Solving Geometric Constraint Systems

Glenn A. Kramer

School of Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH, England

Schlumberger Laboratory for Computer Science*
P.O. Box 200015
Austin, Texas 78720-0015

gak@slcs.slb.com

Abstract

Finding the configurations of a set of rigid bodies that satisfy a set of geometric constraints is a problem traditionally solved by reformulating the geometry and constraints as algebraic equations which are solved symbolically or numerically. But many such problems can be solved by reasoning symbolically about the geometric bodies themselves using a new technique called degrees of freedom analysis. In this approach, a sequence of actions is devised to satisfy each constraint incrementally, thus monotonically decreasing the system's remaining degrees of freedom. This sequence of actions is used metaphorically to solve, in a maximally decoupled form, the equations resulting from an algebraic representation of the problem. Degrees of freedom analysis has significant computational advantages over conventional algebraic approaches. The utility of the technique is demonstrated with a program that assembles and kinematically simulates mechanical linkages.

Introduction

Solving geometric constraint systems is an important problem with applications in many domains, for example: describing mechanical assemblies, constraint-based sketching and design, geometric modeling for CAD, and kinematic analysis of robots and other mechanisms. An important class of such problems involves finding the configurations (positions and orientations) of a set of rigid bodies that satisfy a set of geometric constraints. This paper first examines traditional means of solving such problems. *Degrees of freedom analysis* is then introduced as a novel and more intuitive solution technique with substantially better computational properties. The power of this technique is demonstrated with a system that kinematically simulates mechanical linkages.

Mechanical design presents interesting challenges due to the intimate role that complex 3D geometry plays in design analysis and synthesis [Dixon, 1986]. While algebraic methods are dominant in mechanism analysis, purely geometric methods are also used because they 'maintain touch with physical reality to a much greater degree than do the algebraic methods' and 'serve as useful guides in directing the course of equations' [Hartenberg and Denavit, 1964].

*Author's current address.

This paper describes how to use geometric reasoning to guide the solution of the sets of complicated nonlinear equations that arise in mechanism simulation. A program called TLA embodies this methodology. It simulates a mechanism by first reasoning at the geometric level about how to assemble it. TLA then uses this assembly plan as a metaphor to solve the equations in a stylized, highly decoupled manner. Efficient solution is important because these equations are solved repeatedly in tasks such as simulation and optimization. The approach described in this paper greatly reduces the computational complexity of solving such systems, and is a strategy which is unique to TLA.

Kinematic simulation

Kinematic analysis answers questions about the motion of mechanisms, without regard to the forces which produce that motion [Hartenberg and Denavit, 1964]. *Kinematic assembly* of a mechanism requires determining the configuration of each body to satisfy all assembly constraints. These are either *joint constraints*, which describe how bodies may move relative to each other, or *driving input constraints*, which further restrict a joint by specifying a value for an angle or displacement.

Kinematic simulation involves repeatedly finding the configurations of the parts of a mechanism for particular values of the driving input constraints; this is effectively the same as repeatedly assembling the mechanism for different values of the driving inputs. As the values of the driving inputs are varied, the mechanism will trace its characteristic path. The motion is a function only of geometric relationships between the various joints. Thus, engineers use *kinematic diagrams*, which are stick-figure 'schematics' of mechanisms. They contain only geometry relating the joints, *not* the actual shapes or boundaries of the parts. They help designers understand a mechanism's kinematic behavior. This research is concerned with simulating mechanisms at the level of kinematic diagrams.

