Model-Based Kinematic Simulation

Elisha Sacks*
Computer Science Department
Princeton University
Princeton, NJ 08544

Leo Joskowicz
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

June 18, 1992

Abstract

We present a practical simulation program for rigid part mechanisms, such as feeders, locks, and brakes. The program performs a kinematic simulation of the behavior produced by part contacts and input motions along with a dynamical simulation of the behavior produced by gravity, springs, and friction. It describes the behavior in a compact, symbolic format and with a realistic, three-dimensional animation. The program is more efficient and informative than traditional simulators. It examines roughly 1/6 as many degrees of freedom because the kinematics module eliminates the blocked ones. It spends little time on collision detection because the kinematics module precomputes the configurations where parts collide. It covers more mechanisms than do previous model-based simulators, generates fuller behavioral descriptions, and exploits kinematics more fully. It uses a simple model of dynamics that captures the steady-state effect of forces without the conceptual and computational cost of dynamical simulation. We demonstrate that our simulation algorithm captures the workings of most mechanisms by surveying 2500 mechanisms from an engineering encyclopedia.

*This research is supported by the National Science Foundation under grant No. IRI–9008527 and by an IBM grant.

1 Introduction

This paper presents research in automating the analysis of rigid part mechanisms, such as feeders, locks, and brakes. In previous work, we developed a kinematic analysis program that takes a geometric description of the parts of a mechanism and generates a symbolic description of the space of behaviors for all possible input motions. We now describe a program that simulates the actual behavior of a mechanism for a given input motion. The program simulates the effects of part contacts, input motions, and internal forces, such as gravity and friction. It generates a compact, symbolic behavioral description and a realistic, three-dimensional animation. The simulation algorithm covers most mechanisms in an engineering encyclopedia, including ones with complex part shapes, varying part contacts, and multiple input motions.

Traditional mechanism simulators, such as ADAMS, derive the Newton-Euler or the Lagrange equations of motion, a mixture of algebraic and differential equations, and numerically integrate them for a given initial condition [4]. They always consider six degrees of freedom per part, which yields complicated equations in many variables. The user must infer the qualitative workings of the mechanism from the detailed numerical output. Some simulators assume that all contacts are permanent, hence that the equations are fixed and independent of the shapes of the parts (as in linkages). Others perform an expensive part collision test at each integration step and reformulate the equations after each contact change.

Recent model-based simulators address some of these limitations by incorporating symbolic kinematic analysis into dynamical simulators and by producing behavioral summaries [2, 3, 5]. However, they impose restrictions on the part geometry and mechanism structure, have limited interpretation capabilities, and tend to be fragile and inefficient.

Our research advances the state of the art in mechanism simulation by exploiting knowledge about the structure and function of mechanisms. The ways that mechanisms are designed constrain the shapes, motions, and interactions of their parts. We identify constraints that cover most mechanism, yet allow efficient analysis. The program handles *feasible* mechanisms: linkages, fixed-axes assemblies, or fixed-axes subassemblies connected by linkages. Linkages are one-dimensional rods permanently connected by standard joints. Fixed-axes assemblies consist of 2.5D parts that move along fixed spatial axes. The program uses a simple model of dynamics that captures the steady-state effect of forces without the conceptual and computational cost of dynamical simulation.

Our program covers more mechanisms than do previous model-based simulators, generates fuller behavioral descriptions, and exploits kinematics more fully. It is more efficient than traditional simulators. It examines roughly 1/6 as many degrees of freedom because the kinematics module eliminates the blocked ones. It spends little time on collision detection because the kinematics module precomputes the configurations where parts collide. It generates symbolic output as well as numerical simulations. It complements our previous program: it is fast and specific, whereas

that program is slower and comprehensive.

The program derives the behavior of a mechanism by kinematic simulation with simple dynamics. Kinematic simulation infers the effect of input motions on the motion of the parts of the mechanism, using the physical principle that two rigid objects cannot be in the same place at the same time. Simple dynamics models forces and friction. A force acts on a part along a translational axis or around a rotational axis, imparting a constant linear or angular velocity. The velocity drops to zero when the force stops acting; there is no inertia. Collisions among parts are inelastic. Friction constrains parts that touch along a sticky face to move in tandem along axes parallel to that face.

