A Fast Branch-and-Prune Algorithm for the Position Analysis of Spherical Mechanisms

Arya Shabani, Soheil Sarabandi, Josep M. Porta, and Federico Thomas

Institut de Robotica i Informatica industrial, CSIC-UPC,
Llorens Artigas 4-6, 080028 Barcelona, Spain,
{ashabani, ssarabandi, porta, fthomas}@iri.upc.edu.

Abstract. Different branch-and-prune schemes can be found in the literature for numerically solving the position analysis of spherical mechanisms. For the prune operation, they all rely on the propagation of motion intervals. They differ in the way the problem is algebraically formulated. This paper exploits the fact that spherical kinematic loop equations can be formulated as sets of 3 multi-affine polynomials. Multi-affinity has an important impact on how the propagation of motion intervals can be performed because a multi-affine polynomial is uniquely determined by its values at the vertices of a closed hyperbox defined in its domain.

Keywords: multi-affine polynomials, parallel spherical robots, forward kinematics, position analysis, branch-and-prune algorithms

1 Introduction

Solving systems of geometric constraints is a fundamental topic in Computational Kinematics, Computer Graphics, and Computational Chemistry. In Computational Kinematics, it is commonly referenced to as *position analysis* or *displacement analysis* which includes the inverse (forward) kinematics of serial (parallel) robots.

The solution to position analysis problems is usually preferred to be in closed-form. That is, the solution is given as the roots of a closure polynomial of the lowest possible degree in a single variable (sometime called *characteristic equation* of the mechanism). The most straightforward approach to obtain these polynomials consists in applying a series of variable eliminations to a set of independent kinematic loop equations. Nevertheless, to make this process practical, it is usually needed to simplify the formulation by exploiting the particularities of each problem instance. As a result, obtaining the 16-degree closure polynomial of a general 6R robot [1] is quite different from obtaining the 40-degree closure polynomial of a general Gough-Stewart parallel platform [2]. Automating the process to obtain these polynomials for any case in a reasonable amount of time seems impossible at the moment. Nevertheless, if instead of a set of kinematic loop equations, a distance-based formulation is used, a higher lever of generality and

uniformity has recently been obtained thus opening an avenue for an automatic analysis without ad hoc considerations (see [3] for a step in this direction).

Alternatively to obtain a closure polynomial, it is possible to directly compute the solutions of the system of loop equations by using numerical approaches such as those based on numerical continuation or interval methods (see, for example, [4] and [5], respectively). These numerical approaches are the only alternative to large problems like those arising in mechanism synthesis.

The closed-form position analysis of spherical mechanisms (mechanisms constructed by connecting links with hinged joints such that the axes of each hinge passes through the same point) has received the attention of different authors. For example, the methods presented by Wampler in [6] and by Bai *et al.* in [7] can be applied to obtain closure polynomials for spherical mechanisms with up to three and to two independent kinematic loops, respectively. Numerical approaches has also been tested. Bombín *et al.* reformulated the kinematic loop equations in terms of Bernstein polynomials [8, 9]. This permited to formulate the problem in terms of control points, which could be recalculated for arbitrary ranges using a simple subdivision procedure. Celaya presented an ingenious method to exactly propagate intervals of motion in a spherical kinematic loop [10]. The iterative application of this method to all the kinematic loops describing the problem at hand permitted to converge to all the solutions.

The work of Bombín introduced an important concept in position analysis: that of control points. Specifying curves and surfaces in terms of control points is one of the major techniques used in geometric design [11]. The idea is simple: the values of a polynomial function in a given domain (usually an othotope) lie inside the region determined by the convex hull of a set of control points whose number depend on the degree of the polynomial. Obtaining these control points requires expressing the polynomials in Bernstein base. Nevertheless, if the polynomials are multi-linear or even multi-affine, this operation is trivial because the control points are obtained by simply evaluating the polynomial at the corners of the orthotope defining the domain.

