS DIGITAL LIBRARY

ICINCO 2020

Proceedings of the
17th International Conference on
Informatics in Control, Automation and Robotics

July 7 - 9, 2020

Sponsored by
INSTICC - Institute for Systems and Technologies of Information, Control and Communication

ACM In Cooperation
ACM SIGAI - ACM Special Interest Group on Artificial Intelligence

In Cooperation with
AAAI - Association for the Advancement of Artificial Intelligence
euRobotics AISBL
SPR - Portuguese Robotics Society
INNS - International Neural Network Society
BARA - British Automation and Robot Association

Edited by Oleg Gusikhin, Kurosh Madani and Janan Zaytoon

Printed in Portugal
ISSN: 2184-2809
ISBN: 978-989-758-442-8
Depósito Legal: 470573/20

http://www.icinco.org
icinco.secretariat@insticc.org

CONTENTS

SHORT PAPERS

ROBOTICS AND AUTOMATION

FULL PAPERS

SHORT PAPERS

SHORT PAPERS

Solution of the Forward Kinematic Problem of 3UPS-PU Parallel Manipulators based on Constraint Curves

Adrián Peidró[a], Luis Payá[b], Sergio Cebollada[c], Vicente Román[d] and Óscar Reinoso[e]
Automation, Robotics and Computer Vision Laboratory,
Miguel Hernández University, Avda. de la Universidad s/n, 03202 Elche, Spain
{*apeidro, lpaya, sergio.cebollada, v.roman, o.reinoso*}*@umh.es*

Keywords: Forward Kinematics, k-d Trees, Parallel Robots, Algebraic Elimination, Curve Rendering, Angular Wrapping.

Abstract: Algebraic elimination methods for solving the forward kinematic problem of parallel manipulators are fast and obtain all solutions, but they require eliminating all unknowns except one, and solving a high-degree univariate polynomial whose coefficients often have expressions too complex to be obtained symbolically. This prevents parameterizing these coefficients in terms of all the kinematic parameters involved, which requires repeating the elimination process again whenever these kinematic parameters change. To avoid this, this paper presents an new method to solve the forward kinematics of 3UPS-PU parallel manipulators by eliminating only one unknown, reducing the system to an easily parameterizable set of planar constraint curves in the space of the remaining unknowns, which contain all real solutions of the forward kinematics. By sampling points from these curves densely, and sorting the sampled points using k-d trees, the proposed method manages to search all real solutions along these curves. The proposed method is compared to previous methods that obtain all solutions and is shown to perform about 100 times faster than these methods, but is less general than them.

1 INTRODUCTION

This paper presents a method for quickly obtaining all real solutions of the forward kinematic problem of 3UPS-PU parallel manipulators. This manipulator, which has been studied by several authors, is part of the well-known Tricept machining robot (Neumann, 1988) which has been widespread in industry.

Siciliano (1999) analyzed the inverse kinematics and manipulability of the Tricept robot. Fattah and Kasaei (2000) modelled the dynamics of 3UPS-PU parallel robots using Newton-Euler equations and the natural orthogonal complement method. Caccavale et al. (2003) also modeled the dynamics of the Tricept robot, and designed an impedance controller for interacting with the environment.

Kolláth et al. (2009) studied the workspace and positioning accuracy of this robot. Hosseini and Daniali (2011) studied the proximity to singularities of the Tricept manipulator, and optimized its design to maximize workspace volume. Hu (2014) analyzed the workspace and forward/inverse kinematics at position, velocity, and acceleration levels, for a 6-DOF serial-parallel robot composed of two 3UPS-PU parallel robots connected in series.

Regarding singularities of this robot, Liu and Hsu (2007) and Peidró et al. (2019) analyzed its singularity locus in the output and input space, respectively, characterizing some important singularities. Lu et al. (2007) and Pendar et al. (2008) also analyzed some notable singularities of this robot.

Concerning the forward kinematic problem, Joshi and Tsai (2003) solved this problem for 3UPS-PU robots with flat platforms through algebraic elimination, obtaining the symbolic expression of a univariate polynomial whose roots are the solutions of the forward kinematics. Using also algebraic elimination, Innocenti and Wenger (2006) solved the forward kinematics of the general family of 3UPS-PRR manipulators (which includes 3UPS-PU as a special case), reducing the problem to a 28-th degree polynomial.

Algebraic elimination methods are widely used for solving the forward kinematics of parallel manipulators, due to their advantages: they boil down to finding the roots of a polynomial, which is a mature problem for which efficient root-finding techniques exist. They obtain all solutions, including nonreal

[a] https://orcid.org/0000-0002-4565-496X
[b] https://orcid.org/0000-0002-3045-4316
[c] https://orcid.org/0000-0003-4047-3841
[d] https://orcid.org/0000-0002-3706-8725
[e] https://orcid.org/0000-0002-1065-8944

Peidró, A., Payá, L., Cebollada, S., Román, V. and Reinoso, Ó.
Solution of the Forward Kinematic Problem of 3UPS-PU Parallel Manipulators based on Constraint Curves.
In *Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2020)*, pages 324-334
ISBN: 978-989-758-442-8

ones. Lastly, they are very fast: once one obtains and codes the symbolic expression of the coefficients of the polynomial in terms of all parameters (joint displacements, links' lengths, etc.), one only needs to substitute the numerical values of these parameters into the expressions and get the roots of the polynomial (e.g., using command `roots` in Matlab).

However, elimination methods also have disadvantages: the process of eliminating unknowns until arriving at a univariate polynomial may introduce several spurious solutions, and the degree of such polynomial may increase quickly, especially if the elimination steps are not performed carefully. Furthermore, the symbolic expression of the coefficients of the final polynomial can be tremendous, with expressions so large and complex that span several pages (Joshi and Tsai, 2003) in the most optimistic cases. In other cases, they become too complicated to be obtained symbolically even with the aid of computer algebra, having to settle for numerical coefficients instead, obtained by giving concrete or random values to all parameters (Innocenti and Wenger, 2006).

When implementing programming code for the resolution of the kinematic problems of parallel manipulators, with the aim of simulating these problems for research or educational purposes (Peidró et al., 2016), it is desirable to implement resolution methods that provide all solutions (to provide information as complete as possible) and, at the same time, obtain these solutions as quick as possible, so that the user can change some kinematic parameter and instantaneously get the new updated solutions. These requirements are best satisfied by elimination methods, which require typing the symbolic expression of the coefficients of the polynomial in the programming code, parameterized in terms of all involved variables. However, for robots as complex as the general 3UPS-PU manipulator, this is not feasible, due to these coefficients having too large expressions to obtain them, even with the help of computer algebra software.

In order to solve this problem, this paper presents a new method for obtaining all real solutions of the forward kinematics of 3UPS-PU parallel manipulators, with the advantages of elimination methods (i.e., obtaining all solutions quickly) but avoiding the too-complex-to-obtain symbolic expressions of the coefficients of the polynomial. The proposed method begins by eliminating some unknown of the system of equations (as ordinary elimination methods do), but instead of continue eliminating other unknowns until arriving at a final univariate polynomial of high degree (which has the undesired large coefficients), our method focuses on solving graphically a reduced and simpler system of equations obtained from eliminating the first unknown. Such reduced system is interpreted geometrically as a constraint curve that contains all the sought solutions. Thus, the method proposed in this paper plots that curve and efficiently searches the desired solutions along it.

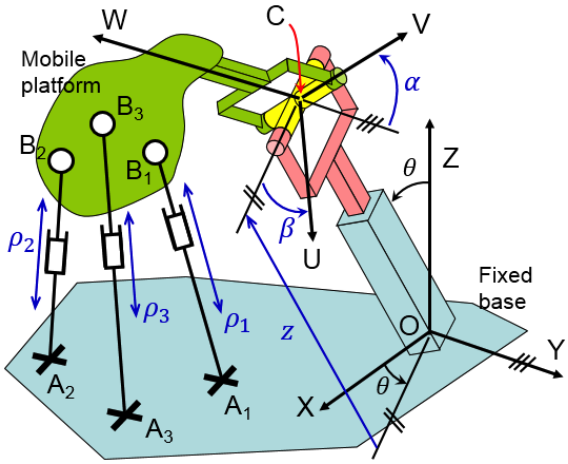

Figure 1: 3UPS-PU parallel manipulator.

