Box Approximations of Planar Linkage Configuration Spaces

J. M. Porta, L. Ros, T. Creemers, and F. Thomas,

Institut de Robòtica i Informàtica Industrial (UPC-CSIC)

Llorens Artigas 4-6, 08028 Barcelona

E-mails: {porta,ros,creemers,thomas}@iri.upc.edu

January 10, 2008

Abstract

This paper presents a numerical method able to compute all possible configurations of planar linkages. The procedure is applicable to rigid linkages (i.e., those that can only adopt a finite number of configurations) and to mobile ones (i.e., those that exhibit a continuum of possible configurations). The method is based on the fact that this problem can be reduced to finding the roots of a polynomial system of linear, quadratic, and hyperbolic equations, which is here tackled with a new strategy exploiting its structure. The method is conceptually simple and easy to implement, yet it provides solutions of the desired accuracy in short computation times. Experiments are included which show its performance on the double butterfly linkage and on larger linkages formed by the concatenation of basic patterns.

1 Introduction

A planar linkage is a set of rigid bodies, also called *links*, pairwise articulated through revolute or slider *joints*, all lying in a plane. A linkage *configuration* is an assignment of positions and orientations to all links that respects the kinematic constraints imposed by all joints. As it is well known, the *configuration space* of a linkage —the set of all possible configurations— corresponds to the solution set of a system of polynomial equations, and thus forms an algebraic variety. This paper presents a numerical method able to approximate this variety at any desired resolution, irrespective of whether it contains a finite number

of isolated points (corresponding to rigid configurations) or higher-dimensional connected components (corresponding to finite motions of the linkage).

Many problems translate into the above one, or require an efficient module able to solve it. For instance, in robotics this problem arises when solving the forward kinematics of parallel manipulators [13], when planning the coordinated manipulation of an object or the locomotion of a reconfigurable robot [26], or, as recently shown, in simultaneous localization and map-building [6]. The problem also appears in other domains, such as in the simulation and control of complex deployable structures [8], the theoretical study of rigidity [1], or the conformational analysis of molecular structures [22]. The common denominator in all cases is the existence of one or more kinematic loops defining a linkage, whose configurations must eventually be sought for.

Whereas specific methods for many linkages abound, a few recent methods are already *universal*, being able to manage arbitrary planar mechanisms. For example, Dhingra used reduced Gröbner-Bases and Sylvester's elimination to obtain a simple polynomial condition describing the solution set [4]. Nielsen and Roth also gave an elimination-based method that uses Dixon's resultant to derive the lowest degree polynomial of the algebraic system under study [17]. This technique was later improved by Wampler [25], who used a complex-plane formulation to reduce the size of the final eigenvalue problem by half. The problem can also be tackled using general continuation-based solvers like [23], that start with a system whose solutions are known, and then transform it gradually into the system whose solutions are sought, while tracking all solution paths along the way. In general, it can be said that, while elimination techniques tend to be faster and acceptably accurate when the number of roots is moderate, continuation methods seem more efficient and accurate when this number is large.

Although the previous strategies properly manage configuration spaces with isolated points, it is unclear how they could be applied to deal with higher-dimensional components. While recent continuation methods are able to compute the irreducible decomposition of the solution variety [21], informing on the number of connected components and their degree, to the best of our knowledge, they can only provide sample-based approximations for each component. Contrarily, the method herein presented is able to return *complete* approximations (i.e., approximations that include all solution points and not just samples) of all the solution components, independently of their dimension. This approximation comes in the form of a collection of boxes, not larger than a user-defined size, that fully encloses the solution variety. Similar approximations can be derived using interval-based solvers. 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 polytope approximations of the solution set. To our knowledge, the first applications of the Hansen algorithm in this field were due to Rao et al. [19] and Didrit et al. [5], 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 [2], 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 [10, 14, 15]. Polytope-based techniques, on the other hand, were developed in the early nineties by Sherbrooke and Patrikalakis in the context of constraint-based CAD [20]. These exploit the convex-hull and subdivision properties of Bernstein polynomials, which avoid the computation of derivatives while maintaining the quadratic convergence of the Hansen algorithm. The method we present here can be seen as part of the latter family. However, our method is conceptually simpler and easier to implement than general polytope-based methods, yet it provides solutions at the desired accuracy in shorter computation times.

The rest of the paper is organized as follows. Section 2 starts by reviewing a standard way to derive the cycle equations of a planar linkage. The strategy used to solve them is then presented in Section 3, followed by experimental results showing its performance and convergence order in Section 4. The treatment of slider joints is then explained in Section 5 and, finally, Section 6 summarizes the main contributions of this work.

2 Formulating the cycle equations

To ease the explanations, we will start by considering linkages only containing revolute joints, leaving the extension to the general case for Section 5 below. Also, for the purpose of this paper, a link will either be a single bar, or multiple bars forming a rigid compound.

To obtain the kinematic equations of a planar linkage, we follow the same formulation used in [17], which references the rotation angles of all bars to a fixed, ground coordinate system. With this, every angle θ_i assigned to a bar b_i defines a unit vector $\mathbf{u}_i = (\cos(\theta_i), \sin(\theta_i))$ that gives the orientation of the

bar. We then consider a graph containing a node for each link, and an edge connecting two links if they are sharing a joint. By traversing a cycle $\mathcal{C}$ of this graph, it must hold that

$$\sum_{b_i \in \mathcal{C}} \lambda(i, \mathcal{C}) \cdot l_i \cdot \mathbf{u}_i = 0, \tag{1}$$

where the sum spans all bars b_i found around $\mathcal{C}$, l_i is the length of the ith bar, and $\lambda(i, \mathcal{C})$ is $+1$ or -1 depending on whether $\mathbf{u}_i$ has the same or opposite orientation than the cycle. This vector sum yields two scalar equations of the form

$$\sum_{b_i \in \mathcal{C}} \lambda(i, \mathcal{C}) \cdot l_i \cdot \cos(\theta_i) = 0, \tag{2}$$

$$\sum_{b_i \in \mathcal{C}} \lambda(i, \mathcal{C}) \cdot l_i \cdot \sin(\theta_i) = 0, \tag{3}$$

and, by collecting all of these for a maximal set of independent cycles of the graph [3], we finally get a set of necessary and sufficient conditions describing the valid configurations of the linkage.

