A Branch-and-Prune Solver for Distance Constraints

Josep M. Porta, Lluís Ros, Federico Thomas, and Carme Torras

***Abstract*—Given some geometric elements such as points and lines in $\mathbb{R}^3$, subject to a set of pairwise distance constraints, the problem tackled in this paper is that of finding all possible configurations of these elements that satisfy the constraints. Many problems in robotics (such as the position analysis of serial and parallel manipulators) and CAD/CAM (such as the interactive placement of objects) can be formulated in this way. The strategy herein proposed consists of looking for some of the *a priori* unknown distances, whose derivation permits solving the problem rather trivially. Finding these distances relies on a branch-and-prune technique, which iteratively eliminates from the space of distances entire regions which cannot contain any solution. This elimination is accomplished by applying redundant necessary conditions derived from the theory of distance geometry. The experimental results qualify this approach as a promising one.**

***Index Terms*—Branch-and-prune, Cayley–Menger determinant, direct and inverse kinematics, distance constraint, interval method, kinematic and geometric constraint solving, octahedral manipulator.**

I. Introduction

THE resolution of systems of kinematic or geometric constraints has aroused interest in many areas of robotics (contact analysis, assembly planning, position analysis of serial and parallel manipulators, path planning of closed-loop kinematic chains, etc.) and CAD/CAM (constraint-based sketching and design, interactive placement of objects, etc.). The solution of such problems entails finding all object positions and orientations that simultaneously satisfy a number of constraints. Examples of such constraints are the closure conditions induced by loops of articulated solids, or orthogonality and parallelism relationships between geometric primitives.

Although the problem can be approached by using geometric constructive techniques [2], only the algebraic approaches have proved general enough to handle all problem instances. These consist of translating the original geometric problem into a system of algebraic equations that is then solved using any suitable standard technique. Unfortunately, a good solution to both algebrization and resolution, treated as independent problems, does not necessarily lead to an efficient solution to the geometric problem. Our aim in this work has been on finding a good combination of algebraization and resolution so that the whole process is easy to understand and to implement, and yet computationally efficient in practice.

Finding all solutions to a system of nonlinear polynomial equations within some finite domain is a ubiquitous problem, for which a wealth of resolution techniques have been proposed. Rewiews of these methods in the context of robotics, CAD/CAM, and molecular conformation can be found, for example, in [3]–[6]. Broadly speaking, the proposed methods fall into three categories, depending on whether they use algebraic geometry, continuation, or interval-based techniques.

The idea of algebraic-geometric methods, including those based on resultants and Gröbner bases, is to use variable elimination in order to reduce the initial system to a univariate polynomial. The roots of this polynomial, once backsubstituted into other equations, yield all solutions of the original system. These methods have proved quite efficient in fairly nontrivial problems, such as the inverse kinematics of general 6R manipulators [7], distance computations of two-dimensional objects [8], or the generation of configuration-space obstacles [9]. Recent progress on the theory of sparse resultants, moreover, qualifies them as a very promising set of techniques [10]–[12].

The idea of continuation methods, on the other hand, is to begin with an initial system whose solutions are known, and then transform it gradually to the system whose solutions are sought, while tracking all solution paths along the way. In its original form, this technique was known as the *Bootstrap Method*, as developed by Roth and Freudenstein [13], and subsequent work by Garcia and Li [14], Garcia and Zangwill [15], Morgan [16], and Li *et al.* [17], among others, led to the procedure into its current highly developed state [18]. This method has been responsible for the first solutions of many long-standing problems in kinematics. For example, using them, Tsai and Morgan first showed that the inverse kinematics of the general 6R manipulator has 16 solutions [19], Raghavan showed that the direct kinematics of the general Stewart–Gough platform can have 40 solutions [20], and Wampler *et al.* solved nine-point path-synthesis problems for four-bar linkages [21].

While methods in the two previous categories are, in theory, *complete* (they are able to find *all* solutions if these exist in a finite number) and *general* (they can tackle *any* system of multivariate polynomial equations), they have a number of limitations in practice. For example, algebraic-geometric methods usually explode in complexity, may introduce extraneous roots, and can only be applied to relatively simple systems of equations. Beyond this, they may require the solution of a high-degree polynomial, which may be a numerically ill-conditioned step in some cases. Also, as noted in [22], continuation techniques must be implemented in exact rational arithmetic to avoid

Manuscript received September 23, 2003; revised March 31, 2004. This paper was recommended for publication by Associate Editor J. M. Schimmels and Editor I. Walker upon evaluation of the reviewers' comments. This work was supported in part by the Spanish CICYT under Contracts TIC2000-0696 and TIC2003-03396, and in part by the Catalan Research Commission through the "Robotics and Control" group. The work of J. M. Porta and L. Ros was supported by a Ramón y Cajal contract from the Spanish Ministry for Science and Technology. This paper was presented in part at the IEEE International Conference on Robotics and Automation, Taipei, Taiwan, R.O.C., September 2003.

The authors are with the Institut de Robòtica i Informàtica Industrial (CSIC-UPC), 08028 Barcelona Catalonia, Spain (e-mail: jporta@iri.upc.edu; ,llros@iri.upc.edu; fthomas@iri.upc.edu; ctorras@iri.upc.edu).

Digital Object Identifier 10.1109/TRO.2004.835450

numerical instabilities, leading to important memory requirements, because large systems of complex initial-value problems have to be solved. For an arbitrary problem, moreover, neither of these approaches is able to obtain the solution variety, or at least characterize it to some extent, if its dimension is greater than zero.

Interval-based methods are also complete and general, and, although they can be slow in practice, they present a number of advantages that make them a competitive alternative: 1) contrary to elimination methods, the equations are tackled in their input form, thus avoiding the need of intuition-guided symbolic reductions; 2) they are numerically stable; 3) they also work if the dimension of the solution variety is greater than zero; 4) they deal with variable bounds in a natural way; and 5) they are simple to implement. These are mainly the reasons that motivated the quest for the algorithm we present here, which belongs to this third class.

Two main classes of interval-based methods have been explored in the robotics literature, those based on the interval version of the Newton method (also known as the Hansen algorithm), and those based on subdivision. To our knowledge, the first applications of the Hansen algorithm in this field were due to Rao *et al.* [23] and Didrit *et al.* [24], who, respectively, applied the interval Newton method to the inverse kinematics of 6R manipulators and the forward analysis of Stewart–Gough platforms. Rather than plunging into specific mechanisms, Castellet and Thomas then tackled general single-loop inverse kinematics problems [25], showing that the Hansen algorithm can be sped up if it is used in conjunction with other necessary conditions drawn from the problem itself. Later on, successful applications of the interval Newton method were also reported by Merlet in singularity analysis and mechanism design of parallel manipulators [26], [27]. Subdivision techniques, in turn, were developed in the early 1990s by Sherbrooke and Patrikalakis in the context of constraint-based CAD [22]. These exploit the subdivision property of Bernstein polynomials, which avoids the computation of derivatives, while maintaining the quadratic convergence of the Hansen algorithm. Their application to general multiloop mechanisms was made possible after explicit expressions for the control points of their closure equations were found in [28], allowing their rewriting in Bernstein form. A specific subdivision technique was then developed in [29], which leads to a remarkably simpler algorithm when the problem can be described only by multilinear constraints. (A constraint is said to be *multilinear* if it is linear in each of its variables.) Given this simplicity, it seems logical to elucidate whether a formulation of every kinematic or geometric constraint-solving problem is possible in terms of such constraints exclusively. We show in this paper that the theory of distance geometry allows such a formulation, thus permitting a reasonably good symbiosis between algebraization and resolution, as initially sought. This problem formulation and a novel subdivision-based constraint-solving technique for multilinear equations are, in sum, the main contributions of this paper.

The paper is structured as follows. In Section II, Cayley–Menger determinants are briefly introduced. Using them, in Section III, it is shown how kinematic constraints, such as loop-closure constraints, and geometric constraints, such as alignment or orthogonality, can be translated into constraints involving only distances. Then, the proposed branch-and-prune algorithm for systems of such constraints is detailed in Section IV. Two applications of the method in the areas of robot kinematics and geometric design are presented in Section V, and finally, some conclusions are drawn in the closing section.

II. Cayley-Menger Determinants

Let us define the function

$$\Xi(\mathbf{p}_1,\ldots,\mathbf{p}_n)=\begin{vmatrix} 0 & r_{1,2} & r_{1,3} & \cdots & r_{1,n} & 1 \\ r_{2,1} & 0 & r_{2,3} & \cdots & r_{2,n} & 1 \\ r_{3,1} & r_{3,2} & 0 & \cdots & r_{3,n} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{n,1} & r_{n,2} & r_{n,3} & \cdots & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{vmatrix}$$

where $\mathbf{p}_1,\ldots,\mathbf{p}_n$ are n points in $\mathbb{R}^3$ and $r_{i,j}=\|\mathbf{p}_i-\mathbf{p}_j\|^2$, i.e., the square distance between $\mathbf{p}_i$ and $\mathbf{p}_j$. Obviously, $r_{i,j}=r_{j,i}$. The previous determinant is the general form of the Cayley–Menger determinant. It was first used by A. Cayley in 1841 [30], but it was not systematically studied until 1928, when K. Menger showed how it could be used to study convexity and other basic geometric problems [31]. Nowadays, this determinant plays a fundamental role in the so-called distance geometry, a term coined by Blumenthal in [32], which refers to the analytical study of Euclidean geometry in terms of invariants without resorting to artificial coordinate systems.