Mechanical constraints

Constraints describing joint behavior can be modeled as relationships between sets of points on different bodies. A *marker* consists of a point in 3D space, along with two orthogonal axes, z and x, which emanate from the point. The position of a marker is the position of its point, while its orientation is determined by its axes. Since all bodies are rigid, constraints between markers constrain the bodies to which they are attached. The constraints between pairs of markers m_1 and m_2 are:

coincident(m_1, m_2): Markers m_1 and m_2 are spatially coincident.

in-line(m_1, m_2): m_1 lies on the line through m_2 parallel to m_2's z axis.

in-plane(m_1, m_2): m_1 lies in the plane through m_2 normal to m_2's z axis.

parallel-z(m_1, m_2): the z axes of markers m_1 and m_2 are parallel.

perpendicular-z(m_1, m_2): the z axes of m_1 and m_2 are perpendicular.

co-oriented(m_1, m_2, α): the z axes of m_1 and m_2 are parallel; and the angle from m_1's x axis to m_2's x axis is α.

screw(m_1, m_2, δ): the z axes of m_1 and m_2 are parallel; and the angle from m_1's x axis to m_2's x axis is linearly related to the distance between m_1 and m_2 by a pitch constant δ.

Combinations of these constraints, relating markers on different rigid bodies, may be used to model all of the 'lower pair' joints described by Reuleaux [Reuleaux, 1876]. For example, a revolute joint, which allows one rotational degree of freedom between two bodies, is modeled with a **coincident** constraint and a **parallel-z** constraint. A translational, or prismatic, joint is modeled with an **in-line** constraint and a **co-oriented** joint. Some types of higher pairs may also be modeled with the above constraints, for example, the 'universal' joint and 'slotted pin' joint. The constraints defined above are sufficient to describe all mechanical linkages as well as many static mechanical assemblies; there is no restriction to 'fixed axis' mechanisms as is common in the literature [Faltings, 1989; Joskowicz, 1987].

Figure 1 illustrates the modeling of a crank-slider mechanism. The crank-slider consists of three parts. The *ground*, **G**, is fixed in space, and serves as the global reference frame. Markers $g1$ and $g2$ are therefore also grounded, or fixed in space. In the figure, marker

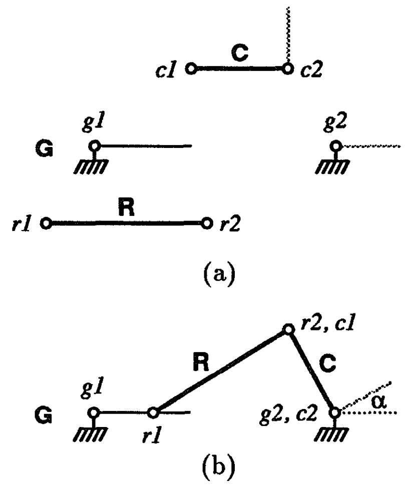

Figure 1: Crank-slider: (a) parts; (b) assembled.

z axes are shown in black; if not shown, they point out of the page. Relevant marker x axes are shown in grey. The geometric constraints are:

in-line$(r1, g1)$	**coincident**$(g2, c2)$
coincident$(r2, c1)$	**parallel-z**$(g2, c2)$
parallel-z$(r2, c1)$	**co-oriented**$(g2, c2, \alpha)$

The **in-line** constraint models a pin ($r1$) in a slot ($g1$'s z axis). The **coincident, parallel-z** pairs model revolute joints. The revolute joint $g2, c2$ has a driving input α, which fully constrains crank **C**'s position and orientation relative to ground. Rotation of the crank is accomplished by changing the value of α. As the crank **C** rotates, marker $r1$ of the connecting rod **R** slides along the z axis of grounded marker $g1$.

Equational solution

Constraint systems like the crank-slider described above are usually solved by modeling the geometry and constraints with algebraic equations. A local coordinate frame is assigned to each body. Then the configuration variables of the different bodies — the six quantities that uniquely specify a local coordinate frame [Snyder, 1985] — are related by equations that model the problem constraints. Solving these equations yields the desired configuration for each body.[1] A simple example, involving a single rigid body, illustrates the solution of a small set of such equations: the brick of Figure 2 must be configured to satisfy the three co-

[1]Solving these types of equations for robotics applications is usually not too difficult because most robot manipulators are open-loop mechanisms. Mechanical linkages, however, involve closed loops. This leads to a much greater degree of equation coupling. Hence, solving these equations must be done simultaneously and is substantially more difficult.