To illustrate the process, and to facilitate the comparison with previous work, we consider the same example as in [17] and [25], a double butterfly linkage, which is the only one of the eight-bar linkages that does not contain a four-bar loop (Fig. 1). Using Laman's theorem [9], it can be shown that this

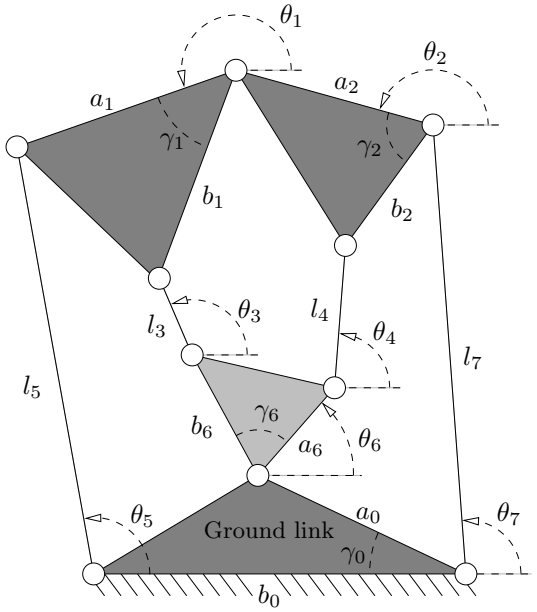


Figure 1: The double butterfly linkage.

mechanism moves with one internal degree of freedom, and that it becomes rigid if the orientation of one more link is fixed, having up to eighteen assembly modes in this case [24]. On this mechanism, we select the three independent cycles that leave the ground link via link 7, and return via links 4, 5, and 3, respectively, to get the following equations

$$\begin{aligned}
l_7\ \cos(\theta_7) + b_2\ \cos(\theta_2 + \gamma_2) - l_4\ \cos(\theta_4) - a_6\ \cos(\theta_6) + a_0\ \cos(\gamma_0) &= 0,\\
l_7\ \sin(\theta_7) + b_2\ \sin(\theta_2 + \gamma_2) - l_4\ \sin(\theta_4) - a_6\ \sin(\theta_6) - a_0\ \sin(\gamma_0) &= 0,\\
l_7\ \cos(\theta_7) + a_2\ \cos(\theta_2) + a_1\ \cos(\theta_1) - l_5\ \cos(\theta_5) + b_0 &= 0,\\
l_7\ \sin(\theta_7) + a_2\ \sin(\theta_2) + a_1\ \sin(\theta_1) - l_5\ \sin(\theta_5) &= 0,\\
l_7\ \cos(\theta_7) + a_2\ \cos(\theta_2) + b_1\ \cos(\theta_1 + \gamma_1) - l_3\ \cos(\theta_3) - b_6\ \cos(\theta_6 + \gamma_6) + a_0\ \cos(\gamma_0) &= 0,\\
l_7\ \sin(\theta_7) + a_2\ \sin(\theta_2) + b_1\ \sin(\theta_1 + \gamma_1) - l_3\ \sin(\theta_3) - b_6\ \sin(\theta_6 + \gamma_6) - a_0\ \sin(\gamma_0) &= 0.
\end{aligned}$$

It is important to realize that one can always derive a similar system for any planar linkage, and that all of its equations will be linear in the sines and cosines of the unknown angles. (The sines and cosines of the shifted angles can always be appropriately expanded so as to satisfy the previous statement.) Actually, if the linkage has n_l links and n_j joints, its graph will have $n_c = n_j - n_l + 1$ independent cycles [3] and the system will be formed by $m = 2n_c$ trigonometric equations involving $v = n_l - 1$ variables (one angle for each link, except for the ground link, whose orientation is fixed, and used as a reference).

To algebraize this system, we can apply the usual change of variables $x_i = \cos(\theta_i)$, $y_i = \sin(\theta_i)$, and add one *circle* equation $x_i^2 + y_i^2 = 1$ for each angle, ending up with a polynomial system of the form

$$\mathbf{L}(\mathbf{v}) = 0,\ \mathbf{C}(\mathbf{v}) = 0, \tag{4}$$

where $\mathbf{v} = (x_1, y_1, \ldots, x_v, y_v)$ are the newly defined variables, $\mathbf{L}(\mathbf{v}) = (l_1(\mathbf{v}), \ldots, l_m(\mathbf{v}))$ is a block of linear functions in the x_i's and y_i's, and $\mathbf{C}(\mathbf{v}) = (c_1(\mathbf{v}), \ldots, c_v(\mathbf{v}))$ is a block of quadratic functions with $c_i(\mathbf{v}) = x_i^2 + y_i^2 - 1$, $i = 1, \ldots, v$. Finally, note that since all variables are sines or cosines of angles, the *search space* where the solutions of System (4) must be sought for is the set

$$\mathcal{B} = [-1, 1] \times \ldots \times [-1, 1] \subset \mathbb{R}^{2v}.$$

In the text below, any set of this kind—defined by the Cartesian product of $2v$ intervals—will be referred to as a *box* of $\mathbb{R}^{2v}$ and we will write $[x_i^l, x_i^u]$ to denote the interval of a box along dimension i.

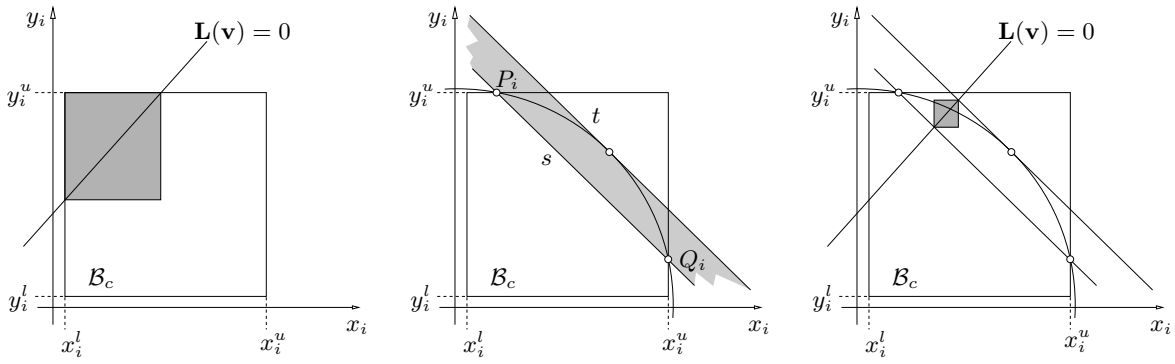


Figure 2: (a) Shrinking $\mathcal{B}_c$ to fit the linear variety $\mathbf{L}(\mathbf{v}) = 0$. (b) Half-planes approximating the circular arc inside $\mathcal{B}_c$. (c) Smallest box enclosing the intersection of $\mathbf{L}(\mathbf{v}) = 0$ with the half-planes in (b).

3 Search strategy

The algorithm starts with the initial box $\mathcal{B}$, and isolates the valid configurations it contains by iterating over two operations, *box shrinking* and *box splitting*. Using box shrinking, portions of $\mathcal{B}$ containing no solution are eliminated by narrowing some of its defining intervals. This process is repeated 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 can no more be "significantly" reduced, in which case it is bisected into two sub-boxes via box splitting—which simply divides the largest interval at its midpoint.

Provided box shrinking is sufficiently efficient, the third case above is symptom that the box contains two or more solution points, with some of them lying close to its walls. Thus, box splitting allows separating such points. To converge to all solutions, the whole process is then repeated for the newly created sub-boxes, and for the sub-boxes recursively created thereafter, until one ends up with a collection of small boxes whose sizes are under a specified size threshold, σ.

Before further precising this process, we will first see how to eliminate portions of a box that cannot contain any solution. Detailed pseudo-code of the whole strategy will be given later, in Section 3.2 below.