If $n=2$

$$\Xi(\mathbf{p}_1,\mathbf{p}_2)=2\,r_{1,2}. \tag{1}$$

If $n=3$

$$\Xi(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3)=-16\,A^2 \tag{2}$$

where A is the area of the triangle defined by $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$. Actually, (2) is Herron's formula, which permits to obtain the area of a triangle in terms of the lengths of its edges.

If $n=4$

$$\Xi(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4)=288\,V^2 \tag{3}$$

where V is the volume of the tetrahedron defined by $\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$, and $\mathbf{p}_4$. If $\Xi(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4)$ vanishes, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, and $\mathbf{p}_4$ lie on the same plane. If it gives a negative value, the tetrahedron cannot be assembled with the given distances. Actually, (3) is known as Euler's tetrahedron formula.

If $n>4$

$$\Xi(\mathbf{p}_1,\ldots,\mathbf{p}_n)=0 \tag{4}$$

because this determinant essentially gives the volume of a simplex in $\mathbb{R}^{n-1}$ but, since this simplex is degenerate in $\mathbb{R}^3$, its volume is zero.

If we have a set of points $\mathbf{p}_1,\ldots,\mathbf{p}_m$, and all distances between pairs of them are given, we can use conditions of this kind to check whether such a point configuration is actually embeddable in $\mathbb{R}^3$. The following result of distance geometry, used later on below, provides a set of necessary and sufficient conditions to this end [32].

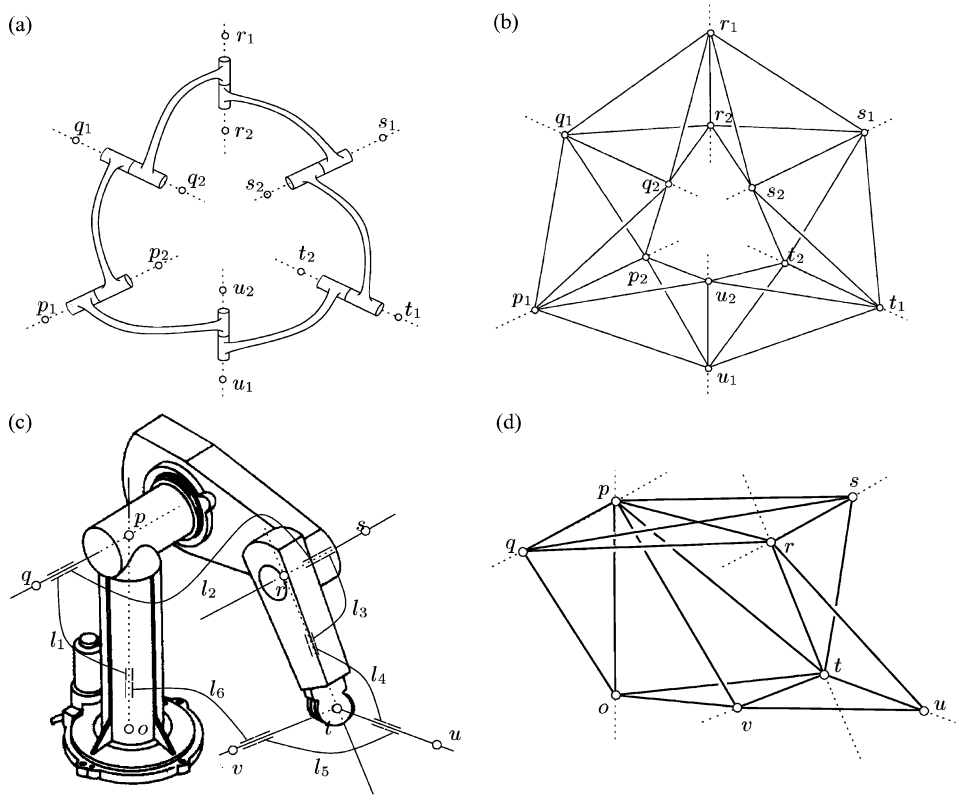


Fig. 1. Top: (a) a general 6R linkage and its equivalent bar-and-joint model. (b) A tetrahedral ring. Bottom: (c) a Puma 560 robot with, overlaid, its six binary links. (d) Its equivalent bar-and-joint model.

The points $\mathbf{p}_1, \ldots, \mathbf{p}_m$ are embeddable in $\mathbb{R}^3$, satisfying the prescribed distances between them, if, and only if, all the following conditions are met:

- R1) all Cayley–Menger determinants of three such points are either negative or zero;
- R2) all Cayley–Menger determinants of four points are either positive or zero;
- R3) all Cayley-Menger determinants of five and six points vanish.

Note that conditions R1, R2, and R3 are indeed necessary as, in accordance with (2), (3), and (4), a Cayley–Menger determinant must be negative or zero, positive or zero, or strictly zero, depending on whether it involves three, four, or more than four points, respectively. Actually, Blumenthal proves a slightly stronger version of this theorem, in which it is sufficient to find an ordering of $\mathbf{p}_1, \ldots, \mathbf{p}_m$ such that the first four points in this ordering satisfy conditions R1 and R2, and then only those Cayley–Menger determinants of five and six points that include these four points vanish. For the purpose of this paper, though, the previous weaker version will be used, as it provides extra redundant equations that enhance the convergence behavior of the presented solver.

III. From Kinematic and Geometric Constraints to Distance Constraints

Many problems of direct and inverse kinematics can be expressed in terms of systems of distance constraints, like those derived from conditions R1–R3 above. Consider, for example, the problem of finding all valid configurations of a closed 6R linkage, a cycle of six binary links pairwise articulated with revolute joints [Fig. 1(a)]. A binary link can be modeled by taking two points on each of its two revolute axes and connecting the four points with rigid bars to form a tetrahedron. Bars meeting at a common point are thought of as articulated through ideal ball-socket joints. By doing so, a 6R linkage is easily translated into a ring of six tetrahedra, pairwise articulated through a common edge [Fig. 1(b)]. Observe that the valid configurations of this ring are in one-to-one correspondence with those of the original 6R linkage. Since the ring only involves ball-socket joints and rigid bars, it can be regarded as a set of points (the joints) that keep some prescribed distances between them (the bars). Hence, all unknown distances within the ring must fulfill all conditions R1–R3, above which, when gathered together, they form a polynomial system whose solutions yield the valid postures of the ring.

When the axes of a link are not skew but intersecting, the link can be modeled by a rigid triangle rather than by a tetrahedron, thus using less ball-socket joints and bars. The case of the PUMA 560 robot depicted in Fig. 1(c) illustrates this. As in each of the links l_1, l_3, l_4, and l_5, the two axes are copunctual, they can be substituted by a triangle in the equivalent bar-and-joint model of Fig. 1(d). The last link l_6 is fictitious, but it is drawn here to emphasize that, in the inverse kinematics of such a robot, the desired position for the robot's hand is *a priori* known.

This process can be generalized to larger classes of mechanisms. When multiple loops are present, for example, some of the links will not be binary, but will necessarily involve three

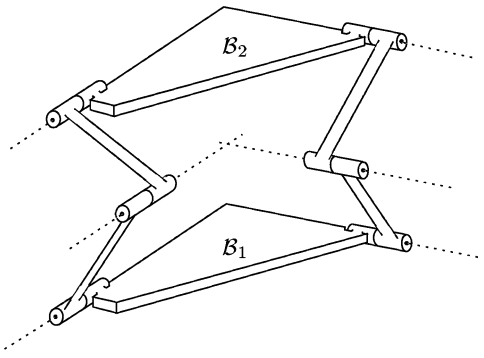


Fig. 2. Sarrus's mechanism. Since the axes of the three revolute pairs in each leg are parallel, body $\mathcal{B}_2$ can only translate with respect to $\mathcal{B}_1$ and, hence, this device can be used to implement a prismatic pair using revolute joints alone.

or more joint axes. In general, a link with m revolute axes is easily modeled by taking two points on each axis and placing all possible bars between the selected $2m$ points to make the whole compound rigid. On the other hand, if a prismatic pair is present, one can always substitute it with Sarrus's equivalent mechanism (Fig. 2), made up with revolute joints alone, and apply the previous transformations to its links. In any case, if only revolute and prismatic pairs are present, a reduction to an equivalent bar-and-joint model is always possible, for which a system of coordinate-free equations giving its valid configurations can be set up by using conditions R1–R3.