The remainder of this paper is organized as follows. Section 2 presents the kinematic model of 3UPS-PU manipulators. In Section 3, this kinematic model is transformed into a set of constraint curves that contain all real solutions of the forward kinematics of this manipulator. A method to search solutions along these curves is described in Section 4. The value of the proposed method is demonstrated through an example in Section 5. Finally, Section 6 summarizes conclusions and points out future work.

2 THE 3UPS-PU MANIPULATOR

This section presents the kinematic model of the 3UPS-PU parallel manipulator, which is depicted in Figure 1. This manipulator is composed of a mobile platform connected to a fixed base through four serial kinematic chains connected in parallel. Three of such kinematic chains (denoted by A_iB_i in Figure 1) have architecture Universal-Prismatic-Spherical, with their prismatic joints being actuated, allowing one to control the length ρ_i of each leg (this chain is denoted by UPS, where underline means "actuated"). The remaining kinematic chain OC, with architecture PU, is passive, and grants three degrees of freedom to the relative motion of the mobile platform with respect to the fixed base: a translation z along a passive slider, and two rotations (α, β) about the axes of a passive Universal joint (see Figure 1). Two frames OXYZ and CUVW are attached to the fixed base and mo-

bile platform, respectively. By controlling the lengths (ρ_1, ρ_2, ρ_3) of the actuated chains, the position z and orientation (α, β) of the platform can be controlled.

The kinematic and geometric parameters of the manipulator are defined as follows. θ is the constant angle between OC and the Z axis, with OC contained in plane XZ. z is the signed position of C along the passive slider OC. α and β are the angles by which the mobile platform is rotated about the axes of the universal joint centered at C. A_i is the universal joint of the i-th actuated kinematic chain, and $\mathbf{a}_i = [a_{ix}, a_{iy}, 0]^T$ are its coordinates in frame OXYZ. B_i is the spherical joint of the i-th actuated chain, and $\mathbf{b}_i = [b_{iu}, b_{iv}, b_{iw}]^T$ are its local coordinates in frame CUVW. The length of each actuated chain is $\overline{A_iB_i} = \rho_i$.

Note that we make no simplification for the geometry of the 3UPS-PU manipulator considered in Figure 1, which is fairly general, unlike other previous papers, which mainly focus on simplified designs for which both the mobile platform and fixed base are flat (Peidró et al., 2019). In case that some of the universal joints $A_1A_2A_3$ were not contained in plane OXY as in Figure 1, one can always define plane XY passing through the centers of the universal joints $A_1A_2A_3$, then fix the origin of frame OXYZ in the point where line OC intersects this plane, and define the X and Z axes as the unit projections of line OC on plane XY and on its perpendicular, respectively.

In Figure 1, all variables except for (z, α, β) and (ρ_1, ρ_2, ρ_3) are constant design parameters, i.e., they are fixed once a concrete design is chosen for this manipulator. (ρ_1, ρ_2, ρ_3) are the controlled inputs, whereas (z, α, β) are the outputs, which can be controlled indirectly through the inputs. The relationship between inputs and outputs is derived by enforcing the condition that the distance between joints A_i and B_i must equal ρ_i $(i = 1, 2, 3)$, which yields:

$$e_1 = \left\| z \cdot \mathbf{u}_\theta + \mathbf{R}_\theta^y \mathbf{R}_\alpha^x \mathbf{R}_\beta^y \mathbf{b}_1 - \mathbf{a}_1 \right\|^2 - \rho_1^2 = 0 \qquad (1)$$

$$e_2 = \left\| z \cdot \mathbf{u}_\theta + \mathbf{R}_\theta^y \mathbf{R}_\alpha^x \mathbf{R}_\beta^y \mathbf{b}_2 - \mathbf{a}_2 \right\|^2 - \rho_2^2 = 0 \qquad (2)$$

$$e_3 = \left\| z \cdot \mathbf{u}_\theta + \mathbf{R}_\theta^y \mathbf{R}_\alpha^x \mathbf{R}_\beta^y \mathbf{b}_3 - \mathbf{a}_3 \right\|^2 - \rho_3^2 = 0 \qquad (3)$$

where $\mathbf{u}_\theta = [s_\theta, 0, c_\theta]^T$ (s_j and c_j denote the sine and cosine of subscript angle j, respectively), and $\mathbf{R}_p^q$ denotes a rotation matrix of angle p around axis q.

The problem analyzed in this paper is the forward kinematics, which consists in solving (z, α, β) from Eqs. (1)-(3) for given known values of (ρ_1, ρ_2, ρ_3). A new solution to this problem will be presented in the next sections.

3 CONSTRAINT CURVES

For given ρ_i, Eqs. (1)-(3) constitute a nonlinear system of three equations in three unknowns (z, α, β). A typical method for solving these systems consists in performing algebraic manipulations in order to eliminate equations and variables, until reducing the original system to a univariate polynomial in one of the unknowns (e.g., z), which can be solved via well-known root-finding algorithms. However, as explained in the introduction, this method has some shortcomings: the final univariate polynomial obtained can have too high a degree if the elimination steps are not done carefully, spurious solutions may appear (i.e., solutions that do not satisfy the original system), and the general symbolic expression of the coefficients of this polynomial is usually too complex or impossible to obtain even with computer algebra software due to involving many terms, which prevents parameterizing the solution in terms of general parameters. The method proposed in this paper begins by eliminating one unknown from the system (z), reducing it to a set of curves that constrain the admissible combinations of the other two unknowns (α and β). Then, these curves can be searched for real solutions of the original system, without needing to further reduce the system to a univariate polynomial, which would lead to a high-degree polynomial with excessively complicated coefficients and likely spurious solutions.

First, note that the squared norm in Eqs. (1)-(3) can be expanded as follows:

$$e_i = z^2 + a_i^2 + b_i^2 + 2z\mathbf{u}_\theta^T(\mathbf{R}\mathbf{b}_i - \mathbf{a}_i) - 2\mathbf{a}_i^T\mathbf{R}\mathbf{b}_i - \rho_i^2 \qquad (4)$$

where $a_i^2 = \|\mathbf{a}_i\|^2$, $b_i^2 = \|\mathbf{b}_i\|^2$, and $\mathbf{R} = \mathbf{R}_\theta^y \mathbf{R}_\alpha^x \mathbf{R}_\beta^y$. Considering Eq. (4), it can be checked that subtracting Eq. (3) from Eq. (1) and dividing by 2 yields:

$$k_{13} + z\mathbf{u}_\theta^T(\mathbf{R}\mathbf{b}_{13} - \mathbf{a}_{13}) - \mathrm{tr}(\mathbf{c}_{13}\mathbf{R}) = 0 \qquad (5)$$

where the following parameters are constants:

$$k_{ij} = 0.5\left[(a_i^2 - a_j^2) + (b_i^2 - b_j^2) - (\rho_i^2 - \rho_j^2)\right] \qquad (6)$$

$$\mathbf{a}_{ij} = \mathbf{a}_i - \mathbf{a}_j \qquad (7)$$

$$\mathbf{b}_{ij} = \mathbf{b}_i - \mathbf{b}_j \qquad (8)$$

$$\mathbf{c}_{ij} = \mathbf{b}_i\mathbf{a}_i^T - \mathbf{b}_j\mathbf{a}_j^T \qquad (9)$$

and where $\mathrm{tr}(\cdot)$ is the trace of a matrix. In order to obtain Eq. (5) from (4), it is necessary to make use of the linearity of the trace operator, and consider that $\mathbf{a}^T\mathbf{R}\mathbf{b} = \mathrm{tr}(\mathbf{b}\mathbf{a}^T\mathbf{R})$. Similarly, subtracting Eq. (3) from Eq. (2) and dividing by 2 yields:

$$k_{23} + z\mathbf{u}_\theta^T(\mathbf{R}\mathbf{b}_{23} - \mathbf{a}_{23}) - \mathrm{tr}(\mathbf{c}_{23}\mathbf{R}) = 0 \qquad (10)$$