3.1 Box Shrinking

When reducing any box $\mathcal{B}_c \subseteq \mathcal{B}$ note first that, since any solution inside $\mathcal{B}_c$ must be in the linear variety $\mathbf{L}(\mathbf{v}) = 0$, we may shrink $\mathcal{B}_c$ to the smallest possible box bounding the portion of this variety falling inside $\mathcal{B}_c$. The limits of this new box along, say, dimension x_i can be easily found by solving the two linear programs

$$\textbf{LP1: Minimize } x_i, \text{ subject to: } \mathbf{L}(\mathbf{v}) = 0, \mathbf{v} \in \mathcal{B}_c,$$

$$\textbf{LP2: Maximize } x_i, \text{ subject to: } \mathbf{L}(\mathbf{v}) = 0, \mathbf{v} \in \mathcal{B}_c,$$

giving, respectively, the new lower and upper bounds for x_i. Fig. 2-(a) illustrates the process on the x_i-y_i plane, in the case that $\mathbf{L}(\mathbf{v}) = 0$ is a straight line.

Note however that $\mathcal{B}_c$ can be further reduced, as the circle equations $\mathbf{C}(\mathbf{v}) = 0$ must also be satisfied. We take them into account as illustrated in Fig. 2-(b). In short, for each angle θ_i, one only needs to consider the gray area bounding the portion of the circle $x_i^2 + y_i^2 = 1$ lying inside the rectangle $\mathcal{B}'_c = [x_i^l, x_i^u] \times [y_i^l, y_i^u]$. This area is the intersection of two half-planes defined by two linear constraints that can be added to the previous linear programs. More formally, for each θ_i, we (a) compute the points P_i and Q_i of intersection of $\mathcal{B}'_c$ with the circle, (b) obtain line s, the secant to the circle through them, and its parallel line t tangent to the circle, and (c) add the two inequalities defining the region between l and t to **LP1** and **LP2**. If we let $R_i = (P_i - Q_i)/2$, these inequalities are simply

$$\mathbf{w}_i \cdot \mathbf{x}_i \geq d_i,$$

$$\mathbf{w}_i \cdot \mathbf{x}_i \leq 1,$$

where $\mathbf{x}_i = (x_i, y_i)$, $d_i = ||R_i||$, and $\mathbf{w}_i = R_i/d_i$. Although more sophisticated bounds for circle arcs could be used, to ease the implementation we just consider this simple one, and we only apply it when $\mathcal{B}'_c$ is fully contained in one quadrant of the x_i-y_i plane.

The effect of using these inequalities in conjunction with $\mathbf{L}(\mathbf{v}) = 0$ is usually a much larger reduction of $\mathcal{B}_c$, as illustrated in Fig. 2-(c). Note also that, altogether, these constraints define a convex polytope bounding the solution space of System (4), i.e., the intersection of the line and the circle in the example of Fig. 2. The smaller $\mathcal{B}_c$, the tighter this polytope approximates the solution space or, in other words, the smaller the error introduced in the circle arc approximations. For small-enough boxes, the error will become negligible and, therefore, the iteration of the above process will reduce $\mathcal{B}_c$ to the smallest box

```
SOLVE-LINKAGE(B, L, C, σ, ρ)
 1: S ← ∅
 2: P ← {B}
 3: while P ≠ ∅ do
 4:    B_c ← EXTRACT(P)
 5:    repeat
 6:       V_p ← VOLUME(B_c)
 7:       SHRINK-BOX(B_c, L, C)
 8:       V_c ← VOLUME(B_c)
 9:    until IS-VOID(B_c) or SIZE(B_c) ≤ σ or V_c/V_p > ρ
10:    if not IS-VOID(B_c) then
11:       if SIZE(B_c) ≤ σ then
12:          S ← S ∪ {B_c}
13:       else
14:          SPLIT-BOX(B_c, B_1, B_2)
15:          P ← P ∪ {B_1, B_2}
16:       end if
17:    end if
18: end while
19: return S
```

Algorithm 1: The top-level search scheme.

bounding the solutions inside it.

3.2 Pseudocode

Algorithm 1 gives the main loop of the process. It receives as input the box $\mathcal{B}$, the lists L and C containing the equations $\mathbf{L}(\mathbf{v}) = 0$ and $\mathbf{C}(\mathbf{v}) = 0$, and two threshold parameters σ and ρ, and it returns as output S, a list of solution boxes. The functions VOLUME($\mathcal{B}$) and SIZE($\mathcal{B}$) compute the volume and the length of the longest side of $\mathcal{B}$, respectively.

Initially, two lists are set up in lines 1 and 2, an empty list S of "solution boxes", and a list P of

SHRINK-BOX($\mathcal{B}, L, C$)

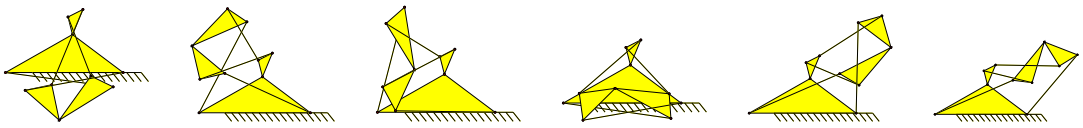
1: $T \leftarrow L$
2: **for all** equations $x_i^2 + y_i^2 - 1 = 0$ in C **do**
3: $\quad \mathcal{B}'_c \leftarrow [x_i^l, x_i^u] \times [y_i^l, y_i^u]$
4: $\quad$ **if** $\mathcal{B}'_c$ is contained in only one quadrant **then**
5: $\quad\quad$ Compute $\mathbf{w}_i$ and d_i (see the text)
6: $\quad\quad$ $T \leftarrow T \cup \{\mathbf{w}_i \cdot \mathbf{x}_i \geq d_i \ , \ \mathbf{w}_i \cdot \mathbf{x}_i \leq 1\}$
7: $\quad$ **end if**
8: **end for**
9: **for** each $i \in \{1, \ldots, v\}$ **do**
10: $\quad x_i^l \leftarrow$ minimize x_i subject to all eqs. in T and $\mathbf{v} \in \mathcal{B}$
11: $\quad x_i^u \leftarrow$ maximize x_i subject to all eqs. in T and $\mathbf{v} \in \mathcal{B}$
12: $\quad y_i^l \leftarrow$ minimize y_i subject to all eqs. in T and $\mathbf{v} \in \mathcal{B}$
13: $\quad y_i^u \leftarrow$ maximize y_i subject to all eqs. in T and $\mathbf{v} \in \mathcal{B}$
14: **end for**

Algorithm 2: The SHRINK-BOX procedure.

"boxes to be processed" containing $\mathcal{B}$. A *while* loop is then executed until P gets empty (lines 3-18), involving the following steps. Line 4 extracts one box from P. Lines 5-9 repeatedly reduce this box as much as possible, via the SHRINK-BOX function, until either the box is an empty set (IS-VOID($\mathcal{B}_c$) is true), or it cannot be significantly reduced ($V_c/V_p > \rho$), or it becomes small enough (SIZE($\mathcal{B}$) $\leq \sigma$). In this last case, the box is considered a solution for the problem. If a box is neither a solution nor it is empty, lines 14 and 15 split it into two sub-boxes and add them to P for further processing.