Simple dynamics is a qualitative theory of steady-state motion that abstracts away transient acceleration. Applying a constant force to an object actually accelerates it to a terminal velocity at which friction balances that force, but simple dynamics assumes that it reaches terminal velocity instantaneously. Ignoring transients makes simple dynamics simple and efficient, but sacrifices the predictive power of Newtonian mechanics. It suffices for mechanisms that rely on forces to push parts in certain directions. It cannot handle mechanisms in which delicate balances of forces, transient behavior, or time varying forces play a major role. The tradeoff is worthwhile because simple dynamics adequately covers most mechanisms.

We formalize kinematic simulation within the configuration space (CS) representation of mechanical engineering. Intuitively, the CS of a mechanism is the space of non-overlapping configurations of its parts. It partitions into regions of uniform part contacts separated by boundaries where part contacts change, called a *region diagram.* Each region is specified by equality and inequality constraints that express part contacts. The regions define the operating modes of the mechanism. Mode transitions occur when the configuration shifts between adjacent regions. Each path through CS defines a possible behavior of the mechanism. The regions that the path goes through provide a symbolic description of the behavior.

The kinematic simulation program traces the path that the mechanism traverses under a given input motion. It starts from the region that contains the initial mechanism configuration, constructs the segment of the path lying in that region, finds the next region that the path enters, and repeats the process. It constructs the segments by propagating the input motion through the constraints imposed by the part contacts within the regions. The simulation ends when the mechanism blocks or after a user-specified time allotment.

We implement simple dynamics forces as external motions akin to input motions, but acting infinitely faster. The difference in time scale captures the role of forces in most mechanisms. Gravity quickly drops unsupported objects onto the objects below. A spring quickly pushes a mobile object against a fixed object then maintains the contact. We assume that at most one external motion acts on a part at any time. If an input motion and an external motion both act on a part, the input motion occurs. We implement friction as constraints akin to kinematic constraints.

Figure 1 shows the relationship between the kinematic simulation program and

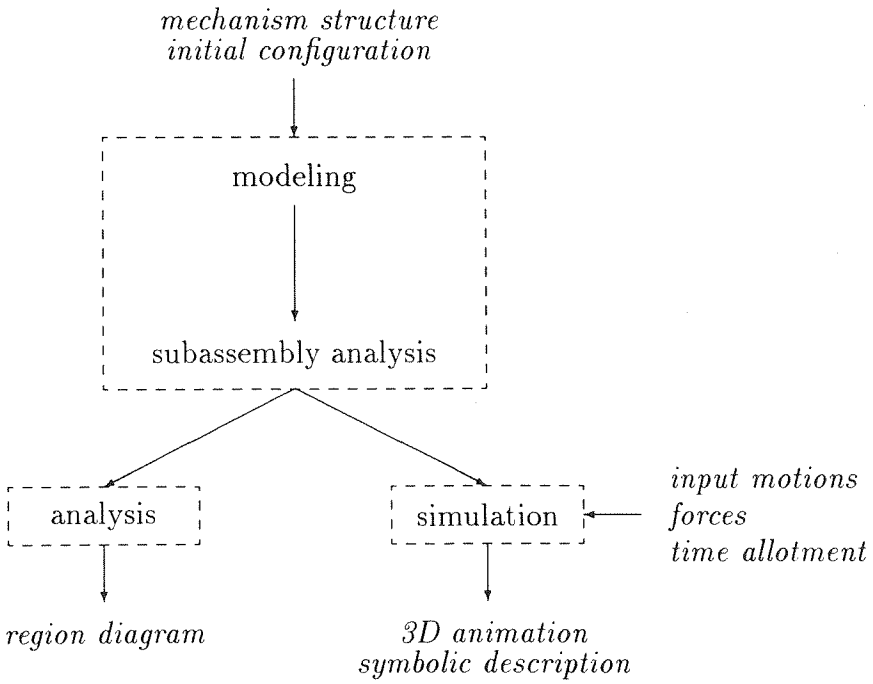


Figure 1: Mechanism analysis flowchart.

