SOLVING MULTI-LOOP LINKAGES BY ITERATING 2D CLIPPINGS

J.M. Porta, L. Ros, F. Thomas and C. Torras
Institut de Robòtica i Informàtica Industrial (CSIC-UPC)
Llorens i Artigas 4-6, 08028-Barcelona, Spain
{jporta,lros,fthomas,ctorras}@iri.upc.es

Abstract

A multi-loop linkage can be viewed as a graph of kinematic constraints, and its solution entails finding joint configurations satisfying all constraints simultaneously. In this paper, a general algorithm to obtain all solution configurations is presented. Each loop provides a matrix equation in homogeneous coordinates, which is handled as a constraint on the involved joint variables. The algorithm uses one constraint at a time to reduce the valid ranges for variables, by iteratively applying two simple geometric operations: segment-trapezoid and circle-rectangle clippings. The obtained experimental results qualify this approach as a valuable alternative to other interval-based strategies.

Keywords: Kinematic constraints, constraint graph, geometric constraint systems, multilinear equations, clipping.

1. Introduction

The resolution of graphs of kinematic constraints has aroused interest not only within Kinematics (analysis of multi-loop linkages, simulation of deployable mechanisms), but also within related disciplines such as Robotics (path planning of closed-loop kinematic chains, contact analysis, assembly specification) and Computer Graphics (constraint-based sketching and design, geometric modeling for CAD). In the latter contexts, the problem is formulated as a system of geometric constraints between objects (Kramer, 1992), whose resolution entails finding object positions and orientations that satisfy all constraints simultaneously.

Solving a constraint graph amounts to solving its underlying system of loop-closure equations. Thus, the general methods developed for finding all the roots of systems of algebraic equations can be (and have been) applied to this problem. Three types of methods can be distinguished: algebraic geometry, homotopy and interval-based ones.

Our group has been exploring the latter type of algorithms, first using interval Newton methods (Castellet and Thomas, 1998) and, afterwards,

the subdivision and convex-hull properties of the Bernstein polynomials (Bombín et al., 2000). An interesting feature of this second technique is that it does not require the computation of any Jacobian. Along this line, we propose here a remarkably simpler branch-and-bound algorithm that only relies on two basic 2D clipping operations.

2. From a constraint graph to a system of equations

When reference frames are attached to the objects, the constraints can be expressed as sets of allowed transformations between these reference frames. Although these transformations are usually described by means of homogeneous coordinates and Denavit-Hartemberg parameters, other representations are possible but this does not modify the discussion that follows.

We will only consider transformations with independent translational and rotational degrees of freedom. This includes most problems of practical interest but excludes, for example, those including screw motions because the translational and rotational motions are linked by the pitch of the screw.

Each loop or cycle c in the constraint graph can be expressed as a sequence of homogeneous transformations, where each transformation is either a constant matrix or a translation/rotation matrix including a single variable. In a solution configuration, the result of composing the transformations in the loop must equal the identity. Thus, each cycle leads to a matrix equation:

$$c(x_1, \ldots, x_n, \theta_1, \ldots, \theta_m) = I, \tag{1}$$

where $x_1, \ldots, x_n$ are variables involved in translation transformations, $\theta_1, \ldots, \theta_m$ are rotational variables, and I is the identity matrix.

For multi-loop graphs, not all resulting matrix equations are independent, but only those belonging to a cycle basis of the graph. In the current implementation, we use a basis of fundamental cycles (derived from a spanning tree), but this is clearly a point susceptible of improvement, as discussed in Section 7.

Each matrix equation leads to 12 scalar equations, one for each row i (but the last, which has constant values) and column j in the transformation matrix:

$$f_{i,j}(x_1, \ldots, x_n, \theta_1, \ldots, \theta_m) = \delta_{i,j}, \tag{2}$$

where $\delta_{i,j}$ is 1 if i is equal to j, and 0 otherwise.