Notice that this algorithm implicitly explores a binary tree of boxes, the internal nodes being boxes that have been split at some time, and its leaves being either solution or empty boxes. Solution boxes are collected in list S and returned as output in line 19. Clearly, the tree may be explored in either depth-first or breadth-first order, depending on whether line 15 inserts the boxes at the head or tail of P, getting identical output in any case.

The SHRINK-BOX procedure is sketched in Algorithm 2. It receives as input the box $\mathcal{B}$ to shrink, and the lists L and C with the equations $\mathbf{L}(\mathbf{v}) = 0$ and $\mathbf{C}(\mathbf{v}) = 0$. The procedure starts by gathering into



θ_1	[-0.49433, -0.49431]	[1.72769, 1.72771]	[0.55302, 0.55303]	[-2.66941, -2.66940]	[-2.25883, -2.25881]	[0.41349, 0.41351]
θ_2	[0.78081, 0.78083]	[1.13863, 1.13865]	[-0.71932, -0.71930]	[-0.64815, -0.64813]	[2.77060, 2.77062]	[2.44494, 2.44497]
θ_3	[-0.81924, -0.81923]	[2.58872, 2.58873]	[-1.20077, -1.20076]	[-0.40127, -0.40125]	[-0.24062, -0.24061]	[2.10928, 2.10929]
θ_4	[-2.42719, -2.42718]	[-2.66043, -2.66042]	[-1.96316, -1.96315]	[0.56971, 0.56972]	[-0.01687, -0.01686]	[-2.79537, -2.79536]
θ_5	[-0.14221, -0.14220]	[1.31798, 1.31799]	[-0.13237, -0.13236]	[0.41104, 0.41105]	[0.41212, 0.41213]	[1.08683, 1.08684]
θ_7	[-2.96808, -2.96808]	[3.12665, 3.12666]	[-2.96187, -2.96186]	[1.54820, 1.54821]	[0.86915, 0.86916]	[2.71004, 2.71005]

Figure 3: The six real solutions of the double butterfly linkage for $\theta_6 = 1.175$ rad (67.38°), given in radians.

a list T all linear constraints in L (line 1) and all half planes approximating the circle equations in C (lines 2-8). Then, the procedure uses these constraints to reduce every dimension of the box, solving the linear programs in lines 10 to 13, which possibly give tighter bounds for the corresponding intervals.

Observe that if System (4) has a finite number of isolated solutions, Algorithm 1 returns a collection of boxes containing them all, with each solution lying in one, and only one box. If, on the contrary, the solution space is an algebraic variety of dimension one or higher, the returned boxes will form a discrete envelope of the variety. The accuracy of the output can be adjusted at will by using the σ parameter, which fixes an upper limit for the width of the widest side of all returned boxes.

4 Experiments

The algorithm has been implemented in C, and all CPU times will be given for a 2.6 GHz Intel Pentium 4 PC, running under Linux. The linear programs in the SHRINK-BOX function have been solved using the Simplex method implemented in the GLPK package [11].

The first experiment presented below solves the position analysis of the double butterfly linkage when θ_6 is a fixed, known angle, yielding a finite number of isolated solutions. The second one solves the same problem but assuming that θ_6 is a free variable, yielding a 1-dimensional continuum of solutions. While the former case allows comparing the results with those published in [17] and [25], the latter shows the algorithm's performance for problems rarely addressed in the literature. In both cases, we adopt the geometric parameters used in [17] and [25]: $a_0 = 7$, $a_1 = 7$, $a_2 = 5$, $b_0 = 13$, $b_1 = 6$, $b_2 = 3$, $\gamma_0 = 36.87°$,

$\gamma_1 = 22.62°$, $\gamma_2 = 53.13°$, $l_3 = 7$, $l_4 = 9$, $l_5 = 12$, $l_7 = 11$, $a_6 = 3$, $b_6 = 2$, and $\gamma_6 = 36.87°$. The third and fourth experiments assess the scalability of the algorithm using large linkages formed by the concatenation of basic patterns.

4.1 A rigid butterfly

The number of solutions of the double butterfly linkage varies depending on the choice of driving joint and the angle given to it. If we set $\theta_6 = 67.38°$, the number of observed solutions is six [17]. We note that, while continuation and elimination-based methods must filter the solutions among the eighteen possible complex roots, the one given here directly provides the six real ones, shown in Fig. 3. All roots are in accordance with the results in [17, 25].

Due to the nature of the algorithm all solutions are obtained as intervals that bound them, which allows estimating the error with respect to the exact position of the roots. This is equal or less than $0.0013°$ in this case (the width of the longest interval in Fig. 3). The solutions were obtained by running the proposed algorithm with $\sigma = 0.0001$ and $\rho = 0.95$ in 0.3 sec of CPU time, after processing 15 boxes. From them, only the six shown in Fig. 3 were considered as solutions (thus returning the minimum possible number of boxes) and 2 boxes were found to be empty.

In this particular problem, the application of methods based on Dixon's resultant [17, 25] boils down to forming and solving an 18×18 eigenvalue problem, which is likely to be faster than the presented approach. However, it is difficult to verify this point mainly because no statistics are given in this respect in those works, and we have found no publicly available package implementing them. In general, we can say that as the ratio of complex solutions versus real solutions grows, methods based on Dixon's resultants become less efficient while our method is immune to such problem since it directly operates in the domain of the reals.

Moreover, we have checked that our method converges in substantially shorter times than those used by the continuation method in [23, 21], using the implementation available from [7], which spent about 8 seconds of CPU time on the same example, running on the same machine. We remark, though, that we are comparing our algorithm with a general-purpose solver targeted to arbitrary systems of algebraic equations, and that a better performance of our algorithm was to be expected, given the fact that it exploits the specific structure of the obtained equations.