our previous kinematic analysis program [6]. The inputs to both programs include the structure and initial configuration of a mechanism. The programs share a modeling module, which decomposes the mechanism into subassemblies and finds their degrees of freedom, and a subassembly analysis module, which constructs the subassembly region diagrams. The kinematic analysis program constructs the mechanism region diagram from the subassembly diagrams and the initial configuration. The kinematic simulation program takes an input motion, internal forces, and time allotment as additional inputs and generates a symbolic description and an animation of the ensuing behavior.

2 Kinematic simulation of a feeder

We illustrate the kinematic simulation program on a mechanism that feeds blocks from a stack onto a processing table (Figure 2). The input motion rotates the driver shaft, which moves the link, which slides the piston left and right. Each time the piston slides left, one block drops onto the table due to gravity. Each time it slides right, it pushes the bottom block onto the table.

The program inputs are the part specifications and initial configurations, the gravitational forces on the blocks, and the motion "driver rotates counterclockwise". Each part is specified by its shape, coordinates, and motion type: fixed, fixed-axes, or linkage. Fixed-axes parts move along fixed spatial axes, whereas linkage parts need not.

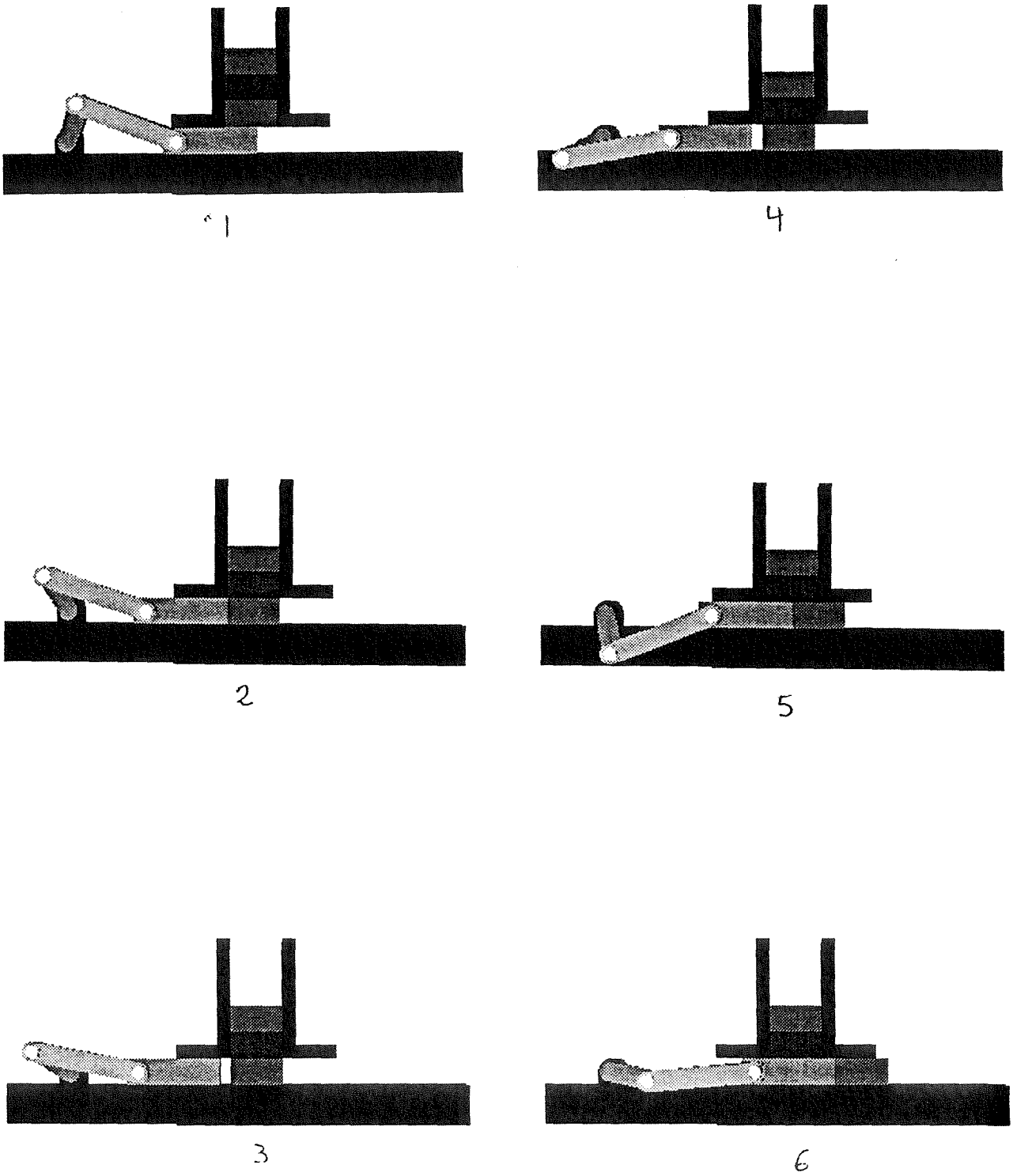


Figure 2: Configurations from a simulation of the feeder mechanism.

segment 1:
(driving-motion (driver cd))
(drives (driver cd) ((piston xp)))
(driver rotates (cd 0 2.1268))
(piston translates (xp 10 5))
(block1 stationary (xb1 12) (yb1 1))
(block2 stationary (xb2 12) (yb2 3))
(block3 stationary (xb3 12) (yb3 5))