Although any polynomial can be easily transformed into a system of multi-affine polynomials by introducing new variables and more equations (as it is done, for example, in [12]), increasing the number of equations is not, in general, a good idea. Fortunately, we will show in this paper how the proper formulation of kinematic loop equations in spherical mechanisms directly leads to a minimal set of multi-affine polynomials. As a consequence, a simple and yet very effective branch-and-prune algorithm is derived to obtain the roots of these systems of equations.

This paper is organized as follow. The next section describes how to obtain an multi-affine formulation for the kinematic loop equations in spherical mechanisms using quaternions. Section 3 deals with the numerical problem introduced by the Weierstrass substitution, implicit in the proposed formulation, and how to avoid it. The branch-and-prune algorithm is described in Section 4. The behavior of this algorithm is analyzed in Section 5 for three problems of increasing complexity. Finally, Section 6 summarizes the points and gives some prospects for

feature research such as the generalization of the presented approach to spatial mechanisms.

2 Kinematic loop equations in spherical mechanisms

An spherical kinematic loop equation can always be formulated in terms of products of unit quaternions

$$q_1 \cdot q_2 \ldots q_r = 1, \tag{1}$$

where each quaternion has the form

$$q_i = \frac{1}{\sqrt{t_i^2+1}} \left[1 + t_i(p_{ix}i + p_{iy}j + p_{iz}k)\right], \tag{2}$$

where $\mathbf{p}_i = (p_{ix}\, p_{iy}\, p_{iz})$ is a unit vector representing the rotation axis and $t_i = \tan\left(\frac{\phi_i}{2}\right)$, ϕ_i being the rotated angle about it. The angle ϕ_i can be constant or variable.

The expansion of equation (1) leads, after clearing denominators, to the following four scalar equations

$$\begin{aligned} f_0(t_1, \ldots, t_n) &= \sqrt{(t_1^2+1)\cdots(t_n^2+1)}, \\ f_j(t_1, \ldots, t_n) &= 0, \quad j = 1, 2, 3, \end{aligned}$$

where in the parentheses only appear the variable t_i, $i = 1, \ldots, n$ which have been renumbered, that is, $n \leq r$. These four equations are not independent because they correspond to the components of a unit quaternion. While f_j, $j = 1, 2, 3$, are affine polynomials in each variable, f_0, after clearing the radical, leads to a quadratic polynomial. For obvious simplicity reasons, in what follows, we will take f_j, $j = 1, 2, 3$, as the set of three independent equations. Therefore, if our problem can be described in terms of n independent loop equations, we will have a set of $3n$ multi-affine polynomial equations.

A word of caution should be added here. Sometimes, in an abuse of language, a system of equations like the one given by f_j, $j = 1, 2, 3$, is called multi-linear. Strictly speaking, it is multi-affine. Indeed, f_j can be expressed as a function of t_i as $f_j = a_j t_i + b_j$ which is not linear in t_i, but affine.

Multi-affine polynomials have two interesting properties which are very useful for our purposes. If we have an affine polynomial function whose domain is an orthotope, then

- the value of the function in any point inside the domain defined by the orthotope is inside the convex hull of the points resulting from evaluating the function at the corners of the orthotope [13].
- the polynomial function is completely determined by the values of the function at the corners of the orthotope. The value of the function in any other point can be obtained by a simple interpolation procedure [14].

These two properties are exploited in Section 4.

3 Numerical issues derived from the Weierstrass substitution

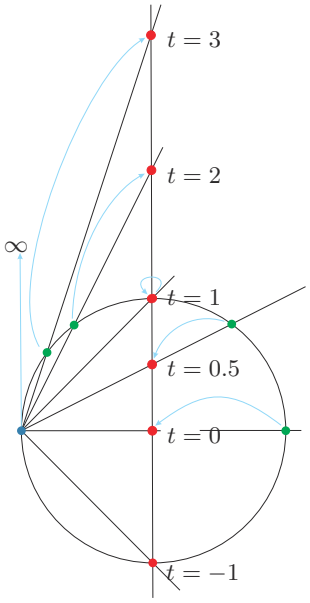

Fig. 1. The Weierstrass substitution maps the interval $[-\pi, \pi]$ onto the interval $[-1, 1]$. Numerical problems arise as we approach π because it is mapped onto ∞.