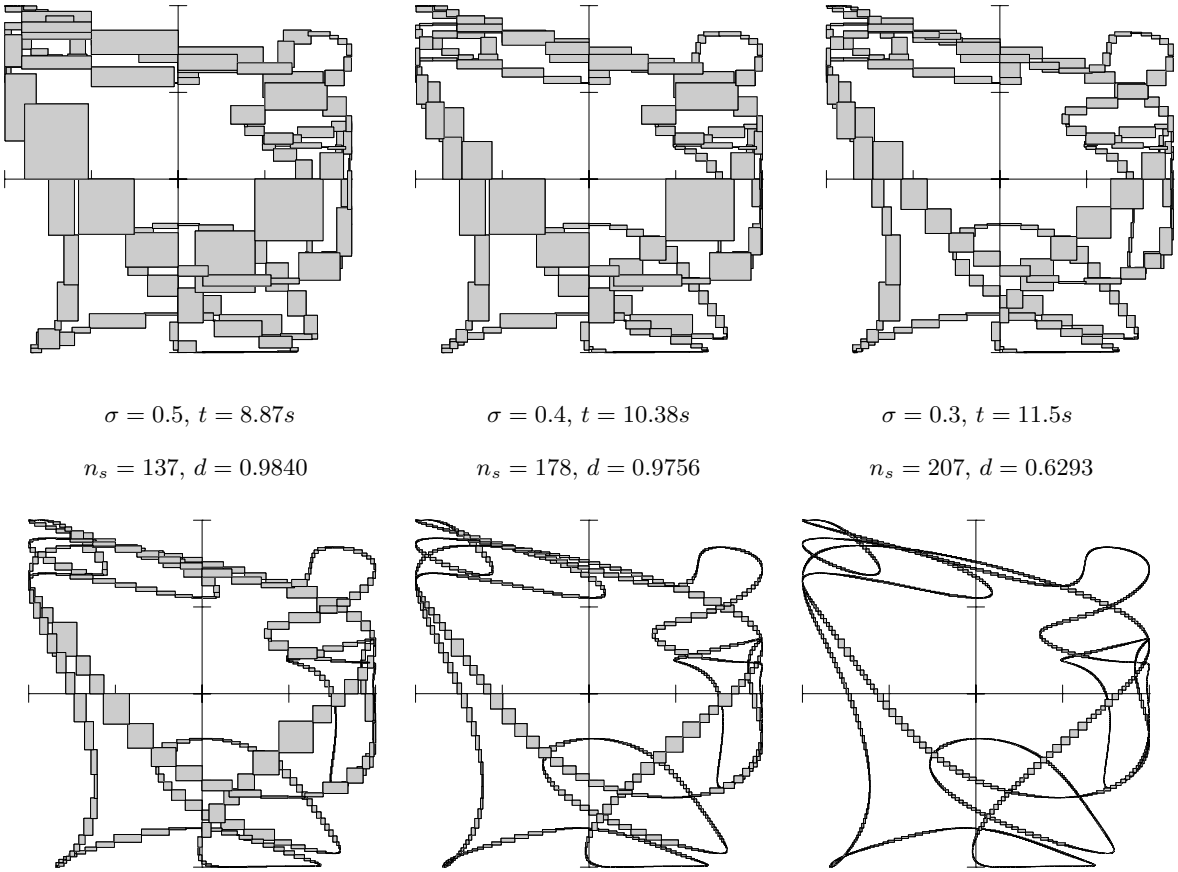


Figure 4: Output boxes at increasing resolution. The horizontal and vertical axes respectively correspond to $\cos(\theta_2)$ and $\cos(\theta_4)$, spanning the range [-1,1] in all cases, with marks separated 0.5 units apart.

4.2 A mobile butterfly

If we now free θ_6, a one dimensional continuum of solutions is obtained. Fig. 4 depicts the projection of the returned boxes onto the $\cos(\theta_2)$-$\cos(\theta_4)$ plane, on six different runs of the algorithm, at decreasing values of the σ parameter. If the algorithm is exploring in breadth-first order, the first five plots can also be interpreted as earlier stages of the run for the last case ($\sigma = 0.05$). In every plot we indicate the σ threshold, the CPU time spent (t), the number of solution boxes returned (n_s), and the diagonal of the largest box (d). The latter serves as an estimation of the maximum distance to the roots, from any point

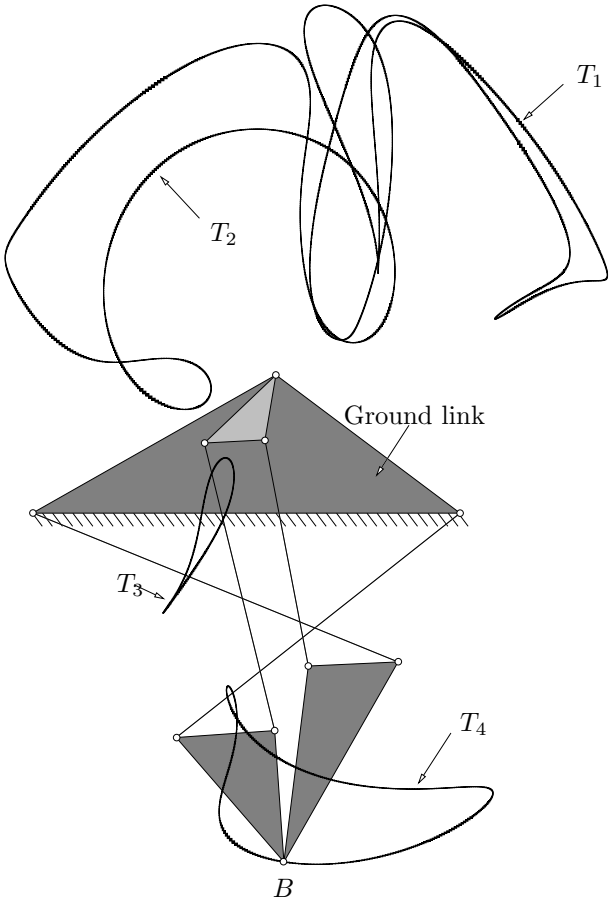


Figure 5: Path followed by point B of the double butterfly linkage (the ground link is rotated 90 degrees to the left with respect to Fig. 1). As observed, B may follow one of four different cyclic trajectories, T_1, T_2, T_3 and T_4, reflecting four (mobile) assembly modes for the mechanism. A sample configuration of the linkage following the fourth mode is also shown overlaid.

inside the boxes. The ρ parameter is set to 0.95 in all runs.

By zooming into the last snapshot on the electronic version of the paper, one can clearly see that the final output is obtained with no clustering, that is, no boxes containing no solutions appear in the neighborhood of the solutions. We note that, although from the plots it seems that the different solution branches cross at many points, these are not true bifurcations of the linkage, as revealed by observing other 3D projections of the same output. Actually, four disjoint closed paths appear, corresponding to the four possible ways to assemble this mobile mechanism.

It is worthwhile noting that, if we wish to visualize the trajectory of any joint J of the linkage, we just need to add the following equation to System (4),

$$(x_J, y_J) = \sum_{b_i \in \mathcal{P}} \lambda(i, \mathcal{P}) \cdot l_i \cdot \mathbf{u}_i \tag{5}$$

where (x_J, y_J) are the unknown coordinates of point J with respect to a reference frame placed on a joint O on the ground link, and the sum is taken over all bars b_i found on a path $\mathcal{P}$ connecting O with J. The returned boxes will then have x_J and y_J as extra dimensions and we need only to plot the ranges for them on a plane to see the motion curve of J. The trajectories of the coupler point B of the double butterfly are shown in Fig. 5 as an example. It is worth mentioning that the analytic form of these trajectories is not trivial, as it can be seen in [18].