Note that subtracting e_3 from e_1 and e_2 cancels the quadratic term z^2 which was present in Eq. (4), and

the two new equations (5) and (10), which replace the original quadratic Eqs. (1)-(2), are linear in z. Accordingly, for a given pair (α,β) to yield the same value of z from these two linear equations (i.e., to enforce these two equations to be consistent), the determinant of the following 2×2 matrix $\mathbf{M}$ must vanish:

$$\mathbf{M}=\begin{bmatrix} \mathbf{u}_\theta^T(\mathbf{R}\mathbf{b}_{13}-\mathbf{a}_{13}) & \mathrm{tr}(\mathbf{c}_{13}\mathbf{R})-k_{13} \\ \mathbf{u}_\theta^T(\mathbf{R}\mathbf{b}_{23}-\mathbf{a}_{23}) & \mathrm{tr}(\mathbf{c}_{23}\mathbf{R})-k_{23} \end{bmatrix} \quad (11)$$

$$\det(\mathbf{M})=0 \quad (12)$$

Recall that $\mathbf{R}=\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{R}_\beta^y$ is a function of (α,β), so the only unknowns in Eq. (12) are (α,β) (z has been eliminated from the system). Thus, Eq. (12) defines a set of *constraint curves* (possibly multiple disjoint curves) in plane (α,β), which contain the solutions of the forward kinematic problem. In particular, the points of these curves where $e_3=0$ will be the sought solutions, since the other two equations of the system are automatically satisfied for some real z due to (α,β) belonging to the curves defined by Eq. (12). The algorithm which will be described in the next Section 4 will perform an efficient search along these curves, looking for zero-crossings of e_3. However, before presenting the proposed algorithm, we will analyze in more detail the degree of such curves in plane (α,β). To that end, these curves will be intersected with straight lines for which α or β remain constant.

3.1 Intersection with Constant α

The objective of this subsection is to intersect the curves defined by Eq. (12) in plane (α,β) with a straight line defined by $\alpha=$ constant. If α is constant and known, then $\mathbf{M}$ is a function of the only unknown β, so $\mathbf{M}$ can be written as follows:

$$\mathbf{M}=\mathbf{C}_\alpha c_\beta+\mathbf{S}_\alpha s_\beta+\mathbf{Y}_\alpha \quad (13)$$

where the following newly defined matrices:

$$\mathbf{C}_\alpha=\begin{bmatrix} \mathbf{u}_\theta^T\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_c\mathbf{b}_{13} & \mathrm{tr}(\mathbf{c}_{13}\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_c) \\ \mathbf{u}_\theta^T\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_c\mathbf{b}_{23} & \mathrm{tr}(\mathbf{c}_{23}\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_c) \end{bmatrix} \quad (14)$$

$$\mathbf{S}_\alpha=\begin{bmatrix} \mathbf{u}_\theta^T\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_s\mathbf{b}_{13} & \mathrm{tr}(\mathbf{c}_{13}\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_s) \\ \mathbf{u}_\theta^T\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_s\mathbf{b}_{23} & \mathrm{tr}(\mathbf{c}_{23}\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_s) \end{bmatrix} \quad (15)$$

$$\mathbf{Y}_\alpha=\begin{bmatrix} \mathbf{u}_\theta^T(\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_Y\mathbf{b}_{13}-\mathbf{a}_{13}) & \mathrm{tr}(\mathbf{c}_{13}\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_Y)-k_{13} \\ \mathbf{u}_\theta^T(\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_Y\mathbf{b}_{23}-\mathbf{a}_{23}) & \mathrm{tr}(\mathbf{c}_{23}\mathbf{R}_\theta^y\mathbf{R}_\alpha^x\mathbf{B}_Y)-k_{23} \end{bmatrix} \quad (16)$$

are constant for constant α. In Eqs. (14)-(16), the following sparse matrices have been introduced:

$$\mathbf{B}_c=\mathrm{diag}(1,0,1),\quad \mathbf{B}_Y=\mathrm{diag}(0,1,0),\quad \mathbf{B}_s=\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad (17)$$

where $\mathrm{diag}(m_1,m_2,...,m_n)$ denotes an $n\times n$ diagonal matrix with numbers m_1, m_2, ... m_n in its diagonal. The rewriting of Eq. (11) as Eqs. (13)-(16) has been achieved by decomposing the rotation $\mathbf{R}_\beta^y$ as follows:

$$\mathbf{R}_\beta^y=\begin{bmatrix} c_\beta & 0 & s_\beta \\ 0 & 1 & 0 \\ -s_\beta & 0 & c_\beta \end{bmatrix}=\mathbf{B}_c c_\beta+\mathbf{B}_s s_\beta+\mathbf{B}_Y \quad (18)$$

Now, performing the *tangent of half-angle* substitution in Eq. (13), i.e.:

$$c_\beta=\frac{1-t_\beta^2}{1+t_\beta^2},\quad s_\beta=\frac{2t_\beta}{1+t_\beta^2},\quad \text{with } t_\beta=\tan\frac{\beta}{2} \quad (19)$$

Using (19), $\mathbf{M}$ in Eq. (13) will be written as follows:

$$\mathbf{M}=\frac{\mathbf{C}_\alpha(1-t_\beta^2)+\mathbf{S}_\alpha 2t_\beta+\mathbf{Y}_\alpha(1+t_\beta^2)}{1+t_\beta^2} \quad (20)$$

Inserting (20) into (12) yields:

$$\frac{\det\left[\mathbf{C}_\alpha(1-t_\beta^2)+\mathbf{S}_\alpha 2t_\beta+\mathbf{Y}_\alpha(1+t_\beta^2)\right]}{(1+t_\beta^2)^2}=0 \quad (21)$$

Clearing the denominator in (21) and expanding the determinant in the numerator yields the following quartic equation in t_β:

$$\omega_4 t_\beta^4+\omega_3 t_\beta^3+\omega_2 t_\beta^2+\omega_1 t_\beta+\omega_0=0, \quad (22)$$

where the coefficients ω_i are computed as follows:

$$\begin{aligned}\omega_4=&C_\alpha^{11}\cdot(C_\alpha^{22}-Y_\alpha^{22})+C_\alpha^{12}\cdot(Y_\alpha^{21}-C_\alpha^{21})\\&+C_\alpha^{21}\cdot Y_\alpha^{12}-C_\alpha^{22}\cdot Y_\alpha^{11}+Y_\alpha^{11}\cdot Y_\alpha^{22}\\&-Y_\alpha^{12}\cdot Y_\alpha^{21}\end{aligned} \quad (23)$$

$$\begin{aligned}\omega_3=&-2\cdot(C_\alpha^{11}\cdot S_\alpha^{22}-C_\alpha^{12}\cdot S_\alpha^{21}-C_\alpha^{21}\cdot S_\alpha^{12}\\&+C_\alpha^{22}\cdot S_\alpha^{11}-S_\alpha^{11}\cdot Y_\alpha^{22}+S_\alpha^{12}\cdot Y_\alpha^{21}\\&+S_\alpha^{21}\cdot Y_\alpha^{12}-S_\alpha^{22}\cdot Y_\alpha^{11})\end{aligned} \quad (24)$$