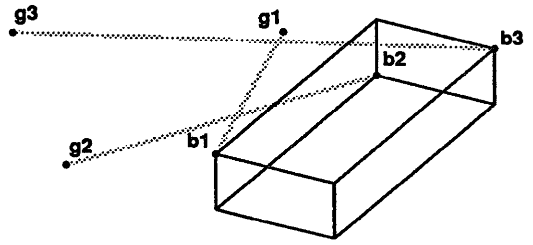

Figure 2: A brick with three **coincident** constraints.

incident constraints graphically depicted as the grey lines between the brick's markers $b1, b2, b3$ and the desired locations, denoted by markers $g1, g2, g3$ fixed in the global coordinate frame. Equations are developed to relate the configuration variables of the brick's coordinate frame to those of the global coordinate frame. The equations may then be solved either numerically or symbolically.

Numerical solution

Numerical solutions represent constraints using *error terms*, which have zero value when the constraint is satisfied, and otherwise have some value proportional to the degree to which the constraint is violated. The *objective function* is the sum of all error terms. Numerical techniques try to find a zero of the objective function by 'sliding' down the function's gradient. This process is necessarily iterative for nonlinear problems, which include any problem involving rotation. Figure 3 shows, in grey, some of the intermediate configurations reached using Newton-Raphson iteration (one of the most efficient methods [Press *et al.*, 1986]) to move the brick from its initial configuration to one satisfying the constraints. Numerical techniques have many drawbacks. Each iteration of Newton-Raphson is slow, taking $O(n^3)$ time, where n is the number of constraints. Overconstrained situations, which are quite common, require pre- and post-analysis to remove redundant constraints before solving and to check them later for consistency.

Symbolic solution

Symbolic solutions use algebraic re-write rules or other techniques to isolate the configuration variables in the equations in a predominantly serial fashion. Once a solution is found, it may be re-used (executed) on any topologically equivalent problem. Execution is fast, approximately linear in the number of constraints. If numerical stability is properly addressed, the solution can be more accurate by virtue of being analytic; there is no convergence tolerance as found in numerical techniques. The principal disadvantage of symbolic techniques is the excessive — potentially exponential — time required to find a solution or determine one does not exist. Poorly-chosen configuration variable assign-

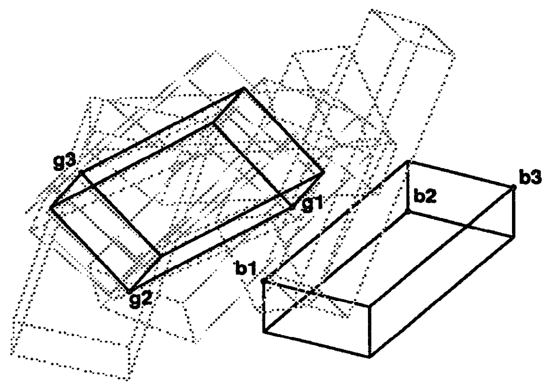

Figure 3: Brick solution using Newton-Raphson.

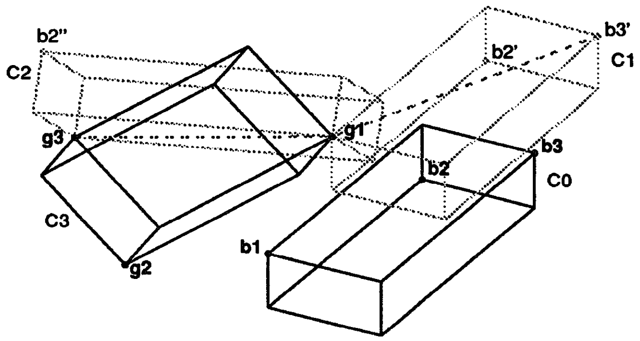

Figure 4: Brick solution using geometric approach.

ments can exacerbate the problem by coupling the equations in unnecessarily complicated ways, requiring more clever and complex inferences. Thus, the symbolic techniques are feasible and complete only for small problems.