4.3 Larger linkages made up of repeated patterns

In order to assess the scalability of the proposed algorithm, experiments were carried out with linkages constructed from the repetitive concatenation of a basic pattern. The first of such studied linkages is the *caterpillar* framework described in [1]. This mechanism is iteratively constructed from *Desargues frameworks* glued together along an edge in a caterpillar fashion (Fig. 6, left). Each such framework can actually be viewed as a 3-RPR planar manipulator [12] with locked actuators (Fig. 6, right). In our experiments, the caterpillar was constructed based on identical 3-RPR patterns with link lengths chosen to yield four discrete configurations of the basic pattern, multiplying the number of solutions of the caterpillar as a whole by four on each repetition of such pattern. Since the position analysis of each 3-RPR pattern is straightforward [12], deriving all configurations for the whole caterpillar is easy. Therefore, this example can also be used to test the reliability of our implementation, because we can compute all its solutions beforehand.

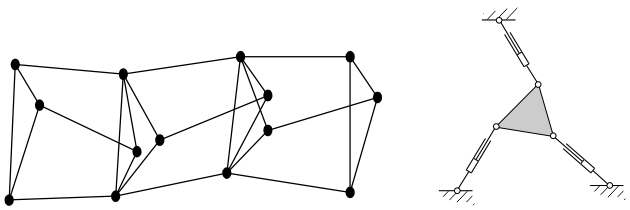


Figure 6: The 3-caterpillar framework (left), and the 3-RPR planar manipulator (right).

length	vars.	eqs.	sols.	bisections	empty boxes	time (s)
1	8	8	4	3	0	0
2	18	19	16	18	3	5
3	28	30	64	88	25	67
4	38	41	256	490	235	630
5	48	52	1024	2395	1372	5013
6	58	63	4096	11023	6928	33582

Table 1: Experimental solver results for the caterpillar construction

Table 1 shows the results of the implemented algorithm for different lengths (repetitions of the basic pattern) of the caterpillar construction. For each length, we give the number of involved variables, equations, found solutions, box bisections performed, empty boxes found, and the execution time. Note that the process time necessarily grows as the simplex tableaus involve more variables and equations, but the increments are reasonable taking into account the exponential increment of the number of solutions. As an example of the results obtained, Fig. 7 shows the 64 solutions obtained for the 3-caterpillar.

In a second series of experiments the repeated pattern is a rigid body consisting of a rectangle and a triangle glued together. The enclosed angle δ of the triangle is decreased slightly on each repetition, yielding the spiral-shaped construction in Fig. 8. Since the whole construction can only be assembled in the given way, this test demonstrates the algorithm's behavior for polynomial systems with a unique solution. One can observe in Table 2 that, as desired, the algorithm never needs to bisect the search space, and that no boxes get discarded, meaning it just reduces the initial box until it approximates the solution to the desired accuracy, without branching along the way.

4.4 Convergence order

The asymptotic performance of a root finding algorithm is normally evaluated by examining its convergence order. An algorithm is said to exhibit a convergence of order r if there exists a constant $k \in (0, 1)$, such that

$$d(\mathbf{x}_{i+1}, \mathbf{x}^*) \leq k \cdot d(\mathbf{x}_i, \mathbf{x}^*)^r,$$

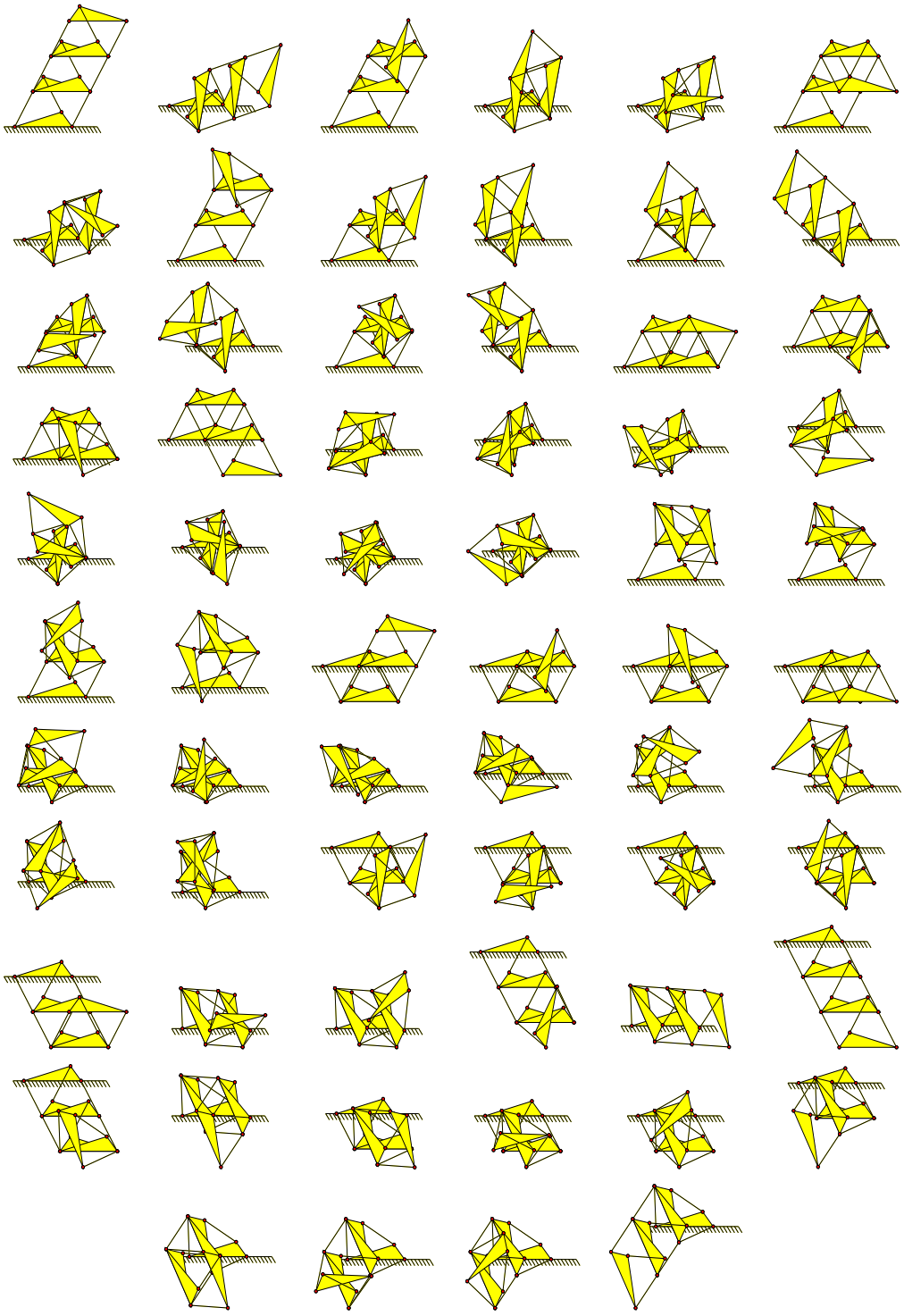


Figure 7: The 64 solutions of the 3-caterpillar

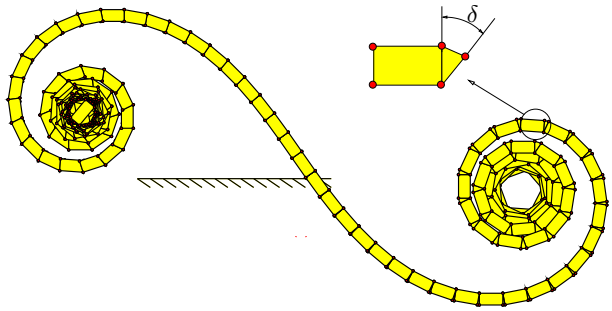


Figure 8: A spiral-shaped construction of 200 repeated patterns

length	vars.	eqs.	sols.	bisections	empty boxes	time (s)
1	2	3	1	0	0	0
2	6	9	1	0	0	0
3	10	15	1	0	0	0
4	14	21	1	0	0	0
5	18	27	1	0	0	0
20	78	117	1	0	0	0
30	118	177	1	0	0	0
40	158	237	1	0	0	1
50	198	297	1	0	0	2
100	398	597	1	0	0	10
200	798	1197	1	0	0	53
500	1998	2997	1	0	0	830