Obviously, the system of matrix equations defined by the cycle basis of the graph can be solved by working on the resulting set of scalar

equations. However, the presence of rotational variables in the scalar functions (2) is quite troublesome because it leads to trigonometric expressions. The tangent-half-angle substitution has been the usual choice to transform them into a system of algebraic equations, but it might introduce extraneous roots because of its singularity at $\theta_k = \pm\pi$.

As an alternative, it is possible to introduce the changes $y_k = \cos\theta_k$ and $z_k = \sin\theta_k$ that present no discontinuity. Using this variable substitution, the scalar equations become:

$$f_{i,j}(x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_m) = \delta_{i,j}. \tag{3}$$

The drawback of this variable substitution is that we have m additional variables and, thus, the search space is considerably enlarged. The advantage is that now all $f_{i,j}$ functions are multilinear and, consequently, the equations are simpler to solve.

In the next sections, we describe how these equations can be solved using simple 2D clipping techniques and how the circle-based restriction ($y_k^2 + z_k^2 = 1$) existing between some of the variables is used to ensure that only valid roots are found.

3. Bounding multilinear functions

Because of its application in the field of nonlinear programming, a great deal of effort has been devoted to finding piecewise-linear bounds for multilinear functions. It is well-known that the convex envelope of one such function on the unit hypercube is polyhedral and that finding this envelope is an NP-hard problem.

In this paper, we will use a much looser and easy-to-compute bound, namely the projections of the convex envelope on the planes $(x_k, f(\mathbf{x}))$ for all k. To this end, we will make use of the following result, which is a direct consequence of Theorem 1.1 in (Rikun, 1997):

The point $(\mathbf{x}, f(\mathbf{x})) \in \Re^{n+1}$, where f is a scalar multilinear function and $\mathbf{x} = (x_1, \ldots, x_n) \in [x_1^l, x_1^u] \times \cdots \times [x_n^l, x_n^u]$ is contained in the convex hull of the 2^n points $\{(\mathbf{x}, f(\mathbf{x}))|x_k \in \{x_k^l, x_k^u\}\}$.

This convex hull property leads to a simple clipping strategy for bounding the solutions of a system of multilinear equations, as described in Section 4.1.

4. Two clipping operations

The algorithm we present takes an initial box $\mathcal{B} \subset \Re^n$ in configuration space and uses two clipping operations —*segment-trapezoid clipping* and *circle-rectangle clipping*— to iteratively cut off portions of $\mathcal{B}$ containing no root. For simplicity, we here explain how they work on a system of

just one equation in two unknowns. The extension to multiple equations and any number of variables will then become straightforward, as summarized in Section 5.

4.1 Segment-trapezoid clipping

Assume we want to find all solutions of a multilinear equation $f(\mathbf{x}) = \delta$, $\mathbf{x} = (x_1, x_2)$, for $\mathbf{x}$ in the box $\mathcal{B} = [x_1^l, x_1^u] \times [x_2^l, x_2^u] \in \Re^2$. Since the graph of $(\mathbf{x}, f(\mathbf{x}))$ must lie within the convex hull of the $2^2 = 4$ points $\{(\mathbf{x}, f(\mathbf{x}))|\ \mathbf{x} \in \{x_1^l, x_1^u\} \times \{x_2^l, x_2^u\}\}$, we can compute the convex hull of these points in $\Re^3$ and intersect it with the plane $f(\mathbf{x}) = \delta$ to obtain a polygon whose rectangular hull gives a better bound for the solutions. Although this method seems inefficient for a high number of variables, Bombín et al. (2000) show that the explicit computation of the convex hull may be avoided, and that the overall procedure can be reduced to solving $2n$ linear programming problems. Nevertheless, the method can be further simplified by accepting looser bounds through the following variation, illustrated in Fig. 1. Instead of computing the intersection of the convex hull with the $f(\mathbf{x}) = \delta$ plane, we simply start by projecting the hull onto each coordinate plane as depicted in Fig. 1a, and intersecting each of the resulting trapezoids with the $f(x) = \delta$ line, as shown in Fig. 1b. Usually, these segment-trapezoid clippings reduce the ranges of some variables, and the Cartesian product of them all gives a box smaller than $\mathcal{B}$ still bounding the root locations (the black rectangle in Fig. 1a). The experiments show that, although the latter strategy produces less pruning than the former, a root finding process using it is faster due to the lower cost of its simpler operations.