segment 2:
(driving-motion (block3 yb3))
(drives (block3 yb3)
((block1 yb1) (block2 yb2)))
(driver stationary (cd 2.1268))
(piston stationary (xp 5))
(block1 translates (yb1 1 -1) (xb1 12))
(block2 translates (yb2 3 1) (xb2 12))
(block3 translates (yb3 5 3) (xb3 12))

segment 3:
(driving-motion (driver cd))
(drives (driver cd) ((piston xp)))
(driver rotates (cd 2.1268 3.1416))
(piston translates (xp 5 4))
(block1 stationary (xb1 12) (yb1 -1))
(block2 stationary (xb2 12) (yb2 1))
(block3 stationary (xb3 12) (yb3 3))

segment 4:
(driving-motion (driver cd))
(drives (driver cd) ((piston xp)))
(driver rotates (cd -3.1416 -2.1268))
(piston translates (xp 4 5))
(block1 stationary (xb1 12) (yb1 -1))
(block2 stationary (xb2 12) (yb2 1))
(block3 stationary (xb3 12) (yb3 3))

segment 5:
(driving-motion (driver cd))
(drives (driver cd) ((piston xp)))
(drives (piston xp) ((block1 xb1)))
(driver rotates (cd -2.1268 -0.7227))
(piston translates (xp 5 9))
(block1 translates (xb1 12 16) (yb1 -1))
(block2 stationary (xb2 12) (yb2 1))
(block3 stationary (xb3 12) (yb3 3))

segment 6:
(driving-motion (driver cd))
(drives (driver cd) ((piston xp)))
(drives (piston xp) ((block1 xb1)))
(driver rotates (cd -0.7227 0))
(piston translates (xp 9 10))
(block1 translates (xb1 16 17) (yb1 -1))
(block2 stationary (xb2 12) (yb2 1))
(block3 stationary (xb3 12) (yb3 3))

Figure 3: Symbolic description of the feeder simulation.

The fixed parts form the frame, the fixed-axes parts form fixed-axes subassemblies, and the linkage parts along with the connected fixed-axes parts form linkages. In the feeder, the driver mounting, magazine, and processing table form the frame, the driver, link, pins, and piston form a linkage, and the frame, driver, piston, and blocks form a fixed-axes subassembly.