Many shortcomings of the above methods can be traced to problems inherent in the configuration variable representation and the complexity of the resulting equations. This suggests a different approach to the solution of geometric constraint problems: avoid equational reformulation entirely, reasoning instead directly about the geometric entities. A program called TLA has been developed to do this.

Geometric solution

TLA solves the brick problem using geometric knowledge to satisfy the constraints incrementally. The solution is shown in Figure 4. Assume that initially the brick is free to move anywhere; it just happens to be in the given initial configuration **C0**. To satisfy **coincident**$(b1, g1)$, TLA translates the brick by the vector from $b1$ to $g1$, leaving the brick in configuration **C1**. To ensure **coincident**$(b1, g1)$ remains satisfied, all further actions that move the brick must be rotations about $g1$, *i.e.,* the brick has only its rotational degrees of freedom left. To satisfy **coincident**$(b3, g3)$, TLA measures the vector $v1$ from

$g1$ to $b3'$ (where $b3$ has been moved by the previous translation) and vector $v2$ from $g1$ to $g3$. These two vectors are shown as dashed lines in Figure 4. Then TLA rotates the brick about $g1$ around vector $v1 \times v2$ by the angle between $v1$ and $v2$, to configuration **C2**. This satisfies **coincident**$(b3, g3)$ without violating **coincident**$(b1, g1)$. This action also removes two of the remaining rotational degrees of freedom; in order to preserve the two already-satisfied constraints, all future actions must be rotations about $v2$. To satisfy the final constraint, TLA drops perpendiculars from $b2''$ to $v2$, and from $g2$ to $v2$, and rotates the brick about $v2$ by the angle between the perpendiculars. This brings the brick to its final configuration. The solution is very deliberate, as opposed to the meandering of the numerical approach of Figure 3. The sequence of actions performed above constitute a plan for moving the brick from an arbitrary position to one satisfying the constraints.

For this part of the problem solution, TLA reasons only about geometry, actions and degrees of freedom. No equations are developed, and no model requiring configuration variables or other abstract state is needed. Constraints are satisfied by measuring the brick's geometric properties (often using additional geometric constructions) and then moving it. This method is called *degrees of freedom analysis.* The brick-moving plan derived using this method is next used to solve for the brick's configuration variables as represented in a computer; this may be done regardless of how the local coordinate frame of the brick is described. All that is required is a set of operators for translating and rotating rigid bodies, and a set of functions that can measure, relative to a global coordinate system, points and vectors attached to any rigid body. These capabilities are provided by homogeneous coordinate transforms [Snyder, 1985], which most 3D graphics and robotics systems use.

The plan, when executed, becomes a metaphor for solving the equational representation of the constraint system. By using the primitive actions of translation and rotation, which are implemented as matrix multiplications, the plan effectively decouples the equations into small independent sets that can be solved analytically.[2] As new constraints are satisfied, previously satisfied constraints (which may correspond to complicated relations between configuration variables) become invariants for later steps in the solution. Geometry, as used in the metaphorical plan, provides the vocabulary and operators that allow preserving these invariants. The use of the assembly plan as a metaphor to guide equation solution distinguishes TLA from other programs that solve large sets of nonlinear equations.

Since the plan does not depend on metric properties of the problem, it can be executed on any topologically equivalent problem.[3] The time required for plan generation is thus amortized over repeated executions.