$$\begin{aligned}\omega_2=&-2\cdot(C_\alpha^{11}\cdot C_\alpha^{22}-C_\alpha^{12}\cdot C_\alpha^{21}-2\cdot S_\alpha^{11}\cdot S_\alpha^{22}\\&+2\cdot S_\alpha^{12}\cdot S_\alpha^{21}-Y_\alpha^{11}\cdot Y_\alpha^{22}+Y_\alpha^{12}\cdot Y_\alpha^{21})\end{aligned} \quad (25)$$

$$\begin{aligned}\omega_1=&2\cdot(C_\alpha^{11}\cdot S_\alpha^{22}-C_\alpha^{12}\cdot S_\alpha^{21}-C_\alpha^{21}\cdot S_\alpha^{12}\\&+C_\alpha^{22}\cdot S_\alpha^{11}+S_\alpha^{11}\cdot Y_\alpha^{22}-S_\alpha^{12}\cdot Y_\alpha^{21}\\&-S_\alpha^{21}\cdot Y_\alpha^{12}+S_\alpha^{22}\cdot Y_\alpha^{11})\end{aligned} \quad (26)$$

$$\begin{aligned}\omega_0=&C_\alpha^{11}\cdot(C_\alpha^{22}+Y_\alpha^{22})-C_\alpha^{12}\cdot(C_\alpha^{21}+Y_\alpha^{21})\\&-C_\alpha^{21}\cdot Y_\alpha^{12}+C_\alpha^{22}\cdot Y_\alpha^{11}+Y_\alpha^{11}\cdot Y_\alpha^{22}\\&-Y_\alpha^{12}\cdot Y_\alpha^{21}\end{aligned} \quad (27)$$

The coefficients C_α^{ij}, S_α^{ij}, and Y_α^{ij} denote the entries in the i-th row and j-th column of matrices $\mathbf{C}_\alpha$, $\mathbf{S}_\alpha$, and $\mathbf{Y}_\alpha$, respectively. All these coefficients are known constants for a given value of α.

The roots of (22) provide the intersection points between the constraint curves defined by Eq. (12) and a straight line defined by $\alpha = \text{constant}$. Since Eq. (22) is a quartic polynomial, it will have a maximum of four real roots, i.e., there will be up to four such intersection points. For each real root t_β^* of this polynomial, the corresponding value of β is obtained as $\beta^* = 2\arctan(t_\beta^*)$. Finally, it is worth noting that Eq. (22) can be solved analytically, since its degree is lower than five (it admits closed-form solution).

3.2 Intersection with Constant β

This case is analogous to the problem studied in §3.1. In this case, we wish to compute the intersections of the constraint curves defined by Eq. (12) in plane (α, β) with a line defined by $\beta = \text{constant}$. Following similar steps as in §3.1, we begin by decomposing the rotation matrix $\mathbf{R}_\alpha^x$ as follows:

$$\mathbf{R}_\alpha^x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha \\ 0 & s_\alpha & c_\alpha \end{bmatrix} = \mathbf{A}_c c_\alpha + \mathbf{A}_s s_\alpha + \mathbf{A}_Y \quad (28)$$

where $\mathbf{A}_c$, $\mathbf{A}_s$ and $\mathbf{A}_Y$ are the following matrices:

$$\begin{array}{l} \mathbf{A}_c = \text{diag}(0,1,1), \\ \mathbf{A}_Y = \text{diag}(1,0,0), \end{array} \quad \mathbf{A}_s = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (29)$$

Inserting (28) into (11), then $\mathbf{M}$ is rewritten as:

$$\mathbf{M} = \mathbf{C}_\beta c_\alpha + \mathbf{S}_\beta s_\alpha + \mathbf{Y}_\beta \quad (30)$$

where the matrices $\mathbf{C}_\beta$, $\mathbf{S}_\beta$ and $\mathbf{Y}_\beta$ are constant for a given β, and they have the following expressions:

$$\mathbf{C}_\beta = \begin{bmatrix} \mathbf{u}_\theta^T \mathbf{R}_\theta^y \mathbf{A}_c \mathbf{R}_\beta^y \mathbf{b}_{13} & \text{tr}(\mathbf{c}_{13} \mathbf{R}_\theta^y \mathbf{A}_c \mathbf{R}_\beta^y) \\ \mathbf{u}_\theta^T \mathbf{R}_\theta^y \mathbf{A}_c \mathbf{R}_\beta^y \mathbf{b}_{23} & \text{tr}(\mathbf{c}_{23} \mathbf{R}_\theta^y \mathbf{A}_c \mathbf{R}_\beta^y) \end{bmatrix} \quad (31)$$

$$\mathbf{S}_\beta = \begin{bmatrix} \mathbf{u}_\theta^T \mathbf{R}_\theta^y \mathbf{A}_s \mathbf{R}_\beta^y \mathbf{b}_{13} & \text{tr}(\mathbf{c}_{13} \mathbf{R}_\theta^y \mathbf{A}_s \mathbf{R}_\beta^y) \\ \mathbf{u}_\theta^T \mathbf{R}_\theta^y \mathbf{A}_s \mathbf{R}_\beta^y \mathbf{b}_{23} & \text{tr}(\mathbf{c}_{23} \mathbf{R}_\theta^y \mathbf{A}_s \mathbf{R}_\beta^y) \end{bmatrix} \quad (32)$$

$$\mathbf{Y}_\beta = \begin{bmatrix} \mathbf{u}_\theta^T (\mathbf{R}_\theta^y \mathbf{A}_Y \mathbf{R}_\beta^y \mathbf{b}_{13} - \mathbf{a}_{13}) & \text{tr}(\mathbf{c}_{13} \mathbf{R}_\theta^y \mathbf{A}_Y \mathbf{R}_\beta^y) - k_{13} \\ \mathbf{u}_\theta^T (\mathbf{R}_\theta^y \mathbf{A}_Y \mathbf{R}_\beta^y \mathbf{b}_{23} - \mathbf{a}_{23}) & \text{tr}(\mathbf{c}_{23} \mathbf{R}_\theta^y \mathbf{A}_Y \mathbf{R}_\beta^y) - k_{23} \end{bmatrix} \quad (33)$$

Next, the same steps performed between Eqs. (19)-(22) should be repeated here: the tangent of half-angle substitution $[\alpha = 2\arctan(t_\alpha)]$ is performed in Eq. (30) and the determinant of $\mathbf{M}$ is expanded, arriving at the following quartic equation:

$$\omega_4 t_\alpha^4 + \omega_3 t_\alpha^3 + \omega_2 t_\alpha^2 + \omega_1 t_\alpha + \omega_0 = 0\,, \quad (34)$$

where the coefficients ω_i depend on the (constant) entries C_β^{ij}, S_β^{ij}, and Y_β^{ij} of matrices $\mathbf{C}_\beta$, $\mathbf{S}_\beta$, and $\mathbf{Y}_\beta$, and these coefficients can be computed using Eqs. (23)-(27), by substituting every α for β in these equations.

We conclude from the analysis of Sections 3.1 and 3.2 that, when intersecting the constraint curves defined by Eq. (12) in plane (α, β) with straight lines for which α or β are constant, yields up to four intersection points, and these points can be obtained by solving analytically the quartic polynomials of Eqs. (22) or (34). This will be exploited in the method presented in the next section for finding all real solutions of the forward kinematics of 3UPS-PU manipulators.

4 PROPOSED METHOD

The constraint curve defined by Eq. (12) in plane (α, β) contains the real solutions of the forward kinematics of the 3UPS-PU manipulator. The method proposed in the present section consists in searching the desired solutions along these curves. This method consists of some steps, beginning by generating these curves, as explained in the following subsection.