The modeling module finds the axes of motion of the fixed-axes parts, decomposes the fixed-axes subassemblies into pairs of interacting parts, and finds their degrees of freedom. For example, it finds that the magazine allows the blocks to move up and down, but prevents them from moving left and right or from rotating. The subassembly analysis module constructs the region diagrams of the linkages and the interacting pairs. For example, it determines that rotating the driver slides the piston left and right, and that the piston supports the bottom block in the initial configuration.

The simulator derives the CS path that the mechanism traverses. Figure 2 shows one configuration from each of the first six segments in the path, which represent the first cycle of the feeder. Segment 1 lies in the initial region. The contact between the piston and the bottom of block 1 prevents the blocks from dropping. The program constructs a path segment in which the driver rotates, the link moves, the piston retracts, and the other parts do not move. The segment ends when the piston moves out from under block 1, causing a contact change. In segment 2, gravity causes the blocks to drop onto the table. In segment 3, the driver moves the piston left. In segment 4, the driver moves the piston right until it touches block 1. In segment 5, the contact between the piston and the side of block 1 enables the piston to push the block to the right. In segment 6, block 1 breaks contact with block 2 and continues right. The cycle repeats until the magazine empties.

Figure 3 shows the symbolic descriptions of the six segments of the CS path. Each description specifies the driving motion, how the driving motion propagates, and how the parts move. The motion description of a part specifies it name, its motion type, and the initial and final values of its mobile coordinates. The motion types are *stationary, translates,* and *rotates.* In segment 1, the c_d coordinate of the driver drives the x_p coordinate of the piston. The driver rotates from $c_d = 0$ to $c_d = 2.1268$, the piston translates from $x_p = 10$ to $x_p = 5$, and so on.

3 Implementation

Figure 4 shows the program organization. We focus on the simulation module in this paper, leaving the other modules to our longer paper [7]. The inputs are the region diagrams of the linkages and the fixed-axes pairs, the initial configuration, the internal forces, a sequence of input motions, and a time allotment. The output is a symbolic description of the motion path, a region diagram, and a sequence of closely spaced configurations, which the animation module outputs to a graphics workstation.

A motion is specified as a coordinate, a velocity, and a sampling rate. The path generated by a motion (x, v, s) within a region is constrained by the part contacts

Input: mechanism structure, initial configuration, input motions, and time allotment

1. modeling
2. subassembly analysis
3. kinematic simulation with simple dynamics
   (a) apply internal motions
   (b) apply next input motion
   (c) if blocked, switch to next input motion
   (d) if time and input motions remain, go to step (a)
4. animation

Output: CS path, animation, and region diagram

Figure 4: Kinematic simulation with simple dynamics.

within the region, which are represented as equalities and inequalities among the part coordinates. The motion explicitly specifies x as a linear function of time $x(t) = x_0 + vt$. The constraints determine certain coordinates as functions of x, hence of t. If these coordinates include the input of a linkage, then the linkage determines its other coordinates as a function of x. (The program handles one degree of freedom, nonredundant linkages.) The constraints may then determine additional coordinates as functions of the linkage output, hence as functions of x, and so on. The motion leaves the other coordinates constant. It ends at the maximum t that satisfies the constraints, at which time some parameter crosses the region boundary. The program traces the CS path by augmenting the region constraints with the constraint $x = x_0 + vt$, calculating t_{max}, solving the constraints for the coordinates as functions of t then substituting the values $0, s, 2s, \ldots, t_{max}$ for t. The program derives the symbolic description of the path segment from the input motion and the coordinate dependencies.

In segment 1 of the feeder animation (Figure 3), the input motion $(c_d, 1, 1/4)$ drives the mechanism. The linkage determines the linkage coordinates, including the output x_p, as functions of c_d. The constraints determine no further coordinates as functions of x_p. The program sets x_{bi} and y_{bi} $(i = 1, 2, 3)$ to their initial values, indicating that the piston does not move the blocks. In segment 2, gravity drives the mechanism, taking precedence over the driver. The constraints determine the y coordinates of the blocks and leave the other parts fixed. In segment 5, the driver drives the mechanism. The region constraints determine x_{b1} as a function of x_p, hence of c_d, because of the contact constraint between block 1 and the piston.