An interesting remark here is that one may be able to detect that a closed-form solution for the mechanism is available, by simply examining which unknowns are present in each of the equations derived from condition R3, without requiring any algebraic manipulation of them. For example, it can be seen that, for the PUMA 560, a sequence of ten Cayley–Menger equations of five points can be selected among the points o, p, q, r, s, t, u, and v of Fig. 1(d), such that every equation in the sequence contains one more variable than the preceding one. Being in echelon form and quadratic in all unknowns, this subsystem is solvable in closed form, and already determines the ten unknown distances among these points, the ten missing "bars" in Fig. 1(d).

Many geometric constraints can also be expressed in a coordinate-free form in terms of distances, by using Cayley–Menger determinants. To give some examples, we here derive three such constraints: collinear points, orthogonal segments, and point-line distance.

- Three points $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are *collinear* if, and only if, $\Xi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = 0$. This follows from (2), since the area of the triangle defined by three collinear points is null.
- Two adjacent segments $\mathbf{p}_1\mathbf{p}_2$ and $\mathbf{p}_2\mathbf{p}_3$ are *orthogonal* if, and only if, $\Xi(\mathbf{p}_1, \mathbf{p}_2) + \Xi(\mathbf{p}_2, \mathbf{p}_3) - \Xi(\mathbf{p}_1, \mathbf{p}_3) = 0$. This is a rewriting of Pythagoras' theorem by using (1).
- Finally, the distance d between a point $\mathbf{p}_1$ and a line passing through $\mathbf{p}_2$ and $\mathbf{p}_3$ satisfies the equation

$$\Xi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) + 4r_{2,3}d^2 = 0 \tag{5}$$

which follows from (2) and the fact that, in this case, $A^2 = r_{2,3}d^2/4$.

Section V will exemplify how some kinematic and geometric constraint-solving problems can be formulated and solved on the basis of the distance constraints introduced above.

IV. The Algorithm

We now present an algorithm able to solve systems of multilinear constraints. Since both Cayley–Menger determinants and identity relations $r_{i,j} = r_{j,i}$ are multilinear, this algorithm can be readily used to solve systems of distance constraints, like those derived in the previous section for kinematic or geometric constraint-solving problems. Specifically, for a system

$$\mathbf{F}(\mathbf{x}) = 0, \quad \mathbf{G}(\mathbf{x}) \geq 0 \tag{6}$$

where $\mathbf{F} = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$, $\mathbf{G} = (g_1(\mathbf{x}), \ldots, g_l(\mathbf{x}))$, and each function f_i or g_i is multilinear in the unknowns $x_1, \ldots, x_n$ (the unknown distances $r_{i,j}$, in our case), the algorithm is able to isolate all solutions that lie in a prespecified rectangular box $\mathcal{B}$ of $\mathbb{R}^n$. $\mathcal{B}$ is defined as the Cartesian product $\mathcal{B} = [x_1^l, x_1^u] \times \cdots \times [x_n^l, x_n^u]$, where $[x_i^l, x_i^u]$ denotes the closed real interval in which the solution values for x_i must be sought.

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 the box is reduced to an empty set, in which case it contains no solution, or the box is "sufficiently" small, in which case it is considered a solution box, or the box cannot be "significantly" reduced, in which case it is split into two subboxes via box bisection. If the latter occurs, the whole process is repeated for the newly created subboxes, and for the subboxes 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 solver's algorithm.

B. Box Reduction and Box Bisection

The box-reduction operation is based on the following lemma, which is a direct consequence of [33, Th. 1].

The point $(\mathbf{x}, f(\mathbf{x})) \in \mathbb{R}^{n+1}$, where f is a scalar multilinear 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 $\{(\mathbf{x}, f(\mathbf{x}))|\mathbf{x} \in \{x_1^l, x_1^u\} \times \ldots \times \{x_n^l, x_n^u\}\}$.

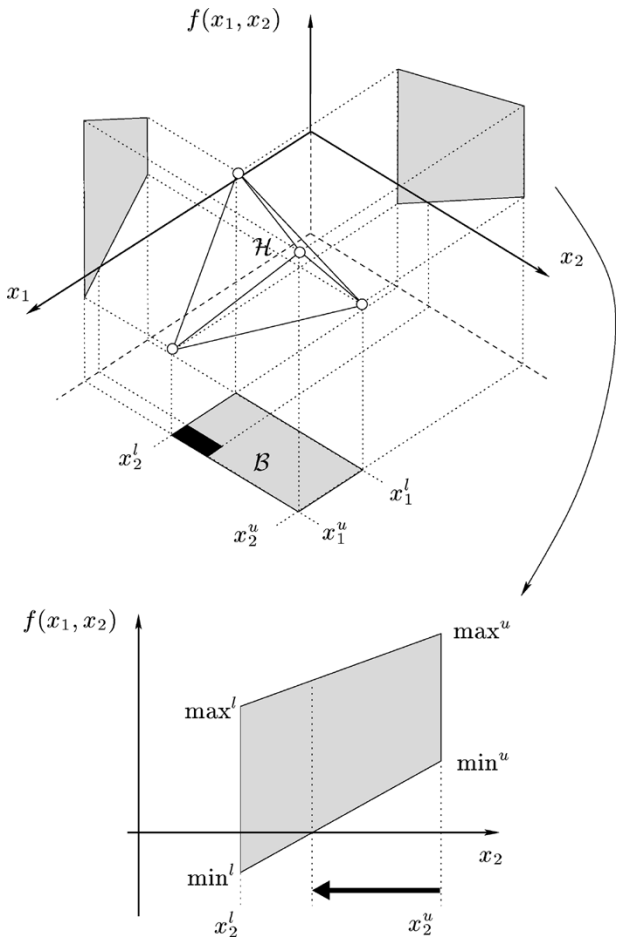


Fig. 3. Segment-trapezoid clipping. Top: in this two-variable case, the graph of $f(x_1, x_2)$ corresponding to points in $\mathcal{B}$ necessarily lies inside the shown tetrahedron $\mathcal{H}$. The vertices of $\mathcal{H}$ are obtained by evaluating f in the corners of $\mathcal{B}$. Bottom: from the initial range for a variable, we can prune all points for which its trapezoid does not intersect the $f(x_1, x_2) = 0$ line.

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 (6). For the sake of clarity, we explain how this is done when (6) 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 multilinear 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. 3). 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 $f(\mathbf{x}) = 0$ 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. 3 (top), and intersect each of the resulting trapezoids with the $f(\mathbf{x}) = 0$ line, as shown in Fig. 3 (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. 3). The experiments show that although this strategy produces less pruning than the convex hull-plane clipping, its results are advantageous due to the lower cost of its operations.

Note that exactly the same pruning strategy can be applied for a multilinear 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 $i = 1, \ldots, n$. Also, if instead of an equation, we have an inequation $g(\mathbf{x}) \geq 0$, we proceed similarly by pruning all values of a variable range for which its trapezoid entirely lies in the $g(\mathbf{x}) < 0$ half-plane. Finally, if we have more than one constraint in the system, the process can be clearly iterated for all constraints by applying the pruning process of each constraint onto the box produced by the pruning process of the preceding one.

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

- The box gets empty. This happens when we are treating an inequation $g(\mathbf{x}) \geq 0$, and the trapezoid for some variable in $g(\mathbf{x})$ is entirely in the $g(\mathbf{x}) < 0$ half-plane, or when we are treating an equation $f(\mathbf{x}) = 0$, and a trapezoid does not intersect the $f(\mathbf{x}) = 0$ line. In any case, a whole variable range is pruned, and we can stop the exploration in the search space the current box delimits.
- 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 subboxes via box bisection. This operation simply divides the longest variable range of the box into two halves, in order to keep the shape of the resulting subboxes as cubic as possible. Box reduction and bisection are then recursively applied to the newly created subboxes.
- 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.

C. Pseudocode

For the sake of conciseness, the given pseudocode only manages equations. Its extension to also deal with inequations is straightforward, if we take into account all comments above. The main loop, as schematized in Fig. 4, uses the following conventions. The arrow "$\leftarrow$" is the assignment operator. The Boolean expression Empty($\mathcal{B}$) is evaluated to *true* when any of the defining intervals of $\mathcal{B}$ is empty. The function *Extract_Box* returns (and removes) one box from its list argument. The expressions Vol($\mathcal{B}$) and Size($\mathcal{B}$) denote the volume of $\mathcal{B}$ and the length of the longest side of $\mathcal{B}$.

Initially, two lists are set up (lines 1 and 2), a list L of "boxes to be processed," and a list S of "solution boxes." L is initialized to contain $\mathcal{B}$, and S to an empty list. A *while* loop is then executed until L gets empty (lines 3–17), by iterating the following steps. Line 4 extracts the first box in L. Lines 5–8 repeatedly reduce this box as much as possible, via the *Reduce_Box* function,

The Solver
Input: m multilinear equations $f_1(\mathbf{x}) = 0, \ldots, f_m(\mathbf{x}) = 0$, an initial box $\mathcal{B}$, a size threshold σ, and a reduction threshold ρ.
Output: A set S of solution boxes.
Process:

```
1   S ← ∅
2   L ← B
3   while not Empty(L)
4       B_c ← Extract_Box(L)
5       do
6           V_prev ← Vol(B_c)
7           Reduce_Box(B_c, f_1(x), ..., f_m(x))
8       until Empty(B_c) or Size(B_c) ≤ σ or Vol(B_c)/V_prev > ρ
9       if not Empty(B_c) then
10          if Size(B_c) ≤ σ then
11              S ← S ∪ {B_c}
12          else
13              Split B_c into two sub-boxes: B_1, B_2
14              Add B_1 and B_2 to L
15          endif
16      endif
17  endwhile
18  return S
```

Fig. 4. Main loop of the equation solver.