In above formulation, the variable t_i is related to the angle variable ϕ_i thought the relationship $t_i = \tan\left(\frac{\phi_i}{2}\right)$, that is, the tangent half-angle substitution (also known as the Weierstrass substitution). From the geometric point of view, this substitution can be seen as the stereographic projection of the unit circle, from $x = -1$, to the line $y = 0$ (see Fig. 1). Thus, if $\phi_i = \pi$, t_i is infinity. Then, if a mechanism has a solution near π, numerical problems occur. Moreover, if we want to apply a branch-and-prune method to obtain the solutions of our system of equations, it is not a good decision to start with a domain ranging from $-\infty$ to $+\infty$ in all variables. One possible solution to this situation is to split the problem in two: one for $\phi_i \in [-\pi/2, \pi/2]$, and another one for $\phi_i' \in [-\pi/2, \pi/2]$, where $\phi_i' = \phi_i + \pi$. The second problem just consists in shifting the origin of the rotation about $\mathbf{p}_i$ π radians. This can be accomplished by including a constant rotation prior to the variable rotation about the same axis. For example, if in the loop equation $q_1 \cdot q_2 \ldots q_r = 1$, we want to shift the origin of the rotation about $\mathbf{p}_i$, $1 \leq i \leq r$, π radians, we just have to substitute q_i by $(ip_x + jp_y + kp_z)q_i$. This operation has to be done for all variables. In other words, if our problem has r variables, we will decompose it into 2^r equivalent problems in the domain $[-1, 1]_1 \times [-1, 1]_2 \times \ldots [-1, 1]_r$. Having this domain for all problems has an important extra advantage: if the values of the functions are obtained by interpolation from the values of the function at the corners of the initial domain, this is simplified because all coordinates are -1 or 1.

4 Solving multi-affine polynomial systems

Let us define the equation

$$\mathbf{F}(\mathbf{x}) = 0, \tag{3}$$

where $\mathbf{F} = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$, and each function f_i is a multi-affine polynomial in the unknowns $x_1, \ldots, x_n$.

Next, we describe an algorithm able to isolate all solutions of (3) that lie in the orthotope $\mathcal{B} \in \mathbb{R}^n$ defined as the Cartesian product $\mathcal{B} = [x_1^l, x_1^u] \times \cdots \times [x_n^l, x_n^u]$.

4.1 A branch-and-prune scheme

Generally speaking, the algorithm isolates the solutions by iterating two operations, *box reduction* and *box bisection*, using the following branch-and-prune scheme. Using box reduction, portions of $\mathcal{B}$ containing no solution are cut off by narrowing some of its defining intervals. This process is iterated until either (1) the box is reduced to an empty set, in which case it contains no solution, or (2) the box is "sufficiently" small, in which case it is considered a solution box, or (3) the box cannot be "significantly" reduced, in which case it is split into two sub-boxes via box bisection. If the latter occurs, the whole process is repeated for the newly created sub-boxes, and for the sub-boxes recursively created thereafter, until one ends up with a collection of boxes whose size is under a specified size threshold, as explained in detail below.

If there is only a finite number of solution points in $\mathcal{B}$, this algorithm returns a collection of small boxes containing them all. If, contrarily, the solution space is an algebraic variety of dimension one or higher, the algorithm returns a collection of small boxes discretizing the portions of this variety contained in $\mathcal{B}$. In all cases, the algorithm is *complete*, in the sense that every solution point will be contained in one of the returned boxes.

Let us now follow the box reduction and bisection operations in detail, to later see how they fit into the algorithm.