4.1 Step 1: Generating the Constraint Curves

In order to search for solutions along the constraint curves implicitly defined by Eq. (12), first it is necessary to generate, "render" or "plot" these curves in plane (α, β), which will be accomplished by the sampling method described next, which was proposed in (Peidró et al., 2018). This method consists in generating a sequence by varying α between $-\pi$ rad and $+\pi$ rad using some step $\Delta\alpha$, so that for each value of α in this sequence $\{-\pi, -\pi + \Delta\alpha, -\pi + 2\Delta\alpha, ..., \pi - 2\Delta\alpha, \pi - \Delta\alpha, \pi\}$, β is solved from Eq. (22), obtaining up to four real solutions. Geometrically, this process consists in intersecting the constraint curves with a bundle of parallel vertical lines uniformly separated a distance $\Delta\alpha$ (see Figure 2a), storing all the obtained intersection points as a point cloud $Cloud_\alpha$.

This point cloud is a discrete approximation of the constraint curves, which may be used for searching the desired solutions of the forward kinematics, by checking if e_3 changes sign between two neighboring intersection points (e.g., Q and R in Figure 2a). However, in order for this strategy to work satisfactorily, the density of intersection points along the curves should be approximately uniform, which is not accomplished yet. Indeed, note that, in Figure 2a, the regions where the constraint curves have vertical tan-

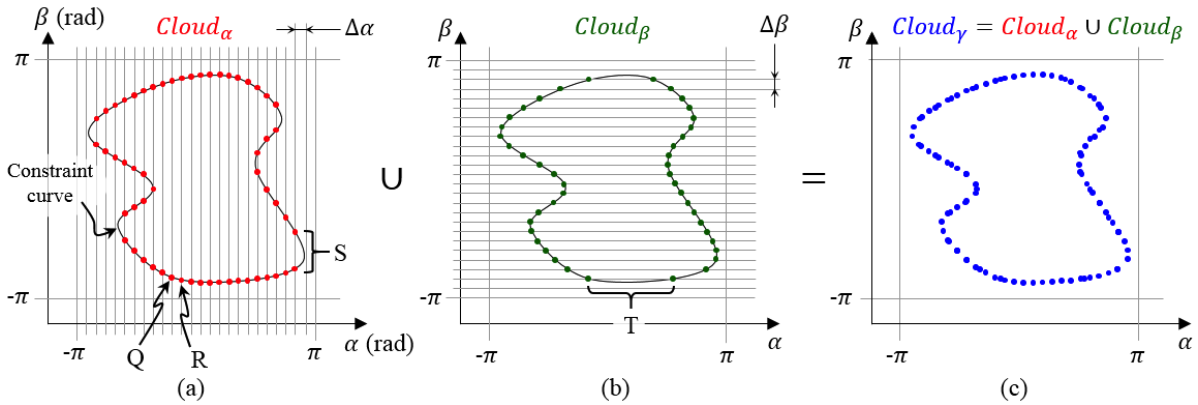

Figure 2: Discrete approximations of the constraint curves obtained by intersecting them with a bundle of vertical lines (a), horizontal lines (b), and a combination of both (c).

gent have lower density of intersection points (e.g., region S in Figure 2a). As a result of this lower density, neighboring points in these regions are too separated and the algorithm may consider that they are not truly neighbors, so it would not check if potential solutions exist between them, which may miss solutions.

In order to avoid this, another sequence is defined, this time by varying β between $-\pi$ rad and π rad with some step $\Delta\beta$, and for each value of β in this sequence $\{-\pi, -\pi+\Delta\beta, -\pi+2\Delta\beta, ..., \pi-2\Delta\beta, \pi-\Delta\beta, \pi\}$, α is solved from Eq. (34), obtaining four real solutions at most. Geometrically, this process consists in intersecting the constraint curves with a bundle of horizontal parallel lines uniformly separated a distance of $\Delta\beta$ (see Figure 2b), storing all the intersection points as another point cloud $Cloud_\beta$. Comparing Figures 2a-b, one sees that the low-density regions in Figure 2a (e.g., region S) are now better defined in Figure 2b, but at the expense of getting poorly sampled regions where the constraint curves have horizontal tangent (e.g., region T in Figure 2b).

As noted in (Simionescu, 2014; Peidró et al., 2018), the union of both point clouds yields a more accurate point cloud $Cloud_\gamma = Cloud_\alpha \cup Cloud_\beta$ that presents an *approximately* uniform density of points along the curves, since the poorly defined regions of $Cloud_\alpha$ are compensated by $Cloud_\beta$ and vice versa (see Figure 2c). This is the solution that will be adopted here for obtaining a discrete approximation of the constraint curves, which can be searched for solutions of the forward kinematic problem.

After this, a set $Cloud_\gamma = \{p_1, p_2, p_3, ...p_N\}$ of points along the constraint curves is obtained. In general, these points do not follow any useful order in $Cloud_\gamma$, since they are points of plane (α, β) that are stored in memory in the same order as they are obtained by intersecting the constraint curves with the aforementioned sequences of vertical and horizontal lines. Therefore, in order to search for solutions between these sampled points, it will be necessary to arrange these points in some data structure that allows for efficient searches of the neighbors of each point. This will be done in Step 2 of the proposed method.

Before proceeding to the next steps of the method, let us define a *unique* companion point $\tilde{p}_i = (z_i, e_{3i})$ associated to each $p_i = (\alpha_i, \beta_i)$. $\tilde{p}_i$ is calculated from p_i as follows: first, z_i is obtained by substituting (α_i, β_i) into Eq. (5) or (10) and solving for z. Any of these linear equations is equally valid: since (α_i, β_i) belongs to the constraint curve (they satisfy Eq. 12), it is guaranteed that these two equations provide the same solution for z. Then, α_i, β_i, and z_i can be substituted into Eq. (3) to obtain a unique value of e_{3i}.

4.2 Step 2: Sorting the Point Cloud

The second step of the proposed method consists in organizing the point cloud $Cloud_\gamma$ using k-d (k-dimensional) trees, which are data structures that sort points in a k-dimensional space (in this paper, k=2). A k-d tree $\mathcal{T}$ is obtained from a point cloud by recursively dividing it at the point whose coordinate is the median (and the coordinate used for doing this division is switched cyclically after each division). For example: Figure 3a shows a set of seven points 1...7 in plane (α, β). Point 5 is the median along coordinate α, so it divides the set into two branches $br_l = \{1, 4, 6\}$ and $br_r = \{2, 3, 7\}$. This division is performed again inside these two branches, but using now points 6 and 2, since these are the medians along coordinate β in each branch. This generates four new branches: $br_{lu} = \{4\}$, $br_{ld} = \{1\}$, $br_{ru} = \{7\}$ and $br_{rd} = \{3\}$. If we had more points, this process would continue recursively, alternating cyclically the coordinate along which the median divides each branch (α, β, α, β...). The resulting k-d tree is shown in Figure 3b.

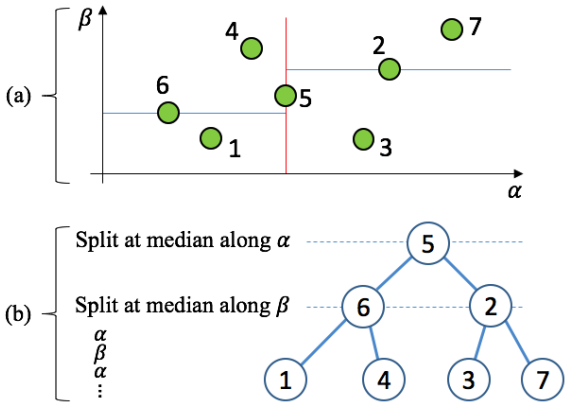

Figure 3: A set of points (a), and their k-d tree (b).

Building a k-d tree is useful for efficiently retrieving data from a point cloud, e.g., obtaining all points whose coordinates lie between specific limits (this is what will be done in this paper). Since each layer divides the previous one into two sets depending if their coordinate is below or above the median, retrieving all points whose coordinates lie within desired limits is done faster than by checking *all* of them, since a k-d tree search discards half of the data at each layer. See (Press et al., 2007) for methods to build k-d trees.