Degrees of freedom analysis

TLA keeps track of the number and types of degrees of freedom each body (or *link*) has as it solves a problem. It represents this information with predicates of the form **link-has-n-TDOF**(*link, arg1, arg2, ...*) and **link-has-n-RDOF**(*link, arg1, arg2, ...*), for $n \in \{0, 1, 2, 3\}$. TDOF stands for translational degrees of freedom, and RDOF for rotational degrees of freedom. The arguments *arg1, arg2, ...* specify any fixed points or axes on the links that restrict their freedom. Initially, every link in the system except the grounded body has 3 TDOF and 3 RDOF. As actions are taken to satisfy constraints, the links in the system lose some of their degrees of freedom. When all bodies have 0 TDOF and 0 RDOF, the problem is solved.

At each step in solving for a body's configuration, TLA must know what action to take given the body's current constraints, and how that action further reduces the body's degrees of freedom. This information is stored in a *plan fragment table.* Conceptually, the plan fragment table is a three-dimensional dispatch table, indexed by TDOF, RDOF, and constraint type. Each entry in the table specifies how to move the rigid body to satisfy the new constraint using only available degrees of freedom, and what degrees of freedom the body will have after the action is performed. The plan fragment table contains information about how to satisfy constraints when one of the markers participating in the constraint has its appropriate attributes fixed, or *globally known.* Thus, a globally known position of one marker is required for solving a **coincident** constraint, and a globally known z axis is needed to solve a **perpendicular-z** constraint.

In the brick example, the first constraint to be satisfied is arbitrarily chosen to be **coincident**$(b1, g1)$. The global position of $g1$ is known. Initially the brick has 3 TDOF and 3 RDOF; thus the index into the plan fragment table is $\langle 3, 3,$ **coincident**$\rangle$. This entry contains the following information (modified for readability):

Initial state:
link-has-3-TDOF(*link*)
link-has-3-RDOF(*link*)

[2]Not all problems may be solved analytically; some require iterative solutions. In such cases TLA fails in the plan construction phase. It is possible, however, to use the information from the failure to reduce significantly the dimensionality of the iterative problem that must be solved. See [Kramer, in preparation] for details.

[3]Actually, this is not quite true. Mathematical degeneracies may cause the plan to fail. For example, the brick plan fails to remove the final rotational degree of freedom if the three markers are collinear. TLA can test for such degeneracies, and try to generate a new plan taking them into account, if possible.

Plan fragment:
begin
translate(*link*,
  vector-difference(**gmp**(*M1*),
    gmp(*M2*)));
end;

New state:
link-has-0-TDOF(*link*, **gmp**(*M2*))
link-has-3-RDOF(*link*)

Explanation:
Body *link* is free to translate. A **coincident** constraint must be satisfied between marker *M1*, whose global position is known, and marker *M2* on *link*. Therefore *link* is translated by the vector from the current global position of *M2* to the known global position of *M1*. This action removes all three translational degrees of freedom.

The variable *link* is bound to the object representing the brick. The initial state of the link is that it has all six of its degrees of freedom; it is free to translate and rotate through space. The variable $M1$ gets bound to the globally known marker (*i.e.*, $g1$), while variable $M2$ is bound to the underconstrained marker in the **coincident** constraint being satisfied (*i.e.*, $b1$). The plan fragment specifies how to move the body to satisfy the constraint (the function name **gmp** stands for "global marker position"). In the specification of the new state, the predicate **link-has-0-TDOF** has an additional argument which specifies the point on the body which is constrained to be stationary. The textual explanation — with variable names replaced by their bindings — helps the user to understand the solution process.

The next constraint satisfied in the brick example is **coincident**$(b3, g3)$. Since the brick now has 0 TDOF and 3 RDOF, the index into the plan fragment table is $\langle 0, 3,$ **coincident**$\rangle$. The plan fragment in that entry specifies how to rotate a body with 0 TDOF, 3 RDOF to satisfy a **coincident** constraint, and specifies that the new state of the body is 0 TDOF, 1 RDOF. The process continues until all constraints are satisfied.