Table 2: Experimental solver results for the spiral construction

where $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are estimations of the exact root $\mathbf{x}^*$ at iterations i and $i+1$, and $d(\mathbf{x}_i, \mathbf{x}^*)$ and $d(\mathbf{x}_{i+1}, \mathbf{x}^*)$ indicate their distance to $\mathbf{x}^*$. The algorithm is said to exhibit linear or quadratic convergence when $r = 1$ or $r = 2$, respectively.

The previous definition is valid for algorithms converging to a single root, and adapting it to our case requires defining $d(\mathbf{x}_i, \mathbf{x}^*)$ and the scope of an iteration. To this end, note that the diagonal of a box is an upper bound of the distance from any point inside that box, to any root in it. Thus, assuming that the search tree explored by Algorithm 1 is traversed in breadth-first order, it seems reasonable to define $d(\mathbf{x}_i, \mathbf{x}^*)$ as the longest diagonal among all boxes in list P, i.e., the boxes waiting to be processed. An iteration will then be defined as the application of lines 4-14 to all boxes in the ith level of such tree.

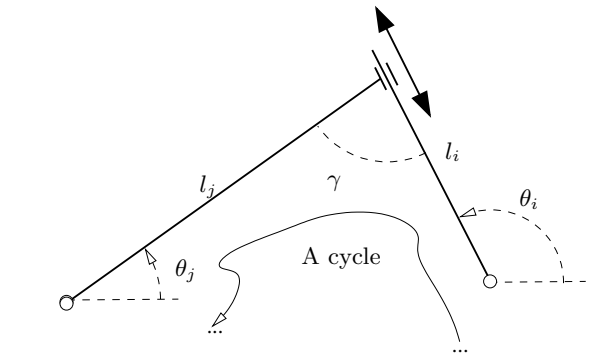
Measuring the performance in this way, we have empirically found that the algorithm converges quadratically to the roots if these are a finite number of isolated points, or linearly, if they form a one-dimensional algebraic variety. In the former case, the convergence order is the same as that of fast single-root-finding procedures, like e.g. the Newton-Raphson method. Although the performance seems worse in the latter case, we should mention that a linear rate is the best one could expect. Consider for example the behavior of an optimal shrink-and-split algorithm discretizing a line (the simplest possible one-dimensional variety). At each iteration, any box $\mathcal{B}_c$ adjusted to the line would be split into two half-boxes, and then, ideally, these would be shrunk to fit the line again. Note that, in such perfect behavior, $d(\mathbf{x}_i, \mathbf{x}^*)$ would decrease by half at each iteration, yielding the linear convergence order we observe.

5 Dealing with slider joints

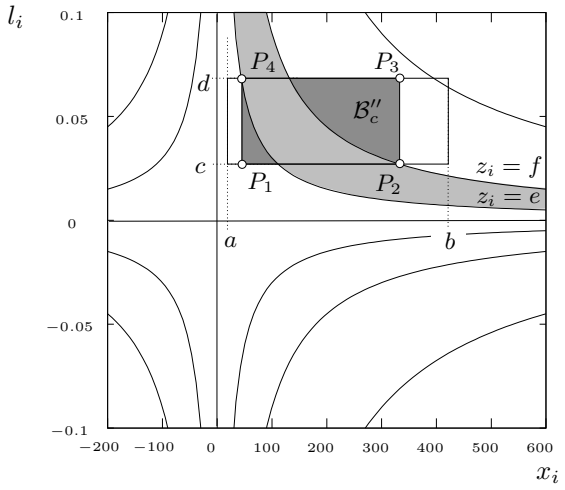
If the linkage has one or more slider joints, the method must be slightly modified. Consider that a slider joint is acting between, say, link i and link j, as depicted in Fig. 9 (a). This fixes the angle γ between the two links, only allowing a translation of one link with respect to the other. Then, Equations (2) and (3) will look like

$$\ldots + l_i \cdot \cos(\theta_i) + l_j \cdot \cos(\theta_j) + \ldots = 0,$$
$$\ldots + l_i \cdot \sin(\theta_i) + l_j \cdot \sin(\theta_j) + \ldots = 0,$$

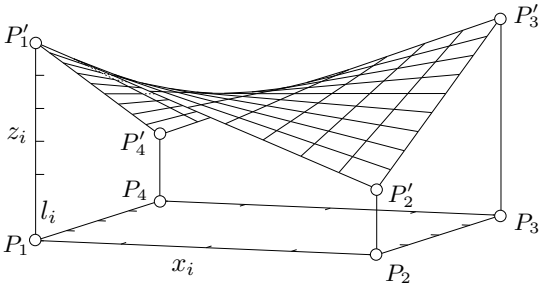
for any cycle traversing links i and j. Since $\theta_j = \theta_i - \gamma$, one of the angles can be eliminated, and only θ_i and l_i are true variables in the previous terms. Note that after performing the substitutions $x_i = \sin(\theta_i)$,



(a)



(b)



(c)

Figure 9: Dealing with slider joints. (a) A slider joint between links i and j. (b) Adjusting a box to fit a hyperbolic paraboloid $z_i = x_i l_i$. (c) The tetrahedron defined by the P'_i's is a convex bound of this surface inside $\mathcal{B}''_c$.

$y_i = \cos(\theta_i)$, these equations will contain bilinear products of the form $l_i x_i$ and $l_i y_i$, which cannot be dealt with by the proposed algorithm. To obtain a whole block of linear equations again, we may simply substitute such terms by dummy variables, say z_i and t_i, and add the hyperbolic equations $z_i = l_i x_i$ and $t_i = l_i y_i$ to System (4). With this, the problem reduces to deriving linear-based bounds for these equations, in a similar way as done for the circle equations. In other words, if we consider one of these equations, say $z_i = l_i x_i$, and we know that its variables can take values inside the ranges $x_i \in [a, b]$, $l_i \in [c, d]$, and $z_i \in [e, f]$, all we need is a collection of half-planes tightly delimiting the set of points that satisfy $z_i = l_i x_i$ inside the box $\mathcal{B}'_c = [a, b] \times [c, d] \times [e, f]$.