4.3 Step 3: Searching Solutions

After building a k-d tree data structure $\mathcal{T}$ to sort the point cloud $Cloud_\gamma$, Algorithm 1 is executed to search solutions along the curves. First, a set $\mathcal{S}$ is defined as the empty set (line 1). After the algorithm finishes, this set $\mathcal{S}$ will contain the sought solutions. Then, for each point $p_i = (\alpha_i, \beta_i) \in Cloud_\gamma$ (line 2), a set $\mathcal{N}(p_i, r_\alpha, r_\beta)$ of neighbors of p_i is retrieved from the k-d tree $\mathcal{T}$ (line 3). $\mathcal{N}(p_i, r_\alpha, r_\beta)$ is defined as follows:

$$\mathcal{N}(p_i, r_\alpha, r_\beta) = Cloud_\gamma \cap \mathcal{B}(p_i, r_\alpha, r_\beta) \setminus \{p_i\} \quad (35)$$

where:

$$\mathcal{B}(p_i = (\alpha_i, \beta_i), r_\alpha, r_\beta) = \left\{ (\alpha_j, \beta_j) : \max\left(\frac{|\alpha_j - \alpha_i|}{r_\alpha}, \frac{|\beta_j - \beta_i|}{r_\beta} \right) \leq 1 \right\} \quad (36)$$

Note that $\mathcal{N}$ is composed of all points p_j of $Cloud_\gamma$ (excluding p_i) which fall inside a box $\mathcal{B}$ centered at p_i and with width $2r_\alpha$ and height $2r_\beta$, with r_α and r_β being small radii (see Figure 4a). This retrieval operation is very fast because p_j have been arranged into a k-d tree $\mathcal{T}$ previously (step 2), which allows one to search very efficiently points whose coordinates lie in specified intervals, like those imposed by box $\mathcal{B}$.

When retrieving the neighbors p_j of p_i, one should be careful with the fact that all these points

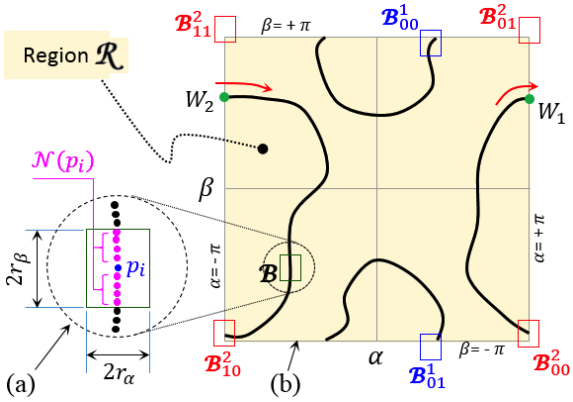

Figure 4: (a) Box $\mathcal{B}$ containing the neighbors of p_i. (b) Constraint curves wrapped at $\alpha, \beta = \pm\pi$, with wrapped boxes $\mathcal{B}_{uv}$. Boxes are exaggerated here, they usually are smaller.

Algorithm 1: Searching solutions $\mathcal{S}$ along the constraint curves defined by Eq. (12).

1: $\mathcal{S} = \emptyset$
2: **for all** $p_i = (\alpha_i, \beta_i) \in Cloud_\gamma$ **do**
3: Retrieve $\mathcal{N}(p_i, r_\alpha, r_\beta)$ from $\mathcal{T}$
4: **for all** $p_j = (\alpha_j, \beta_j) \in \mathcal{N}(p_i, r_\alpha, r_\beta)$ **do**
5: **if** $e_{3i} \cdot e_{3j} \leq 0$ **then**
6: $\alpha^* = (e_{3i}\alpha_j - e_{3j}\alpha_i)/(e_{3i} - e_{3j})$
7: $\beta^* = (e_{3i}\beta_j - e_{3j}\beta_i)/(e_{3i} - e_{3j})$
8: $z^* = (e_{3i}z_j - e_{3j}z_i)/(e_{3i} - e_{3j})$
9: $\mathbf{s}^* = [\alpha^*, \beta^*, z^*]^T$
10: **if** $\mathbf{s}^* \notin \mathcal{B}(\mathbf{s}, r_\alpha, r_\beta)\ \forall \mathbf{s} \in \mathcal{S}$ **then**
11: **do**
12: $\Delta\mathbf{s}^* = -\mathbf{J}(\mathbf{s}^*)^{-1}\mathbf{F}(\mathbf{s}^*)$
13: $\mathbf{s}^* \leftarrow \mathbf{s}^* + \Delta\mathbf{s}^*$
14: **while** $\|\Delta\mathbf{s}^*\| > \varepsilon$
15: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{s}^*\}$

lie on the constraint curves in plane (α, β), which is a plane of angles. Since α and β appear in Eq. (12) inside sines and cosines, the solution set of Eq. (12) in plane (α, β) is periodic every 2π rad along both axes. This is why α and β were swept between $-\pi$ and $+\pi$ when generating the curves as explained in Section 4.1: all information is confined to this region $\mathcal{R} = [-\pi, \pi] \times [-\pi, \pi]$ of the plane. As a result of this, the curves defined by Eq. (12) wrap at the lines $\alpha = \pm\pi$ and $\beta = \pm\pi$, as illustrated in Figure 4b: e.g., a point W_1 marching along these curves and crossing the vertical line $\alpha = \pi$ in positive direction as indicated in Figure 4b would reappear as if coming from $\alpha = -\pi$, as indicated by point W_2 in the same figure.

Due to this wrapping, when retrieving the neighbors of a point p_i that is close to the borders of region $\mathcal{R}$, one should not omit the *wrapped* neighbors of such p_i. For example: the neighbors to the right

of W_2 in Figure 4b are also neighbors of W_1 because, due to wrapping, W_1 and W_2 are identified as the same point. However, k-d tree methods for sorting and retrieving points do not work with wrapped angular coordinates, and would fail to identify the neighbors to the right of W_2 as neighbors of W_1 also.

To avoid this wrapping of angular coordinates, Peidró et al. (2018) proposed using the sines s_j and cosines c_j of angles as the variables of the problem, instead of working with the angles j directly, imposing the additional constraint: $s_j^2 + c_j^2 = 1$. This indeed avoids the problem of wrapping and permits using k-d trees to sort and search points, since the sines and cosines do not wrap as their angles, but it also has the drawback of doubling the dimension of the problem, which increases computational time: each original angular variable j is substituted by two variables (the sine and cosine of j). In this paper, we propose an alternative method for searching points using k-d trees when these points have angular coordinates that wrap along the axes of the ambient space, without having to double the dimension of such a space by unfolding each angle into its sine and cosine.

The alternative method is as follows: in line 3 of Algorithm 1, when using k-d trees to retrieve the set $\mathcal{N}$ of neighboring points inside a box $\mathcal{B}$ centered at p_i, one must "replicate" this box if it intersects any of the wrapping lines $\alpha = \pm\pi$ or $\beta = \pm\pi$, defining "replica boxes" whose centers are translated a distance of 2π along the wrapped axes. More formally, let us define an auxiliary variable σ_α as follows:

$$\sigma_\alpha = \begin{cases} +1 & \text{if } \mathcal{B} \text{ intersects the line } \alpha = +\pi \\ -1 & \text{if } \mathcal{B} \text{ intersects the line } \alpha = -\pi \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