The program incrementally generates the regions that the motion path enters. It starts with an empty region diagram. It retrieves the regions in the current diagram

and returns those that contain the current configuration. A configuration lies in a region if it satisfies the constraints that define the region. If no current region contains the configuration, the program constructs the containing regions, adds them to the region diagram, and returns them. It maintains the regions in a hash table for essentially linear time access.

Given an input configuration outside the current region diagram, the program constructs the containing regions by composing the constraints imposed by the fixed-axes subassembly and by the linkages. It retrieves the containing regions in the region diagram of each pair of fixed-axes parts. This process normally yields one region per diagram, but yields two regions in diagrams where the configuration lies on a region boundary. Each choice of one containing region per pairwise diagram defines a potential region in the fixed-axes subassembly diagram. Intersecting the components of a potential region yields the collective kinematic constraints imposed by the fixed-axes parts. The potential region defines an actual region if the intersection is nonempty.

After finding or constructing the fixed-axes region for a configuration, the program composes it with the linkage region diagrams. Each linkage propagates constraints between its input and output coordinates. Suppose that a linkage has input x, output y, and input/output function $y = f(x)$ and that the fixed-axes constraints restrict x and y to intervals $[x_l, x_u]$ and $[y_l, y_u]$. The linkage further restricts y to the set $f([x_l, x_u])$ and x to the set $f^{-1}([y_l, y_u])$. The program calculates the linkage constraints from the linkage region diagram, which encodes the input/output function in a table.

The program uses a subset of the BOUNDER inequality prover [8] to reason about the linear inequality constraints that define the contact regions of the fixed-axes subassembly. It uses the constraint manager for three tasks: (1) to test if a potential region defines an actual region, that is if the constraints in the potential region have a solution; (2) to derive the bounds on a variable implied by a constraint set; and (3) to test if the contacts within a region determine a coordinate y as a function of x, that is if the upper and lower bounds of y in terms of x coincide.

4 Evaluation

The feeder example shows that kinematic simulation with simple dynamics vividly and efficiently captures the workings of a realistic mechanism. The program generates a CS path containing 90 configurations and a region diagram containing 16 regions. It runs in 10 minutes on a DEC workstation and animates the resulting 90 snapshots in real time on an IRIS workstation. The program constructs 9 region diagrams for pairs of fixed-axes parts: 3 block/block diagrams with 6 regions apiece, 3 piston/block diagrams with 6 regions apiece, and 3 block/frame diagrams with 2 regions apiece. It constructs a single linkage region diagram containing 301 configurations. These pairwise regions yield 373,248 potential regions for the overall mechanism. The program examines 48 of these potential regions (0.01%) in tracing the CS path, whereas our

previous program examines 2115 potential regions (0.5%) in constructing the full 217 region diagram. Thus, simulating the feeder is 50 times less work than constructing its full region diagram.

We have tested our program on a dozen realistic examples, including the feeder, a transmission, a rim lock, and a shoe brake. Each example illustrates different aspects of kinematic simulation with simple dynamics. The feeder has many moving parts, contains a linkage, and uses gravity. The transmission has complex part shapes and interactions. The rim lock has many regions in its region diagram and contains a spring that opposes the input motion. The shoe brake has simultaneous input motions, springs, and friction. They all have varying contacts, multiple operating regions, high-dimensional CSs, and multiple degrees of freedom. All simulations run in under 10 minutes and explore a very small fraction of the CS.

We surveyed over 2500 mechanisms in Artobolevsky's four-volume *Mechanisms in Modern Engineering Design* [1] to determine the percentage of practical mechanisms covered by kinematic simulation with simple dynamics and to identify significant exceptions. We found that 59% of the mechanisms are feasible mechanisms, that 79% are covered by simple dynamics, and that 48% are both feasible and covered by simple dynamics. The details appear in our longer paper [7].