4.2 Box reduction and box bisection

The box reduction operation is based on the following lemma, which is a direct consequence of Theorem 1.1 in [13]:

> The point $(\mathbf{x}, f(\mathbf{x})) \in \mathbb{R}^{n+1}$, where f is a scalar multi-affine function and $\mathbf{x} = (x_1, \ldots, x_n)$ is a point lying in $\mathcal{B} = [x_1^l, x_1^u] \times \cdots \times [x_n^l, x_n^u]$, is contained in the convex hull of the 2^n points $\{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in \{x_1^l, x_1^u\} \times \ldots \times \{x_n^l, x_n^u\}\}$.

In other words, the graph of $f(\mathbf{x})$ corresponding to a box $\mathcal{B}$ necessarily lies inside the convex hull of the 2^n points of the form $(\mathbf{x}, f(\mathbf{x}))$, where $\mathbf{x}$ is a corner of $\mathcal{B}$.

This result can be readily used to refine an initial bound of the solution space of the system (3). For the sake of clarity, we explain how this is done when (3) consists of just one equation in two variables, and show at the end how the same process directly applies to the general case.

Assume that we want to find all solutions of a multi-affine equation $f(\mathbf{x}) = 0$, for $\mathbf{x} = (x_1, x_2)$ in the box $\mathcal{B} = [x_1^l, x_1^u] \times [x_2^l, x_2^u] \in \mathbb{R}^2$ (Fig. 2). Since $(\mathbf{x}, f(\mathbf{x}))$ must lie within the convex hull $\mathcal{H}$ of the 2^2 points $\{(\mathbf{x}, f(\mathbf{x}))| \ \mathbf{x} \in \{x_1^l, x_1^u\} \times \{x_2^l, x_2^u\}\}$ of $\mathbb{R}^3$, we can compute $\mathcal{H}$ and intersect it with the plane $f(\mathbf{x}) = 0$

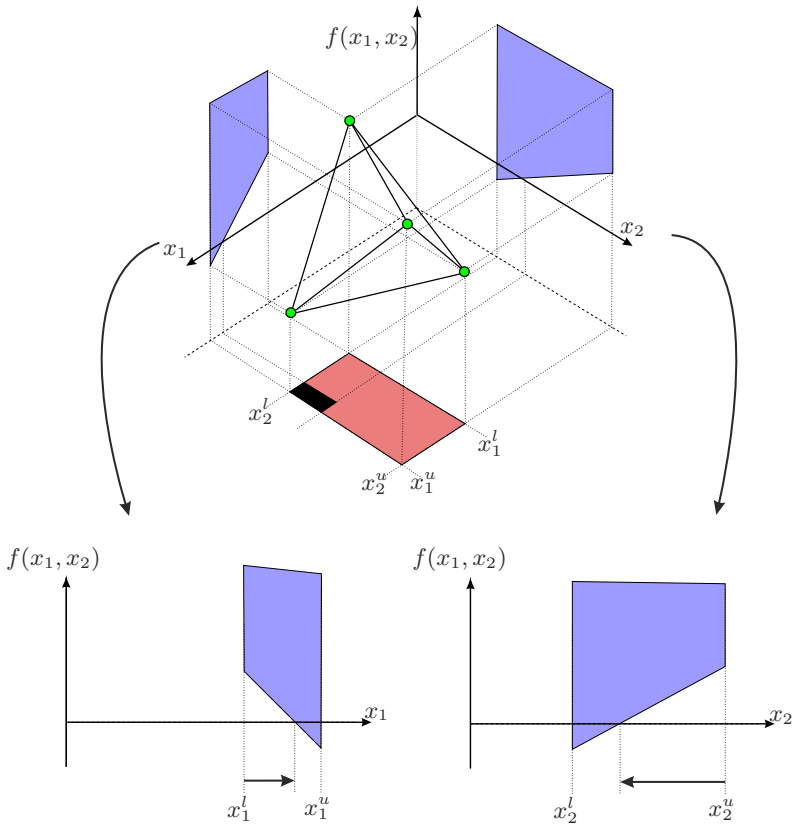

Fig. 2. If $f(x_1, x_2)$ is multi-affine, due to the convex hull property, the image of the points in the domain $[x_1^l, x_1^u] \times [x_2^l, x_2^u]$ (shown in red) necessarily lies inside the shown tetrahedron. The vertices of this tetrahedron (shown in green) are obtained by evaluating f in the corners of the domain. Then, from the projections of this tetrahedron onto the coordinate planes (shown in blue), the initial ranges for the variables can be reduced to the regions where these projections intersect the coordinate axes.