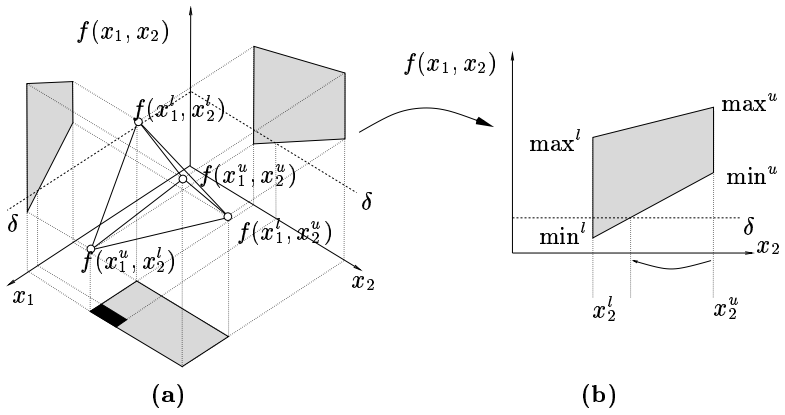

Figure 1. Segment-trapezoid clipping.

4.2 Circle-rectangle clipping

When two variables of a same rotation, say y_k and z_k, are constrained by an equation $y_k^2 + z_k^2 = 1$, their ranges can be further reduced with a circle-rectangle clipping operation, as illustrated in Fig. 2. Suppose that after all possible segment-trapezoid clippings we end up with a certain box $\mathcal{B}$ of possible values for y_k and z_k (see Fig. 2a). Since only the values lying on $y_k^2+z_k^2 = 1$ are consistent with the equations, we can clip this box with the circle, yielding the smaller shaded rectangles in Fig. 2b. Note that, in general, clipping against the unit circle may produce more than one clipped box (Srinivasan, 1992). Although this frequently prunes large portions of the search space, it would also produce a great deal of branching in the search algorithm below, and it is often preferable to take the rectangular hull of all clipped boxes as the output of this clipping operation, rather than the individual boxes themselves. This also permits a simpler box-handling strategy in the search algorithm described next.

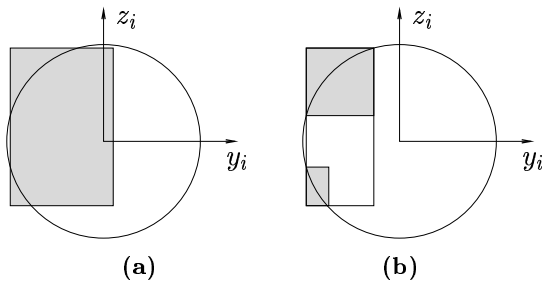

Figure 2. Circle-rectangle clipping.

5. The algorithm

The proposed algorithm, which we have named *Cuik* (Fig. 3), aims at determining a set S of small boxes that include the solutions for a given kinematic problem. The valid ranges for the variables (including those coming from the *sine* and *cosine* substitutions) are used to set up an initial list of boxes. This list constitutes the search space where to look for solution boxes. In this algorithm, a box is a solution when it has a size below a user-provided threshold (σ) and it fulfills all the possible segment-trapezoid and rectangle-cycle clipping tests. This simple criterion may classify as solution boxes those that are very close to a solution point, but that do not actually include the point itself. Avoiding these pseudo-solutions calls for more strict acceptance criteria, as mentioned in Section 7.