In (37), it is assumed that $\mathcal{B}$ will not intersect simultaneously both lines $\alpha = \pi$ and $\alpha = -\pi$ since this box should be small, as in Figure 4b. Let us define an analogous variable σ_β, by substituting every α in (37) for β. After defining σ_α and σ_β, one only has to change $\mathcal{B}$ in Eq. (35) for $\overline{\mathcal{B}}$, which is defined as follows:

$$\overline{\mathcal{B}}\left(p_i, r_\alpha, r_\beta\right) = \bigcup_{u=0}^{|\sigma_\alpha|} \bigcup_{v=0}^{|\sigma_\beta|} \mathcal{B}_{uv}(p_i, r_\alpha, r_\beta, \sigma_\alpha, \sigma_\beta) \tag{38}$$

where:

$$\mathcal{B}_{uv}(p_i, r_\alpha, r_\beta, \sigma_\alpha, \sigma_\beta) = \mathcal{B}\left(p_i - 2\pi(u\sigma_\alpha, v\sigma_\beta), r_\alpha, r_\beta\right) \tag{39}$$

and where $\mathcal{B}$ is defined as in Eq. (36). Equation (38) effectively defines the union of $2^{|\sigma_\alpha|+|\sigma_\beta|}$ wrapped copies or replicas of box $\mathcal{B}$, with each copy being translated a distance of 2π along each axis in order to cover all the wrapped regions intersected with the original box $\mathcal{B}_{00}$. For example, Figure 4b shows two boxes $\mathcal{B}_{00}^1$ and $\mathcal{B}_{00}^2$, which intersect the wrapping lines $\alpha = \pm\pi$ or $\beta = \pm\pi$, and their wrapped replicas $\{\mathcal{B}_{00}^1, \mathcal{B}_{01}^1\}$ and $\{\mathcal{B}_{00}^2, \mathcal{B}_{01}^2, \mathcal{B}_{10}^2, \mathcal{B}_{11}^2\}$, whose centers are translated as indicated by Eq. (39).

Then, in order to retrieve the wrapped neighbors of p_i using k-d trees, one has to perform a search for all points in k-d tree $\mathcal{T}$ whose coordinates fall in the intervals defined by each box $\mathcal{B}_{uv}$ separately.

Continuing with Algorithm 1, after retrieving all neighbors p_j of current point p_i (including possible wrapped neighbors), then the sign of e_{3i} for the current point p_i is compared to the sign of e_{3j} for each neighbor. If e_3 changes sign between p_i and any p_j of its neighbors, a solution exists between them, and this solution is approximated by a linear interpolation between p_i and p_j, by enforcing $e_3 = 0$. This linear interpolation is performed between lines 6-9 of Algorithm 1. Note that not only the coordinates (α, β) are interpolated, but also z (line 8). Recall that the values of z and e_3 are stored in each companion point $\tilde{p}_i$ of p_i computed during the first step of the method. Note also that, in case of wrapping, when interpolating α and β in lines 6-7, the values of (α_i, β_i) must be those of point p_i translated by the displacement of Eq. (39).

After this linear interpolation, an approximate solution $\mathbf{s}^*$ is obtained (line 9 of Algorithm 1). This solution is compared to all other solutions stored in $\mathcal{S}$ so far (line 10), and if this solution is not similar to the previously stored solutions (i.e., if it does not belong to small boxes centered at other solutions, line 10), then it is regarded as a new solution and it is stored in $\mathcal{S}$ (line 15). Before storing a new solution, however, it is refined through Newton-Raphson method for nonlinear systems, which converges very quickly (in about 2-3 iterations) to a very accurate solution, since the starting approximate solution obtained by linear interpolation is already very close to the exact one. Newton-Raphson method is applied between lines 11-14, in which $\Delta\mathbf{s}^*$ is the improvement of the solution in each iteration, ε is a small threshold below which further improvements are considered negligible (stop criterion), and $\mathbf{F}$ and $\mathbf{J}$ are the constraint vector function and its Jacobian matrix, respectively, which are defined as follows: $\mathbf{F} = [e_1, e_2, e_3]^T$ and

$$\mathbf{J} = \begin{bmatrix} \partial e_1/\partial\alpha & \partial e_1/\partial\beta & \partial e_1/\partial z \\ \partial e_2/\partial\alpha & \partial e_2/\partial\beta & \partial e_2/\partial z \\ \partial e_3/\partial\alpha & \partial e_3/\partial\beta & \partial e_3/\partial z \end{bmatrix} \tag{40}$$

where e_i are functions of $\mathbf{s} = [\alpha, \beta, z]^T$, defined in Eqs. (1)-(3). Since $\mathbf{s}^*$ will change little during Newton iterations, matrix $\mathbf{J}^{-1}$ may be computed only once before refining each solution $\mathbf{s}^*$ (right before line 11) and kept constant in line 12, which saves computations without impeding fast convergence.

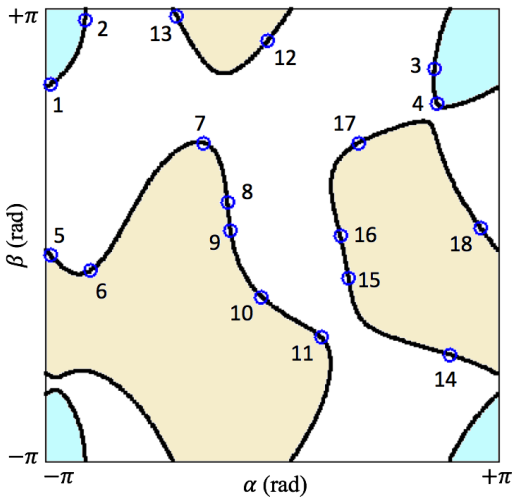

Figure 5: The two closed constraint curves for the example of Section 5. The region enclosed by each curve has been colored to better distinguish these curves under wrapping.

5 EXAMPLE AND DISCUSSION

Next, the efficiency and validity of the proposed method will be demonstrated through an example with the following parameters: $\mathbf{a_1} = [0.7, 2.45, 0]^T$, $\mathbf{a_2} = [2.676, -1.379, 0]^T$, $\mathbf{a_3} = [-2.161, 2.627, 0]^T$, $\mathbf{b_1} = [-2.255, 1.099, 2.728]^T$, $\mathbf{b_2} = [0.675, -2.347, 0.532]^T$, $\mathbf{b_3} = [-1.935, -0.966, -1.953]^T$, $\theta = 0$, $\rho_1 = 5$, $\rho_2 = 4.5$, and $\rho_3 = 4.631$ (assume arbitrary units of length). Our method will be run configured as follows: $\Delta\alpha = \Delta\beta = 0.0315738$ rad, $r_\alpha = r_\beta = 0.0347312$ rad, $\varepsilon = 0.000001$ (threshold to stop Newton refinement).

For these parameters, the constraint curves obtained by the proposed method are shown in Figure 5 in thick black lines. As this figure shows, this example yields 18 real solutions, which are identified by numbers 1-18 and small blue circles along these curves. Note that, considering the wrapping of these curves along the lines $\alpha = \pm\pi$ and $\beta = \pm\pi$, there are two closed constraint curves. The sampling method proposed in Section 4.1 is robust to the frequent case in which these curves have multiple connected components, which may pose troubles to alternative methods based on continuation for tracking and rendering these curves. In that case, continuation methods should guarantee that a continuation starts from one seed point belonging to each connected component, in order to track all components (Gomes, 2014).

The values of (α, β, z) for these 18 solutions are shown in Table 1. To check the correctness of these solutions, these were back-substituted into Eqs. (1)-(3) to solve ρ_i from them, obtaining the original values ($\rho_1 = 5$, $\rho_2 = 4.5$, and $\rho_3 = 4.631$) with errors smaller than 10^{-9} in all cases. Also, to check if the proposed method obtained all real solutions (without missing solutions), these were also obtained by other reliable methods for obtaining all solutions, which are available as software packages: *CUIK* (Porta et al., 2014) and *Bertini* (Bates et al., 2013), which use branch-and-prune and homotopy methods, respectively. Our method was coded in Java programming language, and all three methods (ours, and the CUIK and Bertini methods) were tested on a single-core 2GB RAM Ubuntu 32bit virtual machine with CUIK preinstalled on it, running on a Mac Pro with a 3 GHz 8-Core Intel Xeon E5 processor and 16 GB RAM. All three methods obtained 18 real solutions, and the solutions obtained by our method coincide with those of the other two methods up to the eighth decimal position (CUIK and Bertini work with the cosine and sine of angles intead of the angles themselves, so this comparison was made after computing the sine and cosine of α and β for all 18 solutions displayed in Table 1).