to obtain a polygon whose smallest enclosing box gives a better bound for the solutions. Since the explicit computation of $\mathcal{H}$ and its intersection with $f(\mathbf{x}) = 0$ are inefficient operations when f depends on a high number of variables, we will adopt the following variation of this technique: we simply project the hull $\mathcal{H}$ onto each coordinate plane, as depicted in Fig. 2 (top), and intersect each of the resulting trapezoids with the $f(\mathbf{x}) = 0$ line, as shown in Fig. 2 (bottom). Clearly, from the initial range of a variable we can prune those points for which the trapezoid does not intersect the $f(\mathbf{x}) = 0$ line. Thus, these segment-trapezoid clippings usually reduce the ranges of some variables giving a smaller box that still bounds the root locations (see the black rectangle in Fig. 2). The experiments show that, although this strategy produces less pruning than the convex hull-plane clipping, it results advantageous due to the lower cost of its operations.

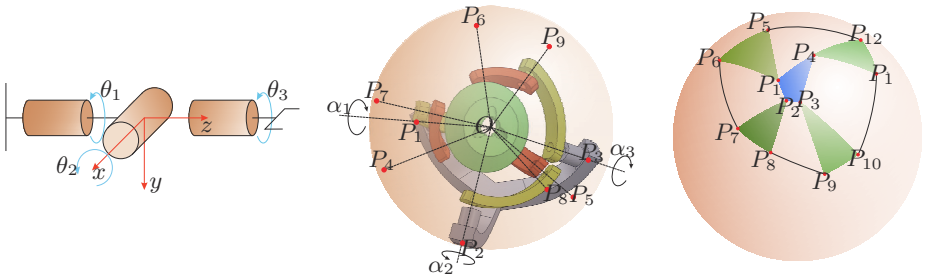

Fig. 3. The three test cases used to validate the branch-and-prune algorithm presented in this paper. Left: A spherical wrist. Center: 3-RRR parallel spherical manipulator. Right: A four loop spherical mechanism.

Note that exactly the same pruning strategy can be applied to a multi-affine equation in n variables, with $n > 2$, because the convex hull of the (then) involved 2^n points will also yield a trapezoidal polygon when projected to a plane defined by the x_i and $f(\mathbf{x})$ axes, for $i = 1, \ldots, n$. Finally, if we have more than one equation in the system, the process can be clearly iterated for all equations by applying the pruning process of each equation onto the box produced by the pruning process of the preceding one.

The previous box reduction procedure is repeatedly applied on a same box until one of the following conditions is met:

- The box gets empty. This happens when we are treating an equation $f(\mathbf{x}) = 0$ and a trapezoid does not intersect the corresponding coordinate axis. In any case, we can stop the exploration in the current box.
- The reduction of the box volume between two consecutive iterations is below a given threshold ρ. At this point, as the box cannot be significantly reduced, it is split into two sub-boxes via box bisection. This operation simply divides the longest variable range of the box into two halves. Box reduction and bisection are then recursively applied to the newly created sub-boxes.
- The longest side of the box is under a specified threshold σ. Here, the box is considered a "solution box" since, if σ is sufficiently small, either this box contains a solution or is very close to one. Hence, it is added to the final list of boxes to be returned by the algorithm.

5 Examples

The described branch-and-prune algorithm has been implemented in C and evaluated on three mechanisms of increasing complexity (see Fig. 3). The first one is a simple spherical wrist whose inverse kinematics consists in solving a single kinematic loop. The second is a 3-RRR [15] orienting robot whose forward kinematics consists in solving two independent kinematic loops. The third one is a spherical mechanism with four independent kinematic loops [16].