To achieve this, note that $z_i = l_i x_i$ is the implicit equation of a hyperbolic paraboloid, whose isocontour lines of constant z_i look as depicted in Fig. 9 (b). Initially, $\mathcal{B}'_c$ can be adjusted to this surface, by easily reducing the ranges of x_i and l_i so that they are compatible with the range for z_i, yielding a new box $\mathcal{B}''_c$, delimited by points P_1, P_2, P_3 and P_4. Let P'_1, P'_2, P'_3 and P'_4 be the points of the hyperbolic paraboloid that project onto P_1, P_2, P_3 and P_4. Note that the line segments $P'_1 - P'_2$, $P'_2 - P'_3$, $P'_3 - P'_4$, and $P'_4 - P'_1$, are all part of the two families of lines of this ruled surface. With the help of Fig. 9 (c), it is easy to see that this implies that the tetrahedron defined by the points P'_1, P'_2, P'_3 and P'_4 completely contains the portion of the surface lying inside $\mathcal{B}''_c$. Hence, to prune portions of a box that do not satisfy the hyperbolic equations, one can simply introduce the half-planes defining this tetrahedron into **LP1** and **LP2** above.

6 Conclusions

We have presented a method able to give box approximations of the configuration space of a planar linkage. The method is *general*, in the sense that it can manage linkages of any number of links, joined to form kinematic loops of arbitrary topology. It is also *complete*, in the sense that every solution point will be contained in one of the returned boxes. Moreover, in all experiments done so far the algorithm was also *correct*, in the sense that all returned boxes contained at least one solution point. Although in theory this is not guaranteed, returning boxes with no solution seems rather improbable, due to the fact that the approximation of circle and hyperbolic equations introduce errors smaller than the size of the considered boxes. Moreover the fact that all equations are simultaneously taken into account during box reduction (whether directly or in a approximated form) eliminates the so-called *cluster effect*, a known

problem of bisection-based techniques of this kind [16], whereby each solution is obtained as a compact cluster of boxes instead of a single box containing it.

A main contribution with respect to previous works is the method's ability to deal with configuration spaces of general structure. This is accomplished by maintaining a collection of boxes that form a tight envelope of such spaces, which can be refined to the desired accuracy in a multiresolutive fashion. Empirical tests show that the method is quadratically convergent to all roots if these are isolated points, and linearly convergent to them if they form one-dimensional connected components. The presented algorithm can be applied without modification to characterize solution spaces of higher dimension. However, the number of boxes required to approximate such spaces rapidly grows with their dimension and, therefore, the algorithm is likely to exhibit sublinear convergence in such cases.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Education and Science through the I+D project DPI2004-07358, and through two Ramón y Cajal contracts supporting the first two authors.

References

[1] Borcea, C., and Streinu, I. The number of embeddings of a minimally-rigid graph. *Discrete and Computational Geometry 31*, 2 (2004), 287–303.

[2] Castellet, A., and Thomas, F. 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. Kluwer Academic Publishers, 1998, pp. 393–403.

[3] Chartrand, G., and Lesniak, L. *Graphs and Digraphs*, 3rd edition ed. Chapman and Hall, 1996.

[4] Dhingra, A. K., Almadi, A. N., and Kohli, D. Closed-form displacement and coupler curve analysis of planar multi-loop mechanisms using Gröbner bases. *Mechanism and Machine Theory 36* (2001), 273–298.

[5] Didrit, O., Petitot, M., and Walter, E. Guaranteed solution of direct kinematic problems for general configurations of parallel manipulators. *IEEE Trans. on Robotics and Automation 14*, 2 (1998), 259–266.

[6] J. M. Porta. CuikSlam: A kinematics-based approach to SLAM. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation* (2005), pp. 2436–2442.

[7] Jan Verschelde's home page. `http://www.math.uic.edu/~jan`.

[8] Jensen, F., and Pellegrino, S. Planar retractable roofs. Public web document. `http://www-civ.eng.cam.ac.uk/dsl/roof/planar/planar.html`.

[9] Laman, G. On graphs and rigidity of plane skeletal structures. *J. of Engineering Math.*, 4 (1970), 331–340.

[10] Lee, E., Mavroidis, C., and Merlet, J.-P. Five precision points synthesis of spatial RRR manipulators using interval analysis. *ASME Journal of Mechanical Design 126*, 5 (2004), 842–849.

[11] Makhorin, A. GLPK - the GNU linear programming toolkit. `http://www.gnu.org/software/glpk`.

[12] Merlet, J.-P. Direct kinematics of planar parallel manipulators. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation* (1996), pp. 3744–3749.

[13] Merlet, J.-P. *Parallel Robots*. Springer Verlag, 2000.

[14] Merlet, J.-P. A formal numerical approach to determine the presence of singularity within the workspace of a parallel robot. In *Proceedings of the 2nd Workshop on Computational Kinematics* (Seoul, South Korea, May 2001), pp. 167–176.

[15] Merlet, J.-P. An improved design algorithm based on interval analysis for parallel manipulator with specified workspace. In *Proc. of the IEEE Int. Conf. on Robotics and Automation* (Seoul, South Korea, May 2001), vol. 2, pp. 1289–1294.

[16] Morgan, A., and Shapiro, V. Box-bisection for solving second-degree systems and the problem of clustering. *ACM Transactions on Mathematical Software 13*, 2 (1987), 152–167.

[17] Nielsen, J., and Roth, B. Solving the input/output problem for planar mechanisms. *ASME Journal of Mechanical Design 121*, 2 (June 1999), 206–211.

[18] Pennock, G., and Hasan, A. A polynomial equation for a coupler curve of the double butterfly linkage. *ASME Journal of Mechanical Design 124*, 1 (March 2000), 39–246.

[19] Rao, R. S., Asaithambi, A., and Agrawal, S. K. Inverse kinematic solution of robot manipulators using interval analysis. *ASME Journal of Mechanical Design 120*, 1 (1998), 147–150.

[20] Sherbrooke, E., and Patrikalakis, N. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design 10*, 5 (1993), 379–405.

[21] Sommese, A. J., Verschelde, J., and Wampler, C. W. Advances in polynomial continuation for solving problems in kinematics. *ASME Journal of Mechanical Design 126*, 2 (March 2004), 262–268.

[22] Thorpe, M. F., and Duxbury, P. M., Eds. *Rigidity Theory and Applications*. Kluwer Academic Publishers, 1999.

[23] Verschelde, J. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software 25*, 2 (1999), 251–276.

[24] Waldron, K. J., and Sreenivasen, S. V. A study of the position problem for multi-circuit mechanisms by way of example of the Double Butterfly linkage. *ASME Journal of Mechanical Design 118* (1996), 390–395.

[25] Wampler, C. W. Solving the kinematics of planar mechanisms by Dixon's determinant and a complex plane formulation. *ASME Journal of Mechanical Design 123*, 3 (September 2001), 382–387.

[26] Yakey, J. H., LaValle, S. M., and Kavraki, L. E. Randomized path planning for linkages with closed kinematic chains. *IEEE Trans. on Robotics and Automation 17*, 6 (December 2001), 951–958.