For the constraints defined in this paper, there are 112 valid entries in the plan fragment table; some plan fragments are quite simple, like the one described above, while others involve more complex calculations and conditionals to handle potential mathematical degeneracies. The complete plan fragment table appears in [Kramer, in preparation].

Interacting bodies

Bodies rarely interact exclusively with fixed points, as in the brick example. Often, they interact with other partially constrained bodies. In Figure 5 body **A** is constrained to 0 TDOF, 1 RDOF by the constraints

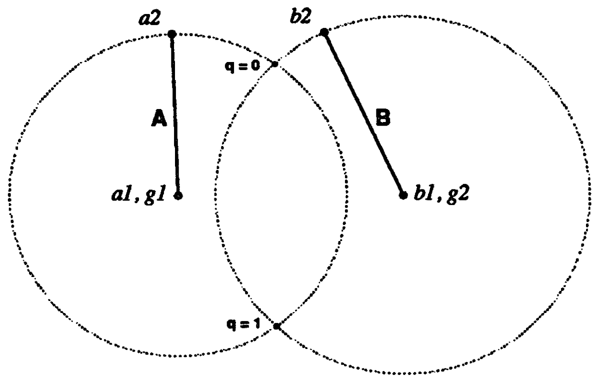

Figure 5: Solving for two interacting bodies (z axes point out of the page).

coincident$(a1, g1)$ and **parallel-z**$(a1, g1)$. TLA infers that marker $a2$ must lie on a circle about $a1$. Body **B** is similarly constrained. To satisfy **coincident**$(a2, b2)$, TLA intersects the circles to find the two globally acceptable locations for the markers. TLA distinguishes the locations with a branch variable **q**. A user of TLA chooses which solution to use by specifying the value of **q**. TLA places a 'pseudo-marker' p at the this location; this is a marker which is not part of the original problem specification, but is introduced during the problem solution.

With the intersection point defined, TLA satisfies the **coincident** constraint for bodies **A** and **B** independently. It does this by introducing the constraints **coincident**$(p, a2)$ and **coincident**$(p, b2)$. Since p's position is globally known, the plan fragment table may be used to find the appropriate actions to satisfy the two introduced constraints. When they are satisfied, **coincident**$(a2, b2)$ is also satisfied.

In this manner, local information, in the form of loci of points on partially constrained bodies, may be combined through locus intersection to yield information about globally permissible locations of points. Pseudo-markers denote these intersections, and auxiliary constraints are introduced to relate the partially constrained markers to the pseudo-marker. Then the plan fragment table is used to find the appropriate actions to satisfy the constraints.

TLA uses a *locus table* to specify the loci to which partially constrained markers are confined. Loci are determined completely by the degrees of freedom that a body has. For example, all markers on a body with 0 TDOF, 1 RDOF are constrained to lie on circles around the body's fixed point. Markers on a body with 0 TDOF, 3 RDOF must lie on spheres, and markers on a body with 2 TDOF, 0 RDOF must lie in planes.

A *locus intersection table* allows TLA to know when enough information is known about sets of partially constrained markers to determine their configurations fully. This table has entries for all pairs of shapes in the locus table. For example, a sphere intersected with a circle yields at most two discrete points (except in the degenerate case of the circle lying on the sphere); a plane intersected with a cylinder yields an ellipse. For the constraints described in this paper, all loci are analytically describable, as are all pairwise intersections of loci.

Plan generation

The plan fragment table and locus tables allow simple and efficient algorithms to solve geometric constraints. TLA's metaphorical plan construction differs from blocks world planners like HACKER [Sussman, 1973], which generate ***physically realizable*** plans to get from one world state to another. Kinematic diagrams do not represent the true physical boundaries of the mechanism's parts, so the geometric entities may pass through each other in intermediate plan states as they move toward their final configurations. TLA's only concern is the final plan state, where all objects satisfy their constraints. This lack of concern about intermediate states allows TLA to satisfy the constraints incrementally, without backtracking.