Table 1. Summary of the results in the tree test cases. See the text for details.

l	r	p	b	b_e	b_s	s	t[ms]
1	3	8	1	0	1	1	3
2	6	64	241	107	14	2	5
4	12	2048	563	276	6	2	9

The inverse kinematics of the spherical wrist in Fig. 3(left) consists in obtaining the values of $t_i = \tan\left(\frac{\phi_i}{2}\right)$, $i = 1, 2, 3$, which satisfy the following equation:

$$\frac{1}{\sqrt{(t_1^2+1)(t_2^2+1)(t_3^2+1)}}(1+t_1k)(1+t_2i)(1+t_3k)q_o^{-1} = 1, \tag{4}$$

where q_0 is the desired orientation for the end-effector. If we set $q_0^{-1} = e_0 + e_1 i + e_2 j + e_3 k$, we derive a set of three multi-affine equations which can be expressed in matrix form as:

$$\begin{pmatrix} e_1 & -e_2 & e_0 & -e_2 & e_3 & -e_1 & -e_3 & e_0 \\ e_2 & e_1 & -e_3 & e_1 & e_0 & -e_2 & -e_0 & -e_3 \\ e_3 & e_0 & e_2 & e_0 & -e_1 & -e_3 & e_1 & e_2 \end{pmatrix} \begin{pmatrix} 1 \\ t_1 \\ t_2 \\ t_3 \\ t_1t_2 \\ t_1t_3 \\ t_2t_3 \\ t_1t_2t_3 \end{pmatrix} = 0 \tag{5}$$

The problem of solving the forward kinematics of the 3-$\underline{\text{R}}$RR spherical parallel manipulator shown in Fig. 3(center) consist in locating the triangle $P_7P_8P_9$ with respect to the base $P_1P_2P_3$ as a function of the motor angles θ_1, θ_2, and θ_3. This mechanism has two independent kinematic loops, which, when formulated in terms of quaternions as described in Section 2, give six multi-affine equations in six unknowns.

Finally, the four-loop mechanism in Fig. 3(rigth) can be formulated, once represented using quaternions, in terms of 12 multi-affine equations in 12 unknowns.

Table 1 summarizes the performance of the algorithm described in Section 3 with $\rho = 0.99$ and $\sigma = 10^{-4}$. For each case, the table gives the number of loops (l), the number of variables (r), the number of initial boxes with all the variables in the range $[-1, 1]$ that can be defined ($p = 2^r$), the number of processed boxes for one of those initial search boxes (b), the number of empty boxes (b_e), the number of solution boxes (b_s), the number of roots of the system (s), and the execution time in milliseconds (t). The program is executed on a PC with an Intel Core 7 processor running at 4.2 GHz.

6 Conclusion

We have presented an easy-to-implement branch-and-prune algorithm for solving position analysis problems of spherical mechanisms. The simplicity of the proposed algorithm comes with two main disadvantages; namely:

- The use of one equation at a time leads to the so-called cluster problem, that is, each solution is obtained as a cluster of boxes instead of a single box containing it [17]. This is clear in Table 1 where, for the two-loop and four-loop problems, the algorithm returns more solution boxes (b_s) than the actual number of roots (s). All the solution boxes, though, are close to the actual solutions and the error (*i.e.*, the norm of the residue of the equations) evaluated at the center of the solution boxes is below 10^{-3} in all cases. Thus, the clusters for each solution can be readily identified.
- The use of a single variable at a time makes the convergence to the roots be linear while other standard interval-based algorithms exhibit quadratic convergence [18]. Although our algorithm compares unfavorably in this respect, it should be beared in mind that we are actually facing a trade-off between cost of each iteration and convergence rate.

Despite these drawbacks, the presented algorithm is remarkable for its simplicity, efficiency, and easy parallelization.

Finally, it is worth to mention that its extension to spatial mechanisms is perfectly feasible by approximating, for example, dual quaternions by double quaternions **??**. This point is currently concentrating our efforts.