Reduce_Box($\mathcal{B}$, $f_1(\mathbf{x})$, ..., $f_m(\mathbf{x})$)
Input: A box $\mathcal{B}$, defined as a set of intervals $\mathcal{B} = \{[x_1^l, x_1^u], \ldots, [x_n^l, x_n^u]\}$, and m multilinear equations $f_1(\mathbf{x}) = 0, \ldots, f_m(\mathbf{x}) = 0$.
Output: The same box, but eventually resized.
Process:

```
1   for each equation f_i(x) = 0
2       V ← {v_1, ..., v_k} (Indices of variables in f_i)
3       C ← {x_{v_1}^l, x_{v_1}^u} × ... × {x_{v_k}^l, x_{v_k}^u}
4       for each v ∈ V
5           min^l  ← min {f_i(x) | x ∈ C, x_v = x_v^l}
6           max^l  ← max {f_i(x) | x ∈ C, x_v = x_v^l}
7           min^u  ← min {f_i(x) | x ∈ C, x_v = x_v^u}
8           max^u  ← max {f_i(x) | x ∈ C, x_v = x_v^u}
9           Trapezoid_Clipping(x_v^l, x_v^u, min^l, max^l, min^u, max^u)
10      endfor
11  endfor
```

Fig. 5. *Reduce_Box* function.

until either the box is empty (Empty($\mathcal{B}$) is true), or it is a solution box (Size($\mathcal{B}$) $\leq \sigma$), or it cannot be significantly reduced (Vol($\mathcal{B}$)/$V_{\text{prev}} > \rho$). Finally, if the box is not empty, lines 9–16 either add it to the list S of solution boxes, if it is sufficiently small, or they split it into two subboxes and add them to L for further processing. The list of solution boxes is finally returned as the algorithm's output in line 18.

The *Reduce_Box* function in Fig. 5 implements the box-reduction operation. As input, the function takes a box $\mathcal{B}$ and the m multilinear equations in the system (6), and returns the same box $\mathcal{B}$, eventually resized. For each equation $f_i(\mathbf{x}) = 0$, the following is done. Line 2 determines which of the n variables are actually present in the expression of $f_i(\mathbf{x})$. Their indexes are stored in V, and hereafter referred to as $v_1, \ldots, v_k$. Line 3 stores all corners of the box $[x_{v_1}^l, x_{v_1}^u] \times \ldots \times [x_{v_k}^l, x_{v_k}^u]$ into a set C. For each variable x_v present in f_i, then, lines 5–8 compute the values $\min^l$, $\max^l$, $\min^u$ and $\max^u$, the minimum and maximum values that the function $f_i(\mathbf{x})$ takes inside $\mathcal{B}$ when x_v is set to the lower and upper values of its range, respectively. These values define the trapezoid for this variable (see Fig. 3, bottom), and now, it only remains to intersect it with the $f_i(\mathbf{x}) = 0$ line, hoping to reduce the range for x_v. The call to *Trapezoid_Clipping* in line 9 performs this straightforward operation.

D. Performance

We emphasize that this algorithm follows the simplest possible scheme for an interval-based technique, which consists of iteratively taking one equation at a time and projecting an approximation of the solution set for this equation onto a single variable, to prune it accordingly. The key point is then obtaining these approximations. In the context of interval Newton methods, pruning a single variable by projecting approximations of the solution set onto this variable is known as a "Newton cut," which requires evaluating interval derivatives [34], [35]. When using algorithms based on subdivision, these approximations are just the convex hull of a set of control points for the input polynomials. Our algorithm for multilinear equations can be seen to be equivalent to that obtained using the standard subdivision approach [22], which is, in turn, as good as a Newton cut in the worst case. Obviously, such a simple algorithmic structure comes with advantages and disadvantages.

On the negative side, the use of one equation at a time leads to the so-called *cluster problem*, that is, each solution is obtained as a compact cluster of boxes instead of a single box containing it [36]. As will be shown in the experiments, fortunately, the use of redundant equations, which naturally appear in our formulation, reduces this effect. Also, the use of a single variable at a time makes the convergence to the roots be linear [22], while other standard interval-based algorithms exhibit quadratic convergence. Although our algorithm compares unfavorably in this respect, it should be borne in mind that we are actually facing a tradeoff between the cost of each iteration and convergence rate. However, despite these drawbacks, its stability, simplicity, easy parallelization, and speed for many problems make it attractive.

On the positive side, the algorithm does not suffer from two common problems of classical root-finding procedures. On the one hand, it is immune to singularities of the Jacobian of the equations because it does not use derivative information. Certainly, this is a typical drawback of many Newton–Raphson methods. Given a system of equations $F(\mathbf{x}) = 0$, such methods iteratively work on an estimation $\mathbf{x}_i$ of the solution to derive a better estimation $\mathbf{x}_{i+1}$, using the recurrence $\mathbf{x}_{i+1} = \mathbf{x}_i - D^{-1}F(\mathbf{x}_i)$, where D is the matrix of partial derivatives of $F(\mathbf{x})$. Clearly, when D is close to singular, the method may fail to converge, as the examples in Section V-A below will illustrate. On the other hand, it is well known that the numerical stability of polynomial root finding is often surprisingly low [37], [38], and that a very small perturbation in just a few coefficients can yield solutions completely different from the intended ones. For example, classic solutions to the forward and inverse kinematics problems that rely on solving a resultant polynomial must carefully deal with this issue, especially in configurations of the mechanism near a singularity, where solutions may be lost. Contrarily, our algorithm is robust in this sense, because it directly works with the input equations.

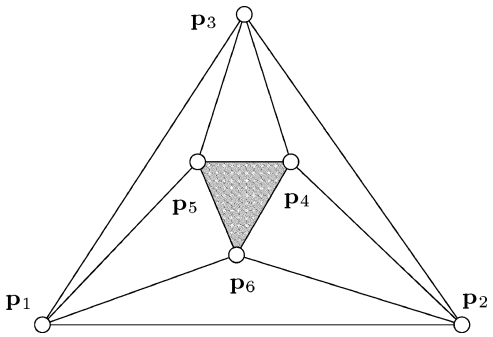


Fig. 6. Points involved in the forward kinematics of an octahedral manipulator.

V. Experiments

The algorithm has been implemented in C, and we next show how it performs in two test cases: solving the forward kinematics of octahedral manipulators, and finding all lines simultaneously tangent to four spheres.

A. Solving the Octahedral Manipulator

An octahedral manipulator is formed by two triangles, the *base* $\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and the *platform* $\mathbf{p}_4\mathbf{p}_5\mathbf{p}_6$, joined by six linearly actuated *legs*: $\mathbf{p}_1\mathbf{p}_5$, $\mathbf{p}_1\mathbf{p}_6$, $\mathbf{p}_2\mathbf{p}_6$, $\mathbf{p}_2\mathbf{p}_4$, $\mathbf{p}_3\mathbf{p}_4$, and $\mathbf{p}_3\mathbf{p}_5$ (Fig. 6). The forward kinematics problem is to find all poses of the platform (relative to the base) that are compatible with the six specified leg lengths. No closed-form solution to this problem is known, but numerical procedures have been given that involve finding the roots of an eighth-degree univariate polynomial, obtained by symbolic elimination techniques [39], [40]. Using Cayley–Menger determinants, though, it is possible to give the following simple formulation, directly solvable by the above algorithm. To this end, consider the following Cayley–Menger equations:

$$\begin{aligned} \Xi(\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6) &= 0 \\ \Xi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_6) &= 0. \end{aligned} \tag{7}$$

Note that, among all involved squared distances, only $r_{1,4}$ and $r_{3,6}$ are unknown in (7), and that once this system is solved for them, we can determine the spatial position of the three points of the platform by trilateration [41], since each of these points will have a tripod of known lengths with three points at a known position. Thus, although a third squared distance is also unknown in this problem $(r_{2,5})$, there is no need to take it into account if only these two equations are used. We may now use our algorithm to solve them.