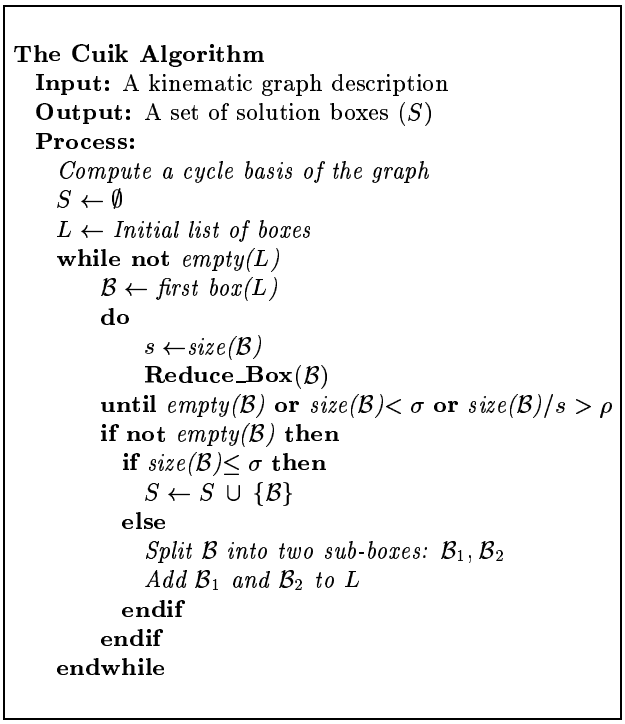

Figure 3. The Cuik algorithm.

To determine the set of solution boxes, each one of the initial boxes is reduced as much as possible by the successive application of the trapezoid and circle clippings described in the previous section (see Fig. 4). Three possible outcomes are possible from the box-reduction process:

- **The box becomes empty:** In any of the clippings a variable interval can become empty because there is no intersection between the segment and the trapezoid or between the circle and the rectangle. This indicates that the analyzed box does not contain solution points and we can simply stop the exploration in the search space delimited by this box.
- **The box becomes small:** Then we add it to the solution set.
- **The box can not be significantly reduced:** If the reduction ratio of the box is above a user-provided threshold (ρ), the box reduction is stopped and the box is split into two sub-boxes. In the current version of the algorithm, we simply split the box along its larger dimension. However, as commented in section 7 this

Reduce_Box($\mathcal{B}$)
Input: A box defined as a set of intervals:
$\mathcal{B} = \{[x_1^l, x_1^u], \ldots, [x_n^l, x_n^u]\}$
Output: The same box but eventually resized
Process:
for each cycle c
$V \leftarrow \{v_0, \ldots, v_k\}$ *(Set of indexes of variables involved in c)*
$C = \{c(\mathbf{x}) \,|\mathbf{x} = (x_{v_0}, \ldots, x_{v_k}) \in \{x_{v_0}^l, x_{v_0}^u\} \times \ldots \times \{x_{v_k}^l, x_{v_k}^u\}\}$
for each $v \in V$
for each $i \in [1,3]$ **and** $j \in [1,4]$
$\min^l \leftarrow \min\{c_{i,j} \mid c \in C,\ x_v = x_v^l\}$
$\max^l \leftarrow \max\{c_{i,j} \mid c \in C,\ x_v = x_v^l\}$
$\min^u \leftarrow \min\{c_{i,j} \mid c \in C,\ x_v = x_v^u\}$
$\max^u \leftarrow \max\{c_{i,j} \mid c \in C,\ x_v = x_v^u\}$
if $i = j$ **then** $\delta = 1$
else $\delta = 0$
Trapezoid_Clipping$(x_v^l, x_v^u, \min^l, \max^l, \min^u, \max^u, \delta)$
if x_v *is a rotation variable* **then**
$w \leftarrow$ *Index of the circle-related variable of* x_v
Circle_Clipping$(x_v^l, x_v^u, x_w^l, x_w^u)$
endif
endfor
endfor
endfor

Figure 4. The **Reduce_Box** function. $c(x_0, \ldots, x_k)$ refers the evaluation of the cycle matrix equation at point $(x_0, \ldots, x_k)$, and $c_{i,j}$ denotes the element at row i and column j of matrix c.

point requires further consideration. After the split, the circle clipping process is used to transmit the interval reduction to the corresponding circle-related variable. If the two new boxes are added to the beginning of the list, the Cuik algorithm implements a depth-first search, and if they are added to the end of the list the search is breadth-first.