An assembly plan generated by TLA is compiled into an ***assembly procedure***, which is a machine executable version of the plan. The assembly procedure is optimized in various ways: removing nested function calls, removing duplicate calculations, *etc.* Mechanism simulation is accomplished by alternately changing the values of the driving inputs and then calling the assembly procedure; the simulation moves the mechanism through its characteristic motions.

The assembly procedure may be reused when the sizes and shapes of the parts change; however, if the mechanism topology (*e.g.*, number of bodies, or number or types of joints connecting the bodies) changes, a new assembly plan and procedure must be derived.

Implementation

The current version of TLA is written in Common Lisp and CLOS, and runs on a Symbolics Lisp Machine. A rule-based system generates the assembly plans. Each rule implements part of the plan fragment table or the locus tables, of which about 60% have been implemented to date. A database stores assertions during the assembly planning. The database grows monotonically; no retractions are made. A simple pattern matcher is used, rather than full unification, and the few search heuristics (for efficiency only) are hard-wired into the rule triggers. This allows a simple control structure:

- Make any applicable deduction (*e.g.*, 'marker m lies on a circle').
- Perform any applicable action (*e.g.*, a translation or rotation).
- Succeed when all bodies have zero degrees of freedom.
- Fail when there is no applicable deduction or action.

While a rule-based system allowed flexibility in deciding how TLA would be structured, a future implementation will use explicit tables and object-oriented programming to avoid the need for pattern matching, substantially reducing the computational complexity of constructing assembly plans.

Complexity analysis

A complete analysis of the computational complexity of TLA is given in [Kramer, in preparation]; only the results appear here. For the rule-based implementation, plan generation time is $O(n^d)$, where n is the number of constraints, and d is a constant determined by the average number of arguments for each database predicate ($d \approx 3$). In practice, TLA's plan generator tends to run in time nearly proportional to n^2.

Thus, for generating a solution, TLA's planning algorithm has polynomial complexity, as opposed to the exponential complexity of symbolic algebraic techniques. For executing a solution, TLA's compiled plan runs in $O(n)$ time, as opposed to the $O(n^3)$ time of iterative numerical methods.

Speed comparisons

TLA has simulated dozens of complex planar and spatial mechanisms, the largest example being a sofa-bed, shown in Figure 6. This mechanism has 16 links, 22 joints, and two driving inputs, and is described by 115 algebraic constraints, 19 of which are redundant. A plan is generated in 297 seconds, and the assembly procedure compiled from it (655 lines of Lisp code) executes in 0.29 seconds on a Symbolics 3675. This is almost two orders of magnitude faster than simulation speeds using some of the commercially available numerically-based simulators, after scaling for differences in processor speed (commercial programs run on machines other than the Symbolics).

Discussion

Algebra has long been the *lingua franca* of science and engineering, but it can provide only a partial appreciation of the actual domain under study. An understanding of geometry is essential to solving problems insightfully and efficiently in the mechanical world. TLA demonstrates this for the task of mechanical assembly and simulation. By using geometry to guide equation solving, TLA provides orders of magnitude speedup over 'general-purpose' mathematical techniques. This

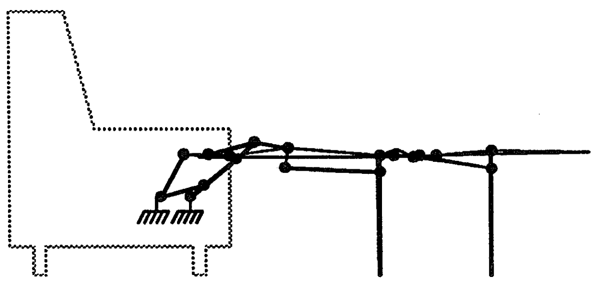

Figure 6: Sofa-bed mechanism (extended).

means that interactive tools for the simulation, optimization, and synthesis of complex mechanical devices become feasible [Kramer and Barrow, 1989].