Fig. 7(a) and (b) show the results for two different sets of leg lengths. In both cases, the base and platform triangles are equilateral, of side $\sqrt{3}$ and $\sqrt{3}/2$, respectively. Fig. 7(a) shows the solution boxes found when the squared leg lengths are set to $r_{1,5} = r_{2,6} = r_{3,4} = 4.25$ and $r_{1,6} = r_{2,4} = r_{3,5} = 5.75$, a case hereafter referred to as configuration "a." Fig. 7(b) shows the solution boxes when all squared leg lengths are set to 4.75, a case hereafter referred to as configuration "b." Insight into the behavior of the solver may be gotten by comparing these two outputs with the corresponding plots of the implicit curves of (7), shown in Fig. 7(e) and (f). Note that while in configuration "a," the two curves are rather different and intersect only in two points, in configuration "b," they are quite close to one another

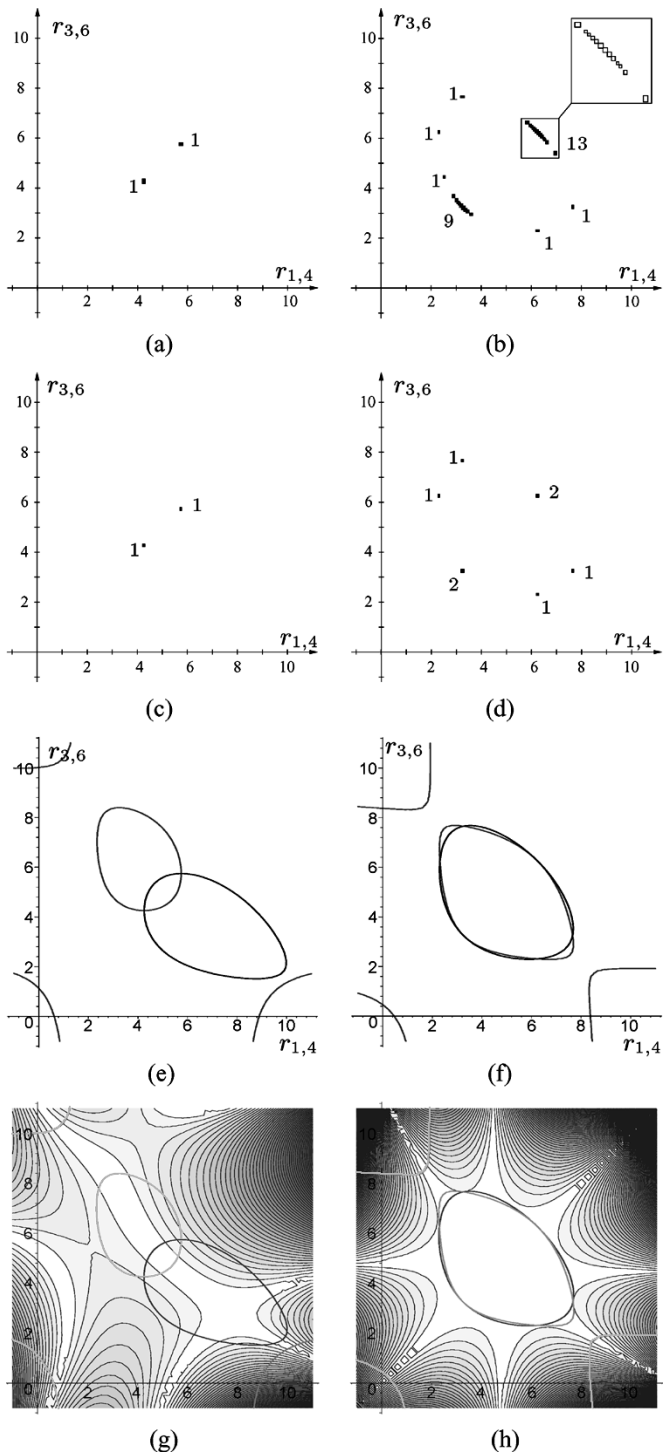


Fig. 7. Solving the octahedral manipulator. Left and right columns refer, respectively, to configurations "a" and "b". The numbers in (a), (b), (c), and (d) indicate the number of solution boxes returned around each solution point.

TABLE I
Algorithm's Performance on Forward Kinematics of the Octahedral Manipulator

	configuration "a"		configuration "b"	
	non redundant	redundant	non redundant	redundant
t	< 0.01 (0.09)	< 0.01 (0.35)	0.03 (0.28)	0.03 (2.15)
b_p	7 (851)	3 (1477)	65 (2173)	39 (6841)
n_s	2 (86)	2 (67)	27 (680)	8 (764)

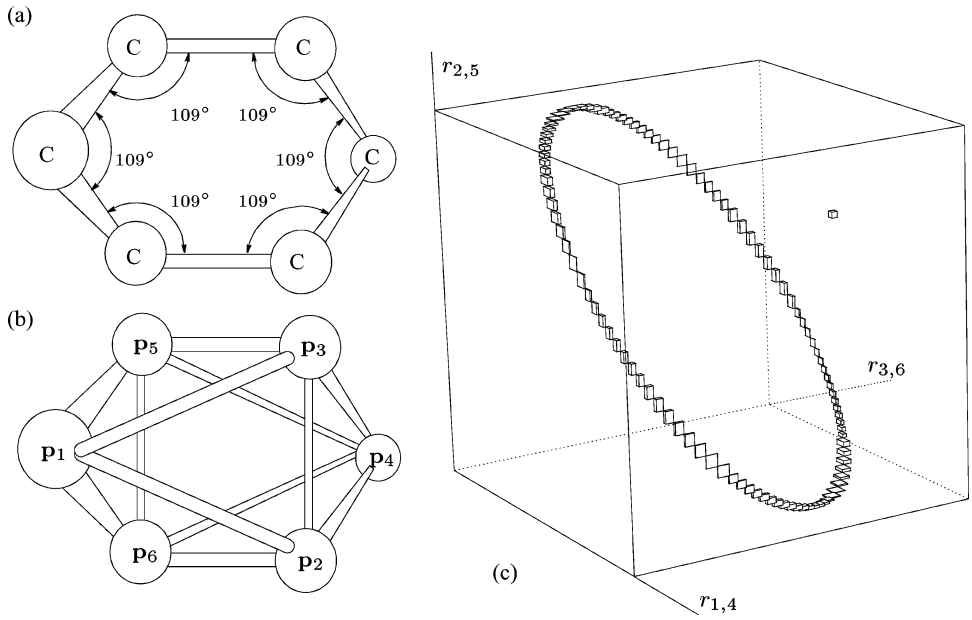


Fig. 8. (a) Cyclohexane molecule. (b) Its bar-and-joint model. (c) Solution boxes found by the solver, plotted in the space defined by the variables $r_{1,4}$, $r_{2,5}$, and $r_{3,6}$. The shown bounding box is defined by the intervals $r_{1,4} = r_{2,5} = r_{3,6} = [2.75, 3.95]$. The isolated box and the continuum correspond, respectively, to the "chair" and "boat" forms of this molecule.

and intersect in six points, with tangency on two of them. This proximity explains why our solver gives larger clusters of boxes in Fig. 7(b) than in Fig. 7(a).

We may add redundant equations to the system of (7). For example, if we add the remaining Cayley–Menger equations of five points

$$\begin{aligned}\Xi(\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4,\mathbf{p}_5,\mathbf{p}_6) &= 0\\ \Xi(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_4,\mathbf{p}_5,\mathbf{p}_6) &= 0\\ \Xi(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_5,\mathbf{p}_6) &= 0\\ \Xi(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4,\mathbf{p}_5) &= 0\end{aligned} \tag{8}$$

we end up with a system of six equations, now including the three unknowns of this problem. The solution boxes found by the solver are displayed in Fig. 7(c) and (d), for configurations "a" and "b," respectively. Comparing Fig. 7(a) and (b) with Fig. 7(c) and (d), we clearly see that the use of redundant equations produces extra pruning, and that the solutions are bounded with higher accuracy. Table I shows the execution time t (in seconds, on a Pentium IV PC at 1.8 GHz) for both configurations, the number b_p of boxes processed by the algorithm, and the number n_s of solution boxes found. These statistics are separately given for the nonredundant formulation of (7), and for the redundant one involving (7) and (8). In parentheses, the table also gives t, b_p, and n_s, for a slightly modified version of the algorithm that uses interval arithmetic to compute the vertical sides of the trapezoids, instead of evaluating the functions in the 2^n corners of each box. In all cases, the global control parameters have been set to $\sigma = 0.1$ and $\rho = 0.9$.

The table clearly shows the positive effect of adding redundant equations. Although the two configurations are solved in practically the same time, in the redundant case, fewer boxes have to be explored. It is remarkable that, for configuration "a," the redundancy of equations allows isolating the solutions by only exploring three boxes, the minimum required when two solutions exist. In configuration "b," the solver also isolates the solutions, but returns whole clusters of boxes for those lying in tangency points [Fig. 7(b)]. This effect is nevertheless reduced when adding redundancy, as the delivered clusters contain only two boxes each, as shown in Fig. 7(d). Furthermore, the cost of processing each box during the segment-trapezoid clipping is $O(2^n)$, where n is the maximum number of variables per equation. When using interval arithmetic in this process, this cost is reduced to $O(n)$, but despite this lower complexity, we observe that both t and b_p increase considerably in this case, as shown in the table. This is due to the looser bounds that interval arithmetic yields for the vertical sides of the trapezoids.