Comparing times, our method took only 0.01 seconds to generate the curves and obtain all 18 solutions (steps 1 to 3), whereas CUIK and Bertini took 1.11 and 0.89 seconds to obtain these solutions, respectively (CUIK and Bertini were run using their default parameters). These times are averages of 10 runs for each method, with standard deviations of 0.002, 0.02 and 0.08 seconds for our method, CUIK, and Bertini, respectively. The times measured for the CUIK and Bertini methods were extracted from the output files generated by these packages, which are identified as "User time in process" in CUIK, and "Track Time" in Bertini. According to this comparison, our method is roughly 100 times faster than other methods.

Nevertheless, CUIK and Bertini have other advantages over our proposed method. For example, Bertini can obtain also all nonreal solutions, which are not physically possible but still are useful for learning how they become real when approaching singularities (Peidró et al., 2019). In particular, the example considered in this section has 10 nonreal solutions besides the 18 real ones obtained by the proposed method, and Bertini obtains all 28 solutions (real and nonreal). Regarding CUIK, it can handle degenerate cases which are otherwise elusive or very difficult to solve, e.g., when a constraint curve degenerates into a point solution, which would be overlooked by the sampling method described in Section 4.1. Finally, both CUIK and Bertini work with any parallel manipulator, whereas our method is *ad-hoc* for 3UPS-PU manipulators. However, we believe that this method can be easily generalized and extended to more complex manipulators, which is the topic of future work.

Table 1: 18 real solutions for the example of Section 5.

# Solution	α (rad)	β (rad)	z
1	-3.074015668	2.096303267	-1.560581389
2	-2.598732611	2.977271815	-1.218233476
3	2.259863227	2.298271789	1.836517445
4	2.296596123	1.825026909	-0.312209976
5	-3.076668574	-0.285479858	1.695349818
6	-2.521795906	-0.498199503	0.138552682
7	-0.944416244	1.279997012	-5.636730120
8	-0.618751656	0.447473831	2.571815016
9	-0.573215460	0.056916560	2.345009820
10	-0.139016511	-0.862859797	2.372701275
11	0.684167157	-1.425432943	4.033720688
12	-0.056524769	2.706168583	-2.505056890
13	-1.325219478	3.042199401	0.924688594
14	2.483381960	-1.670214061	0.751169173
15	1.065465274	-0.595531432	-2.109833757
16	0.950424577	-0.014348808	-2.409395862
17	1.203800574	1.274843283	-1.554851753
18	2.911141509	0.085737211	2.937707838

6 CONCLUSIONS

In this paper, we have presented a new method for solving the forward kinematics of 3UPS-PU parallel manipulators. The proposed method eliminates one unknown, obtaining an equation that defines a constraint curve in the plane of the remaining unknowns. This constraint curve is densely sampled and then efficiently searched to find all real solutions of the forward kinematics. This method makes use of k-d trees and requires solving quartic polynomials at most. Furthermore, explicit symbolic expressions of the coefficients of these polynomials have been derived, which are parameterized in terms of all kinematic parameters. This allows us to change the value of any kinematic parameter and instantaneously recompute and update all solutions. This is not feasible in general for elimination methods, which yield high-degree polynomials that involve too large and complicated expressions to be obtained symbolically.

The proposed method is about 100 times faster than other methods, but has also a much more limited scope, since it works only for 3UPS-PU robots. However, we are currently working on the extension of this method to broader families of manipulators with more degrees of freedom, succeeding to find "constraint manifolds" (analogous to the constraint curves here analyzed) that can be efficiently sampled and searched for all real solutions of the forward kinematics. Extension of this method to the complex domain will also be explored, to find also nonreal solutions.

ACKNOWLEDGEMENTS

This research was funded by the Spanish government through the project DPI2016-78361-R (AEI/FEDER, UE): "Creación de mapas mediante métodos de apariencia visual para la navegación de robots", and by the Generalitat Valenciana through the project AICO/2019/031: "Creación de modelos jerárquicos y localización robusta de robots móviles en entornos sociales".

REFERENCES

Bates, D. J., Hauenstein, J. D., Sommese, A. J., and Wampler, C. W. (2013). *Numerically Solving Polynomial Systems with Bertini*. SIAM.

Caccavale, F., Siciliano, B., and Villani, L. (2003). The tricept robot: dynamics and impedance control. *IEEE/ASME Transactions on Mechatronics*, 8(2):263–268.

Fattah, A. and Kasaei, G. (2000). Kinematics and dynamics of a parallel manipulator with a new architecture. *Robotica*, 18(5):535–543.

Gomes, A. J. (2014). A continuation algorithm for planar implicit curves with singularities. *Computers & Graphics*, 38:365 – 373.

Hosseini, M. A. and Daniali, H.-R. M. (2011). Kinematic analysis of tricept parallel manipulator. *IIUM Engineering Journal*, 12(5):7–16.

Hu, B. (2014). Complete kinematics of a serial–parallel manipulator formed by two tricept parallel manip-

ulators connected in serials. *Nonlinear Dynamics*, 78(4):2685–2698.

Innocenti, C. and Wenger, P. (2006). Position analysis of the RRP-3(SS) multi-loop spatial structure. *J. Mech. Design*, 128(1):272–278.

Joshi, S. and Tsai, L.-W. (2003). The kinematics of a class of 3-dof, 4-legged parallel manipulators. *J. Mech. Design*, 125(1):52–60.

Kolláth, L., Kureková, E., Ploskuňáková, L., and Beniak, J. (2009). Non-conventional production machines. *Scientific Proceedings. SjF, STU Bratislava*, pages 69–75.

Liu, C. H. and Hsu, F.-K. (2007). Direct singular positions of the parallel manipulator tricept. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 221(1):109–117.

Lu, Y., Hu, B., and Liu, P. (2007). Kinematics and dynamics analyses of a parallel manipulator with three active legs and one passive leg by a virtual serial mechanism. *Multibody System Dynamics*, 17(4):229–241.

Neumann, K. (1988). Robot, US4732525A.

Peidró, A., Gil, A., Marín, J. M., and Reinoso, Ó. (2016). A web-based tool to analyze the kinematics and singularities of parallel robots. *Journal of Intelligent & Robotic Systems*, 81(1):145–163.

Peidró, A., Marín, J. M., Reinoso, Ó., Payá, L., and Gil, A. (2019). Parallelisms between planar and spatial tricept-like parallel robots. In Arakelian, V. and Wenger, P., editors, *ROMANSY 22 – Robot Design, Dynamics and Control*, pages 155–162, Cham. Springer International Publishing.

Peidró, A., Reinoso, O., Gil, A., Marín, J. M., and Payá, L. (2018). A method based on the vanishing of self-motion manifolds to determine the collision-free workspace of redundant robots. *Mechanism and Machine Theory*, 128:84 – 109.

Pendar, H., Roozbehani, H., Sadeghian, H., and Zohoor, H. (2008). Singularity analysis of a 3dof parallel manipulator using infinite constraint plane method. *Journal of Intelligent and Robotic Systems*, 53(1):21–34.

Porta, J. M., Ros, L., Bohigas, O., Manubens, M., Rosales, C., and Jaillet, L. (2014). The cuik suite: Analyzing the motion closed-chain multibody systems. *IEEE Robotics Automation Magazine*, 21(3):105–114.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing (3rd Edition)*. Cambridge University Press.

Siciliano, B. (1999). The tricept robot: Inverse kinematics, manipulability analysis and closed-loop direct kinematics algorithm. *Robotica*, 17(4):437–445.

Simionescu, P. A. (2014). *Computer-aided graphing and simulation tools for AutoCAD users*. CRC Press.