Two factors have to be taken into account to determine the cost of any interval-based algorithm: the number of processed boxes and the cost to process each box. In the case of the Cuik algorithm, the emphasis is put in efficiently processing each box, even at the cost of having to process more boxes than if a more costly box reduction process were used.

Note that the algorithm in Fig. 4 projects 4 times the 2^k cycle evaluations for each cycle c_l ($l \in [1..n]$), for each variable x_v ($v \in [1..k]$), and for each element of the matrix equation. This amounts to $4 \cdot n \cdot k \cdot 12 \cdot 2^k$

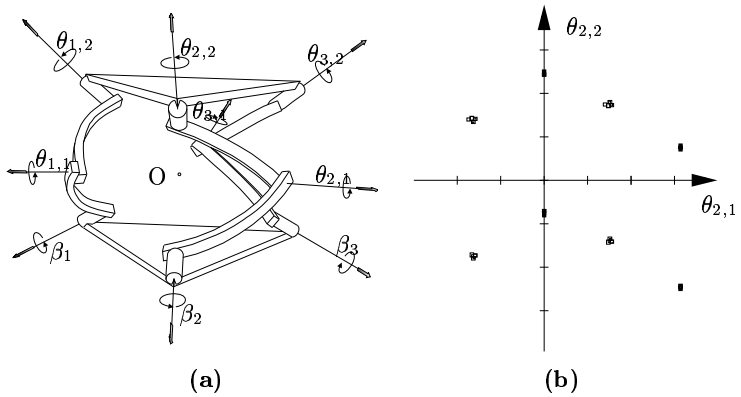

Figure 5. **(a)** The Gosselin platform. **(b)** The solution boxes determined by the Cuik algorithm with $\beta_1 = -0.2266$, $\beta_2 = -0.7103$ and, $\beta_3 = 0.073$.

point projections and, hence, even for few variables per cycle, this is the most expensive step of the box-reduction process.

One advantage of the Cuik algorithm is that box-reductions are completely independent of one another and, therefore, the algorithm can be readily parallelized with the consequent increment in efficiency.

6. Experimental results

Fig. 5a shows the platform presented in (Gosselin and Gagné, 1995). In this mechanism, the β parameters are the inputs for the platform actuators and the θ's are the joint values to be determined by the Cuik algorithm. This platform has eight valid configurations, four attainable for any value of the actuators parameters, and four particular for each set of inputs.

Observe that this problem involves six rotational variables and, consequently, the Cuik algorithm works in a twelve-dimensional space.

The kinematic graph for this platform has two fundamental cycles, each involving four variables (two of them shared with the other cycle).

The initial range for the θ variables is $[0, 2\pi]$. With $\sigma = 0.1$, these initial ranges conform a search space where approximately $4 \cdot 10^{15}$ solution boxes can be defined. Despite this large search space, the problem is solved by an optimized version of the Cuik algorithm in less than 1.25 seconds on a Pentium III at 700MHz, using the parameters $\sigma = 0.1$ and $\rho = 0.8$.

In this example, the algorithm starts with a single box and it examines up to 679 boxes, only 58 of which are considered solutions. Fig. 5b shows the solution boxes projected on a plane defined by a pair of prob-