In view of Fig. 7(f), one may ask whether an octahedral manipulator exists for which the two oval curves coincide, thus yielding a whole continuum of solutions. Certainly, this may happen, if the manipulator's geometric parameters are those of a *Bricard octahedron*. As Bricard showed in [42], there are three distinct types of fully mobile octahedra. One is symmetrical about a line, another about a plane, and the third is such that the "opposite" angles at every vertex (in Fig. 6) are equal or supplementary. Mechanisms of this kind are said to be *architecturally singular*, and are usually avoided for being difficult to control. However, a specific realization of them exists in nature: the cyclohexane molecule, a cycle of six carbon atoms pairwise connected with simple covalent bonds [Fig. 8(a)]. Its structure can be viewed as six rigid bars of equal length cyclically connected with ball-socket joints at the atoms, plus the additional constraint that the angle between every two adjacent bars is of 109°—the usual angle between covalent bonds in a carbon. If we establish this constraint by adding extra bars of a proper length, we realize that the kinematic model of this molecule is actually an octahedron, as we get the bar-and-joint framework of Fig. 8(b), whose combinatorial structure is the same as that of the octahedron in Fig. 6.

Fig. 8(c) shows the solution boxes returned by the solver in this case, obtained by solving (7) and (8) together. Using $\sigma =$

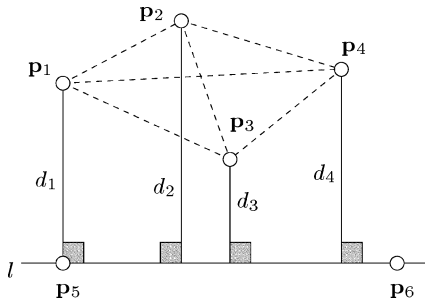


Fig. 9. Points involved in computing all lines tangent to four spheres.

0.05, 105 solution boxes have been found, in 0.39 s. In nature, the cyclohexane takes two forms. While the so-called *chair* form is rigid, and corresponds to the isolated box in Fig. 8(c), the *boat* form is flexible, and its feasible motions are described by the continuum of solution boxes shown in this figure.

Finally, these examples show that the presented algorithm performs well, even when we seek solutions for which the matrix D of partial derivatives of the input equations is singular (or close to singular), as already mentioned in Section IV-D. Fig. 7(g) and (h) show isocontours of the determinant of D for configurations "a" and "b," overlaid with the curves in Fig. 7(e) and (f), respectively. The white areas correspond to points where this determinant is less than 10^{-8}. One can verify that, using the Newton–Raphson routines of MAPLE, for example, it is impossible to compute the two solutions where the curves in Fig. 7(f) are tangent, precisely because they lie inside a region of near-singularitiness of D. The situation is worse in the cyclohexane molecule, since in that case the Jacobian is null for all values of $r_{1,4}$, $r_{2,5}$, and $r_{3,6}$. Despite this, though, our algorithm is able to bound all solutions at the desired accuracy, as shown above.

B. All Lines Tangent to Four Spheres

Given four spheres of radii $d_1, \ldots, d_4$ in $\mathbb{R}^3$, with their centers located in $\mathbf{p}_1, \ldots, \mathbf{p}_4$, we want to find their common tangent lines. Equivalently, the problem can be stated as finding all possible lines that keep the prescribed distances $d_1, \ldots, d_4$ to the four points $\mathbf{p}_1, \ldots, \mathbf{p}_4$. This problem was first formulated by Larman [43], and later discussed by Karger [44] and Verschelde [45], and finds several applications in computer graphics and computational geometry, including visibility computations with moving viewpoints [46], computing the smallest enclosing cylinders of point sets [47] and placement problems in geometric modeling [48]. It has been proved that there are at most 12 discrete solutions, and that this bound is tight. A method to find them has recently been given by MacDonald *et al.* [49], who formulate it as a system of two algebraic equations, a cubic and a quartic, in the involved point and vector coordinates. After elimination, this yields a 12th-degree univariate resultant that must be numerically solved. As an alternative, one can arrive at the following coordinate-free formulation, directly tackleable with the presented constraint solver.

First, we characterize the tangent line l by two points on it, say $\mathbf{p}_5$ and $\mathbf{p}_6$, placed a unit distance apart, such that $\mathbf{p}_5$ is the tangent point of l with the first sphere (see Fig. 9). With this, and using the right triangle $\mathbf{p}_1\mathbf{p}_5\mathbf{p}_6$, we directly see that $r_{1,6} = d_1^2 + 1$. Finally, we state the following distance constraints among all

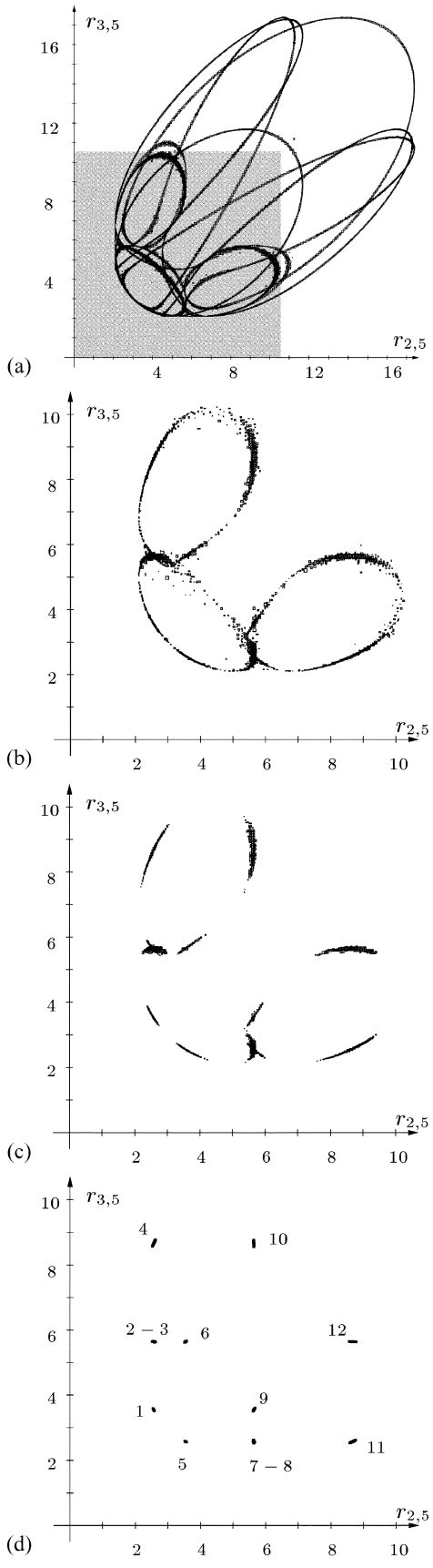


Fig. 10. Output for the second test case. Lengths of the axes in (b), (c), and (d) correspond to the shaded area in (a). Clusters in (d) correspond to the same-labeled ones in Fig. 11.