References

1. M. Raghavan and B. Roth, "Inverse kinematics of the general 6R manipulator and related linkages," *ASME Journal of Mechanical Design*, Vol. 115, pp. 502-508, 1993.
2. M.L.Husty, "An algorithm for solving the direct kinematics of general Stewart-Gough platforms,"' *Mechanism and Machine Theory*, Vol. 31, No. 4, pp. 365-379, 1996.
3. J.M. Porta and F. Thomas, "Yet another approach to the Gough-Stewart platform forward kinematics," *Proc. of the IEEE International Conference in Robotics and Automation*, Brisbane, Australia, 2018.
4. C. Wampler, A. Morgan, and A. Sommese, "Numerical continuation methods for solving polynomial systems arising in kinematics," *ASME Journal Mechanical Design*, Vol. 112, pp. 59-68, 1990.
5. A. Castellet and F. Thomas, "An algorithm for the solution of inverse kinematics problems based on an interval method," Advances in Robot Kinematics, M. Husty and J. Lenarcic (Eds.), pp. 393-403, Kluwer, 1998.
6. C.W. Wampler, "Displacement analysis of spherical mechanisms having three or fewer loops," *ASME Journal of Mechanical Design*, Vol. 126, No. 1, pp. 93-100, 2004.
7. S. Bai, M. R.Hansen, and J. Angeles, "A robust forward-displacement analysis of spherical parallel robots," *Mechanism and Machine Theory*, Vol. 44, No. 12, pp. 2204-2216, 2009.

8. C. Bombín, L. Ros, and F. Thomas, "A concise Bezier clipping technique for solving inverse kinematics problems," *Advances in Robot Kinematics*, J. Lenarcic and M. Stanisic (Eds.), pp.53-61, Kluwer, 2000.
9. C. Bombín, L. Ros, and F. Thomas, "On the computation of the direct kinematics of parallel spherical mechanisms using Bernstein polynomials," *Proc. IEEE Int. Conf. on Robotics and Automation*, Seoul, Korea, 2001.
10. E. Celaya, "Interval propagation for solving parallel spherical mechanisms," Advances in Robot Kinematics, J. Lenarcic and F. Thomas (Eds.), pp. 415-422, Springer, 2002.
11. J. Gallier, *Curves and Surfaces in Geometric Modeling: Theory And Algorithms*, Morgan Kaufmann, 1999.
12. J.M. Porta, L. Ros, F. Thomas, and C. Torras, "A branch-and-prune solver for distance constraints," *IEEE Transactions on Robotics*, Vol. 21, No. 2, pp. 176-187, 2005.
13. A. Rikun, *A convex envelope formula for multilinear functions*, *Journal of Global Optimization*, Vol. 10, No. 4, pp.425-437, 1970.
14. R. Wagner, "Multi-linear interpolation," Web document available at: http://rjwagner49.com/Mathematics/Interpolation.pdf, 2004.
15. X. Kong and C.M. Gosselin, "Type synthesis of 3-DOF spherical parallel manipulators based on Screw Theory," *ASME Journal Mechanical Design*, Vol. 126, No. 1, pp. 101-108, 2004.
16. J. Borrás, R. Di Gregorio, "Direct position analysis of a large family of spherical and planar parallel manipulators with four loops," *Second International Workshop on Fundamental Issues and Future Research Directions for Parallel Mechanisms and Manipulators*, pp. 19-29, 2008.
17. A. Morgan and V. Shapiro, "Box-bisection for solving second-degree systems and the problem of clustering," *ACM Transactions on Mathematical Software*, Vol. 13, No. 2, pp. 152-167, 1987.
18. E. Sherbrooke and N. Patrikalakis, "Computation of the solutions of nonlinear polynomial systems," *Computer Aided Geometric Design*, Vol. 10, No. 5, pp. 379-405, 1993.
19. F. Thomas, "Approaching dual quaternions from matrix algebra," *IEEE Transactions on Robotics*, Vol. 30, No. 5, pp. 1037-1048, 2014.