Using degrees of freedom analysis to generate an assembly plan, and using the resulting plan as a metaphor to guide equation solution both appear unique to TLA. Sketchpad [Sutherland, 1963] and ThingLab [Borning, 1979] represented geometric constraints equationally, relying on relaxation for nonlinear equations. Popplestone *et al.* explored, with limited success, solving assembly problems algebraically using some geometric guidance [Popplestone *et al.*, 1980]. More recently Popplestone has focused on using group theory to represent geometric symmetries [Popplestone, 1987]. This work could profitably be incorporated into TLA. Faltings [Faltings, 1989] and Joskowicz [Joskowicz, 1987] are investigating deriving kinematic constraints directly from geometry. Such a facility would free the user of TLA from having to model a mechanism in terms of abstract concepts like markers.

The ideas embodied in TLA may be extended in many ways, including: expanding the range of constraints TLA understands (*e.g.*, gears, cams, *etc.*); analyzing dynamic behavior more efficiently by virtue of understanding the kinematics; using knowledge of geometry to aid in design synthesis. In all of these cases, geometric knowledge leads to a better understanding of the underlying mathematics. Degrees of freedom analysis allows unifying geometric reasoning with algebraic techniques for efficient and intuitive modeling of real-world mechanisms and assemblies.

Acknowledgments

Phil Agre helped implement the latest version of TLA, and contributed technically in many ways. I would also like to thank Harry Barrow, David Barstow, Geoff Goodhill, David Gossard, Walid Keirouz, Mark Shirley, Reid Smith, Bob Young, and David Ullman.

References

[Borning, 1979] Alan H. Borning. *Thinglab: A Constraint-Oriented Simulation Laboratory.* PhD thesis, Stanford University, Stanford, California, July 1979.

[Dixon, 1986] John R. Dixon. Artificial intelligence and design: A mechanical engineering view. In *Proceedings of the National Conference on Artificial Intelligence*, pages 872–877, Seattle, WA, 1986.

[Faltings, 1989] Boi Faltings. Reasoning about kinematic topology. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit, Michigan, August 1989.

[Hartenberg and Denavit, 1964] R. S. Hartenberg and J. Denavit. *Kinematic Synthesis of Linkages.* McGraw Hill, New York, 1964.

[Joskowicz, 1987] Leo Joskowicz. Shape and function in mechanical devices. In *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, August 1987.

[Kramer and Barrow, 1989] Glenn A. Kramer and Harry G. Barrow. New approaches to linkage synthesis. In *International Joint Conference on Artificial Intelligence* (video track), Detroit, Michigan, August 1989.

[Kramer, in preparation] Glenn A. Kramer. *Geometric Reasoning in the Kinematic Analysis of Mechanisms.* PhD thesis, University of Sussex, Brighton, UK, (in preparation).

[Popplestone *et al.*, 1980] R. J. Popplestone, A. P. Ambler, and I. M. Bellos. An interpreter for a language for describing assemblies. *Artificial Intelligence*, 14(1):79–107, August 1980.

[Popplestone, 1987] R. J. Popplestone. The Edinburgh Designer System as a framework for robotics or, the design of behavior. COINS Technical Report 87-47, University of Massachusetts, Amherst, MA, May 1987.

[Press *et al.*, 1986] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge, England, 1986.

[Reuleaux, 1876] M. M. Reuleaux. *The Kinematics of Machinery.* Macmillan & Co., New York, 1876. Translated by Alex B. W. Kennedy.

[Snyder, 1985] Wesley E. Snyder. *Industrial Robots: Computer Interfacing and Control.* Industrial Robot Series. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[Sussman, 1973] Gerald Jay Sussman. *A Computational Model of Skill Acquisition.* PhD thesis, MIT, Cambridge, Massachusetts, August 1973.

[Sutherland, 1963] Ivan E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System.* PhD thesis, MIT, Cambridge, Massachusetts, 1963.