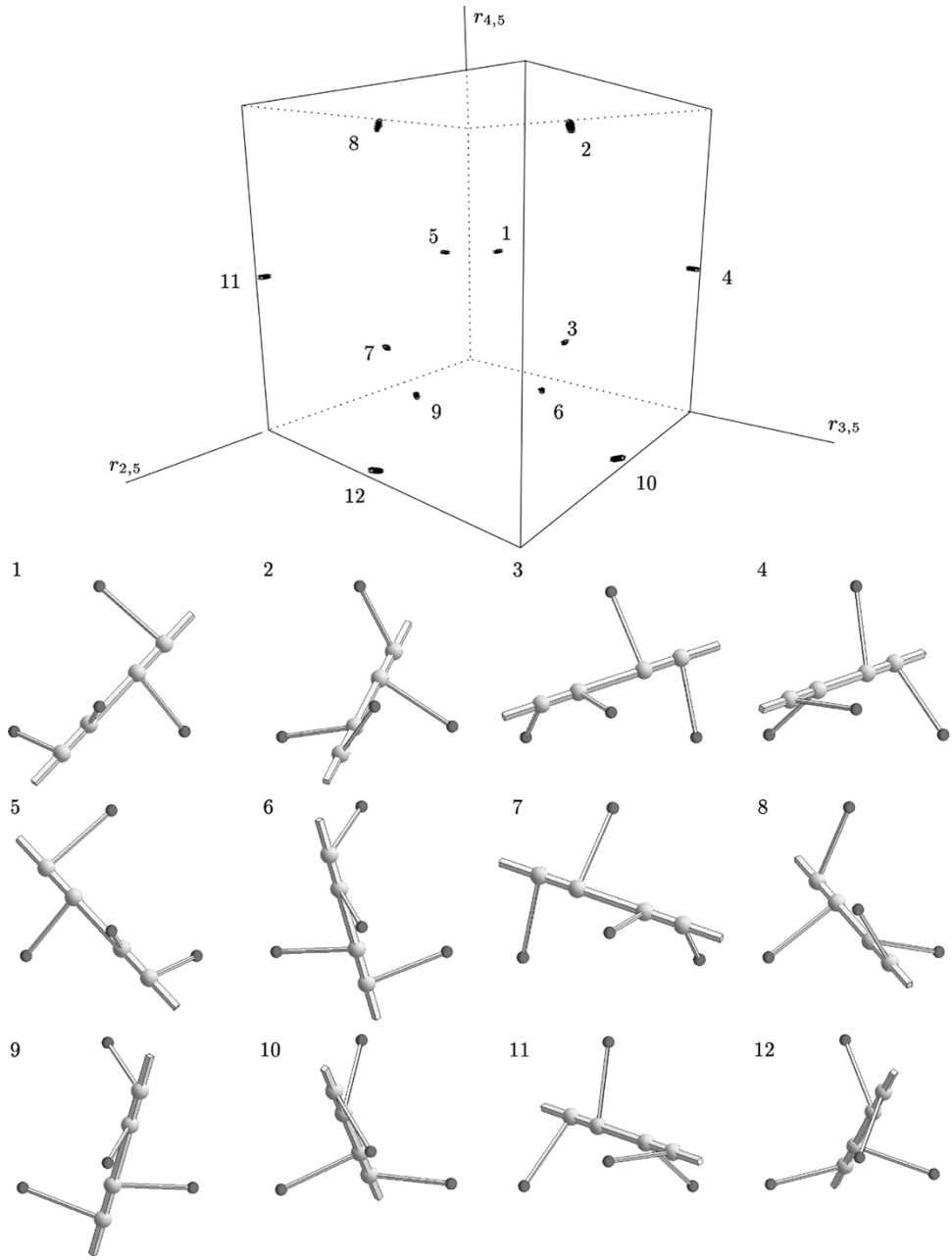


Fig. 11. Top: clusters of solution boxes found in the second test case. The shown bounding box is defined by the intervals $r_{2,5} = r_{3,5} = r_{4,5} = [2.5, 8.8]$. Bottom: the corresponding 12 lines that keep the specified distances to the given four red points. Each configuration corresponds to the same numbered cluster in the plot above.

points in $S = \{\mathbf{p}_1, \ldots, \mathbf{p}_6\}$, defining a redundant system of ten equations in six unknowns.

- C1) Three constraints of the form of (5) to force that the distance from each of $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ to l be d_2, d_3, and d_4, respectively.
- C2) The two Cayley–Menger equations of five points $\Xi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5) = 0$, and $\Xi(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_6) = 0$, each involving three unknowns.
- C3) The remaining four Cayley–Menger equations of five points of S, three involving four unknowns, and one involving six unknowns.
- C4) The unique Cayley–Menger equation of the six points in S, with six unknowns.

One can now use the solver to treat them all together, but it is illustrative to successively apply it to larger subsets of these equations instead, and see the outputs. Let us study the case where all intercenter distances are $\sqrt{8}$, and all radii are 1.45, for which 12 solutions exist [49]. If we start by setting $\sigma = 0.1$ and we just consider the five constraints in C1 and C2, we obtain the 1-D continuum of solutions depicted in Fig. 10(a). The continuum disappears when we solve C1, C2, and C3 together, as seen in Fig. 10(b), giving rise to very large clusters of solution boxes. These can be further reduced if the last constraint C4 is taken into account, to get the box clusters in Fig. 10(c). At this point, we have exhausted all possible distance constraints between the selected points, and we cannot further reduce the clusters, unless we ask a higher accuracy from the solver. If we do this, by setting $\sigma = 0.01$, we get the small clusters in Fig. 10(d), each corresponding to one of the 12 solutions of the problem. Actually, the two pairs of clusters 2–3 and 7–8 appear overlaid, but

TABLE II
ALGORITHM'S PERFORMANCE ON THE "TANGENTS TO FOUR SPHERES" PROBLEM

	$\sigma = 0.1$	$\sigma = 0.01$
t (sec.)	315	385
b_p	7875	14579
n_s	870	813

they can be seen separated if we plot them on the 3-D space defined by $r_{2,5}$, $r_{3,5}$, and $r_{4,5}$. This plot, together with the line configuration corresponding to each cluster, is shown in Fig. 11.

Table II gives the values of t, b_p, and n_s for the last two experiments. The $\sigma = 0.1$ and $\sigma = 0.01$ columns correspond, respectively, to the computation of Fig. 10(c) and (d). Both experiments have been done with $\rho = 0.99$. We note that the time to compute the solutions does not increase substantially, despite the fact that σ has been decreased by one order of magnitude in the second experiment. Furthermore, although a higher number of boxes is processed for $\sigma = 0.01$, the final number of solution boxes remains practically the same as for $\sigma = 0.1$. This is not casual. One can see that, from a certain point, after asking higher and higher accuracies from the solver, the number of boxes around each solution point will practically remain constant. The avoidance of this phenomenon, also associated with the cluster problem observed in [36], constitutes part of our current research.

VI. CONCLUSIONS

We have presented a general solver for systems of kinematic and geometric constraints. Target applications are those where a system of constraints on the relative positions of a set of objects must be solved, such as those arising in the position analysis of serial and parallel robots, the contact analysis of polyhedral models, or the automatic generation of constraint-specified designs or assemblies.

A special emphasis has been put on finding a good combination of algebraization and resolution techniques. To this end, the use of standard coordinate-free constraints derived from the theory of distance geometry, combined with a novel branch-and-prune technique to solve them, has proved efficient, yielding a solver that is conceptually simple and easy to implement. The solver is also complete, in the sense that it loses no solutions, and is able to discretize entire continua of solutions if they exist.

According to our experiments, the addition of redundant constraints speeds up, in general, the resolution process, and reduces the number of final boxes delivered. Although not illustrated by the presented examples, the addition of redundant variables, on the contrary, usually introduces a tradeoff between the number of final boxes and execution times. As the number of redundant variables is increased, the solver needs longer execution times, but, in return, it obtains a lower number of final boxes.

The algorithm as it stands leaves many choices open, as is usually the case in constraint-based search (variable ordering, constraint selection, redundancy dosage, etc.). This offers a range of possibilities to speed up the resolution process, which we will tackle in future research by devising good heuristics for the choice points above.

REFERENCES

[1] J. M. Porta, L. Ros, F. Thomas, and C. Torras, "A branch-and-prune algorithm for solving systems of distance constraints," in *IEEE Int. Conf. Robot. Autom.*, vol. 1, Taipei, Taiwan, R.O.C., Sep. 2003, pp. 342–348.

[2] I. Fudos and C. Hoffmann, "A graph-constructive approach to solving systems of geometric constraints," *ACM Trans. Graphics*, vol. 16, no. 2, pp. 179–216, 1997.

[3] J. Nielsen and B. Roth, "Formulation and solution for the direct and inverse kinematics problems for mechanisms and mechatronic systems," in *Proc. NATO Adv. Study Inst. Computat. Methods Mechanisms*, vol. I, J. Angeles and E. Zakhariev, Eds., Jun. 1997, pp. 233–252.

[4] ——, "On the kinematic analysis of robotic mechanisms," *Int. J. Robot. Res.*, vol. 18, no. 12, pp. 1147–1160, 1999.

[5] C. H. Hoffmann and B. Yuan, "On spatial constraints solving approaches," in *Automated Deduction in Geometry: Third International Workshop*, ser. Lecture Notes in Computer Science. New York: Springer, 2000, vol. 2061.

[6] I. Z. Emiris and B. Mourrain, "Computer algebra methods for studying and computing molecular conformations," *Algorithmica*, no. 25, pp. 372–402, 1999.

[7] D. Manocha and J. Canny, "Efficient inverse kinematics for general 6r manipulators," *IEEE Trans. Robot. Autom.*, vol. 10, pp. 648–657, Jun. 1994.

[8] K. Sridharan, "Computing two penetration measures for curved 2d objects," *Inf. Process. Lett.*, vol. 72, pp. 143–148, 1999.

[9] C. Bajaj and M.-S. Kim, "Generation of configuration space obstacles: Moving algebraic surfaces," *Int. J. Robot. Res.*, vol. 9, no. 1, pp. 92–112, 1990.

[10] D. Kapur and T. Saxena, "Sparsity considerations in Dixon resultants," in *Proc. Annu. ACM Symp. Theory of Computing*, 1996, pp. 184–191.

[11] I. Emiris and J. Canny, "Efficient incremental algorithms for the sparse resultant and the mixed volume," *J. Symbolic Comput.*, vol. 20, no. 2, pp. 117–149, 1995.

[12] J. Canny and I. Emiris, "A subdivision-based algorithm for the sparse resultant," *J. ACM*, vol. 47, no. 3, pp. 417–451, 2000.

[13] B. Roth and F. Freudenstein, "Synthesis of path-generating mechanisms by numerical methods," *ASME J. Eng. Ind.*, vol. 85, pp. 298–307, 1963.

[14] C. B. Garcia and T. Y. Li, "On the number of solutions to polynomial systems of equations," *SIAM J. Numer. Anal.*, vol. 17, pp. 540–546, 1980.