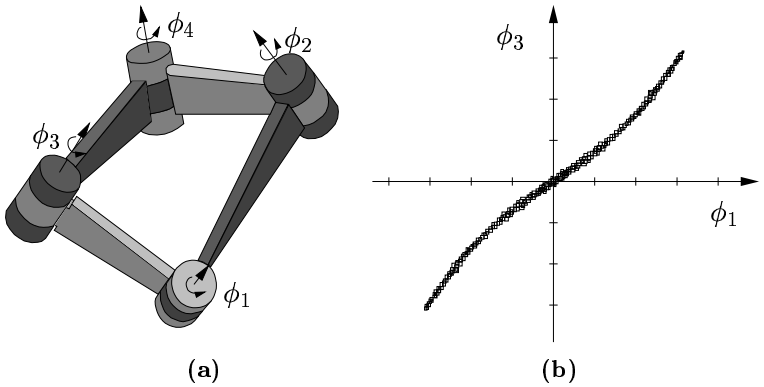

Figure 6. **(a)** The Bennett linkage. **(b)** The discretization of the one-dimensional space of solutions provided by the Cuik algorithm.

lem variables. We see that, as expected, the solution boxes accumulate around the eight solution points of the proposed problem.

If we need more precision in the solution boxes, we can run the program with a lower value for the parameter σ. For instance, for $\sigma = 0.01$ (10 times smaller than before), the search space is enlarged up to $4 \cdot 10^{27}$, but the execution time is increased in less than 0.2 seconds.

Fig. 6a represents the Bennett's linkage, a 4R spatial closed chain with 1-d.o.f. mobility. Fig. 6b shows the discretization of the one-dimensional solution space provided by the Cuik algorithm. This discretization includes about 300 boxes and is obtained in less than 1 second (with $\sigma = 0.1$ and $\rho = 0.8$).

We have also tested our implementation with more complex problems. As a reference, all the valid solutions of a 6R manipulator are found by the algorithm in about 5 minutes.

7. Conclusions and envisaged enhancements

In this paper, we have shown the feasibility of efficiently solving complex kinematic problems using very simple 2-D clipping techniques.

The presented Cuik algorithm can be seen as the skeleton of a software package to solve arbitrary graphs of kinematic constraints. Currently, the algorithm deals with cycles, variables and matrix components in an exhaustive, uninformed way. Future work will try to develop good heuristics for all choice points in the algorithm. For example, it is not clear what the most suitable cycle basis is, or even if the addition of redundant cycles could speed up the process. Likewise, first solving the cycles with discrete solutions may prune drastically the search space. It seems equally advisable to begin by processing variables shared between

many cycles. Another point that also deserves consideration is that of deciding how to split boxes whose size can not be further reduced. Solution accuracy should also be improved by using more elaborate criteria to determine whether a small box contains a root.

The results we have obtained show that the presented algorithm is specially adequate for kinematic problems with many loops but with few variables per loop (up to 4 or 5). To efficiently confront problems with more variables per loop, the algorithm should be extended with other techniques for solving algebraic equations.

Acknowledgements

This research has been partially supported by the Catalan Research Commission, through the "Robotics and Control" group, and the Spanish CICYT under contracts TAP99-1086-C03-01 and TIC2000-0696.

References

Bombín, C., Ros, L., and Thomas, F. (2000), A concise Bézier clipping technique for solving inverse kinematics problems, in *Advances in Robot Kinematics,* J. Lenarcic and M.M. Stanisic (eds.), pp. 53-60, Kluwer Academic Publishers.

Castellet, A., and Thomas, F. (1998), An algorithm for the solution of inverse kinematic problems based on an interval method, in *Advances in Robot Kinematics,* M. Husty and J. Lenarcic (eds.), pp. 393-403, Kluwer Academic Publishers.

Gosselin C. M. and Gagné M. (1995), A close-form solution for the direct kinematics of a special class of spherical three-degree-of-freedom parallel manipulators, in *Computational Kinematics*, J.-P. Merlet and B. Ravani (eds.), pp. 231-240, Kluwer Academic Publishers.

Kramer, G.A. (1992), *Solving Geometric Constraint Systems*, Cambridge, The MIT Press.

Rikun, A.D. (1997), A convex envelope formula for multilinear functions, *J. of Global Optimization*, vol. 10, pp. 425-437.

Srinivasan, R.V. (1992), A fast circle clipping algorithm, in *Graphics Gems III*, pp. 182-187, Academic Press.