[15] C. B. Garcia and W. I. Zangwill, *Pathways to Solutions, Fixed Points, and Equilibria*. Upper Saddle River, NJ: Prentice-Hall, 1981.

[16] A. P. Morgan, "A homotopy for solving polynomial systems," *Appl. Math. Computat.*, vol. 18, pp. 87–92, 1986.

[17] T. Y. Li, T. Sauer, and J. A. York, "The cheater's homotopy: An efficient procedure for solving systems of polynomial equations," *SIAM J. Numer. Anal.*, vol. 18, no. 2, pp. 173–177, 1988.

[18] C. Wampler, A. Morgan, and A. Sommese, "Numerical continuation methods for solving polynomial systems arising in kinematics," *ASME J. Mech. Design*, vol. 112, pp. 59–68, 1990.

[19] L.-W. Tsai and A. Morgan, "Solving the kinematics of the most general six- and five-degree-of-freedom manipulators by continuation methods," *ASME J. Mechan., Trans., Autom. Design*, vol. 107, pp. 189–200, 1985.

[20] M. Raghavan, "The Stewart platform of general geometry has 40 configurations," *ASME J. Mech. Design*, vol. 115, pp. 277–282, 1993.

[21] C. Wampler, A. Morgan, and A. Sommese, "Complete solution of the nine-point path synthesis problem for four-bar linkages," *J. Mech. Design*, vol. 114, pp. 153–159, 1992.

[22] E. Sherbrooke and N. Patrikalakis, "Computation of the solutions of nonlinear polynomial systems," *Comput. Aided Geom. Design*, vol. 10, no. 5, pp. 379–405, 1993.

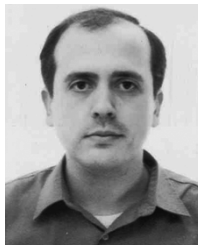
[23] R. S. Rao, A. Asaithambi, and S. K. Agrawal, "Inverse kinematic solution of robot manipulators using interval analysis," *ASME J. Mech. Design*, vol. 120, pp. 147–150, 1998.

[24] O. Didrit, M. Petitot, and E. Walter, "Guaranteed solution of direct kinematic problems for general configurations of parallel manipulators," *IEEE Trans. Robot. Autom.*, vol. 14, pp. 259–266, Apr. 1998.

[25] A. Castellet and F. Thomas, "An algorithm for the solution of inverse kinematics problems based on an interval method," in *Advances in Robot Kinematics*, M. Husty and J. Lenarcic, Eds. Norwell, MA: Kluwer, 1998, pp. 393–403.

[26] J.-P. Merlet, "A formal numerical approach to determine the presence of singularity within the workspace of a parallel robot," in *Proc. 2nd Workshop Computat. Kinematics*, Seoul, South Korea, May 2001, pp. 167–176.

[27] ——, "An improved design algorithm based on interval analysis for parallel manipulator with specified workspace," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, Seoul, South Korea, May 2001, pp. 1289–1294.
[28] C. Bombín, L. Ros, and F. Thomas, "A concise Bézier clipping technique for solving inverse kinematics problems," in *Advances in Robot Kinematics*, J. Lenarcic and M. Stanisic, Eds. Norwell, MA: Kluwer, 2000, pp. 53–61.
[29] J. M. Porta, L. Ros, F. Thomas, and C. Torras, "Solving multi-loop linkages by iterating 2d clippings," in *Advances in Robot Kinematics*, F. Thomas and J. Lenarcic, Eds. Norwell, MA: Kluwer, 2002, pp. 255–264.
[30] A. Cayley, "A theorem in the geometry of position," *Cambridge Math. J.*, vol. II, pp. 267–271, 1841.
[31] K. Menger, "New foundation for Euclidean geometry," *Amer. J. Math.*, no. 53, pp. 721–745, 1931.
[32] L. Blumenthal, *Theory and Applications of Distance Geometry*. Oxford, U.K.: Oxford Univ. Press, 1953.
[33] A. Rikun, "A convex envelope formula for multilinear functions," *J. Global Optim.*, vol. 10, pp. 425–437, 1997.
[34] P. V. Hentenryck, D. McAllester, and D. Kapur, "Three cuts for accelerating interval propagation," MIT, Artif. Intell. Lab., Cambridge, MA, Tech. Rep., 1995.
[35] ——, "Solving polynomial systems using a branch and prune approach," *SIAM J. Numer. Anal.*, vol. 34, no. 2, pp. 797–827, 1997.
[36] A. Morgan and V. Shapiro, "Box-bisection for solving second-degree systems and the problem of clustering," *ACM Trans. Math. Software*, vol. 13, no. 2, pp. 152–167, 1987.
[37] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd ed. New York: McGraw-Hill, 1978.
[38] D. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[39] M. Griffis and J. Duffy, "A forward displacement analysis of a class of Stewart platforms," *J. Robot. Syst.*, vol. 6, pp. 703–720, 1989.
[40] P. Nanua, K. Waldron, and V. Murthy, "Direct kinematic solution of a Stewart platform," *IEEE Trans. Robot. Autom.*, vol. 6, pp. 438–444, Aug. 1990.
[41] F. Thomas, E. Ottaviano, L. Ros, and M. Ceccarelli, "Coordinate-free formulation of a 3-2-1 wire-based tracking device based on Cayley–Menger determinants," in *IEEE Int. Conf. Robot. Autom.*, vol. 1, Taipei, Taiwan, R.O.C., May 2003, pp. 355–361.
[42] R. Bricard, "Mémoire sur la théorie de l'octaèdre articulé" (in French), *J. Math. Pures et Appliquées*, no. 3, pp. 113–148, 1897.
[43] D. Larman, "Problem posed in the problem session of the DIMACS workshop on arrangements," Rutgers Univ., New Brunswick, NJ, 1990.
[44] A. Karger, "Classical geometry and computers," *J. Geom. Graphics*, vol. 2, no. 1, pp. 7–15, 1998.
[45] J. Verschelde, "Polynomial homotopies for dense, sparse and determinantal systems," Dept. Math., Michigan State Univ., East Lansing, MI, Tech. Rep. 1999-041, 1999.
[46] O. Devillers, V. Dujmović, H. Everett, X. Goaoc, S. Lazard, H.-S. Na, and S. Petitjean, "The expected number of 3D visibility events is linear," INRIA, Rennes, France, Tech. Rep. 4671, Dec. 2002.
[47] P. Agarwal, B. Aronov, and M. Sharir, "Line transversals of balls and smallest enclosing cylinders in three dimensions," *Discr. Computat. Geom.*, no. 21, pp. 373–388, 1999.
[48] C. Durand, "Symbolic and numerical techniques for constraint solving," Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1998.
[49] I. Macdonald, J. Pach, and T. Theobald, "Common tangents to four unit balls in R^3," *Discr. Computat. Geom.*, vol. 26, no. 1, pp. 1–17, 2001.



Josep M. Porta received the Engineer degree in computer science in 1994 and the Ph.D. degree in artificial intelligence in 2001, both from the Technical University of Catalonia, Catalonia, Spain.

He performed research in legged robots, machine learning, vision-based methods for autonomous robot localization and computational kinematics. Currently, he holds a Postdoc position at the Institut de Robòtica i Informàtica Industrial, Spanish Scientific Research Council, Barcelona, Spain.

Lluís Ros received the mechanical engineering degree in 1992, and the Ph.D. degree (with honors) in industrial engineering in 2000, both from the Technical University of Catalonia, Catalonia, Spain.

From 1993 to 1996 he worked with the Control of Resources Group, Institut de Cibernètica, Barcelona, Spain, involved in the application of constraint logic programming to the control of electric and water networks. Since August 2000, he has been a Research Scientist with the Institut de Robòtica i Informàtica Industrial, Spanish Scientific Research Council, Barcelona, Spain. His research interests include geometry and kinematics, with applications to robotics, computer graphics, and machine vision.

Federico Thomas received the B.Sc. degree in telecommunications engineering in 1984, and the Ph.D. degree (with honors) in computer science in 1988, both from the Technical University of Catalonia, Catalonia, Spain.

Since March 1990, he has been a Research Scientist with the Institut de Robòtica i Informàtica Industrial, Spanish High Council for Scientific Research, Barcelona, Spain. His research interests are in geometry and kinematics, with applications to robotics, computer graphics, and machine vision.

Carme Torras received the M.Sc. degrees in mathematics and computer science from the University of Barcelona, Barcelona, Spain, and the University of Massachusetts, Amherst, respectively, and the Ph.D. degree in computer science from the Technical University of Catalonia, Catalonia, Spain.

She is currently a Research Professor with the Spanish Scientific Research Council (CSIC), Barcelona, Spain. She has published four books and more than 100 papers in the areas of robot kinematics, geometric reasoning, computer vision, and neurocomputing. She has been local project leader of several European projects, such as "Planning RObot Motion" (PROMotion), "Robot Control based on Neural Network Systems" (CONNY), "Self-organization and Analogical Modeling using Subsymbolic Computing" (SUBSYM), and "Behavioral Learning: Sensing and Acting" (B-LEARN).