We examined the accompanying text descriptions to determine if simple dynamics captures the workings of the mechanism. The descriptions focus on the aspects of the mechanisms relevant to their function and abstract away other aspects. We deem that simple dynamics covers a mechanism if it matches the text description of the forces and frictions. For example, the text describes the workings of the feeder as follows. (We have changed the part names to ours for clarity.)

> Workpieces drop from the magazine onto the processing table. A mechanism which is not shown periodically rotates the driver through one complete revolution, beginning from its extreme left-hand position. Rotating about a fixed axis, the driver, by means of the connecting link, reciprocates the piston which ejects the bottom workpiece into a chute not shown. When the driver returns to its extreme left-hand position, the next workpiece drops onto the processing table (Vol 2. p. 592).

This description captures the function of the feeder without specifying the rate at which the blocks drop, the effect of friction, or the transient accelerations. It shows that the simulation in Figure 2 and its symbolic description in Figure 3 appropriately capture the workings of the feeder.

5 Conclusion

This paper presents a practical simulation program for rigid part mechanisms. The simulation captures the kinematic constraints imposed by part contacts and input

motions along with the dynamical constraints imposed by gravity, springs, and friction. The program represents the kinematics as a partition of the mechanism CS into regions of uniform motion. It generates the simulation by tracing the CS path that the mechanism traverses under the input motions and dynamical constraints. It produces a symbolic description and a three-dimensional animation of the simulation.

Our simulation algorithm is limited by the types of mechanisms it can analyze and by the dynamical phenomena it can model. In a previous paper [6], we describe methods for extending the kinematic coverage from 59% to about 90% while maintaining reasonable program complexity and computational efficiency. These extensions would raise the overall coverage from 48% to about 72%, since simple dynamics covers 80% of the mechanisms. Extensions to simple dynamics include improving modeling, steady-state dynamics, and full dynamical simulation. Improving modeling invests increased modeling effort for ease of analysis. An example is replacing 2D springs by 1D springs where possible. Steady-state analysis abstracts away transient acceleration and vibration, but derives the precise steady-state effect of forces, masses, moments of inertia, and friction. It suffices for friction mechanisms, mechanisms with competing forces or inertia such as governors and tripping mechanisms, and brakes. Full dynamical analysis is necessary for correctly simulating mechanisms not covered by simple dynamics and for accurately simulating covered mechanisms. An example of such mechanism is a clock escapement, since the precise transient behavior determines the exact interval between clock ticks.

Kinematic simulations with simple dynamics sets the stage for full dynamical analysis. Modeling identifies the relevant CS coordinates and possible part interactions. We can formulate the full dynamical equations in CS coordinates instead of in part coordinates, reducing by a factor of six the number of equations and making them less stiff. Subassembly analysis and simulation compute part interactions and coordinate dependencies. We need not test for part collisions at each integration step because the region diagram specifies the configurations where parts collide. The simulator can find the initial region, integrate the equations within the region bounds, then shift to the next region. This procedure should combine the robustness and efficiency of kinematic simulation with the accuracy of traditional simulation.

We believe our research serves the larger goal of automating many aspects of mechanical engineering, including design, validation, and cataloging. Engineers work with concise descriptions of mechanisms that specify only the information relevant to the intended behavior. A typical description consists of a blueprint of the mechanism geometry and of an English explanation of the relevant dynamics. Engineering programs should generate and understand these descriptions in order to communicate with users and with engineering databases. We demonstrate that the symbolic output of our program matches these descriptions for a large class of mechanisms. We hypothesize that the descriptions set the stage for more detailed analysis and provide a computational basis for other engineering tasks, such as designing mechanisms that achieve specified functions.

References

[1] Artobolevsky, I. *Mechanisms in Modern Engineering Design*, volume 1–4. (MIR Publishers, Moscow, 1979). English translation.

[2] Cremer, J. An architecture for general purpose physical system simulation—integrating geometry, dynamics, and control. Technical Report 89-987, Cornell University, Apr. 1989.

[3] Gelsey, A. Automated physical modeling. in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[4] Haugh, E. (Ed.). *Computer Aided Analysis and Optimization of Mechanical System Dynamics.* (Springer-Verlag, 1984).

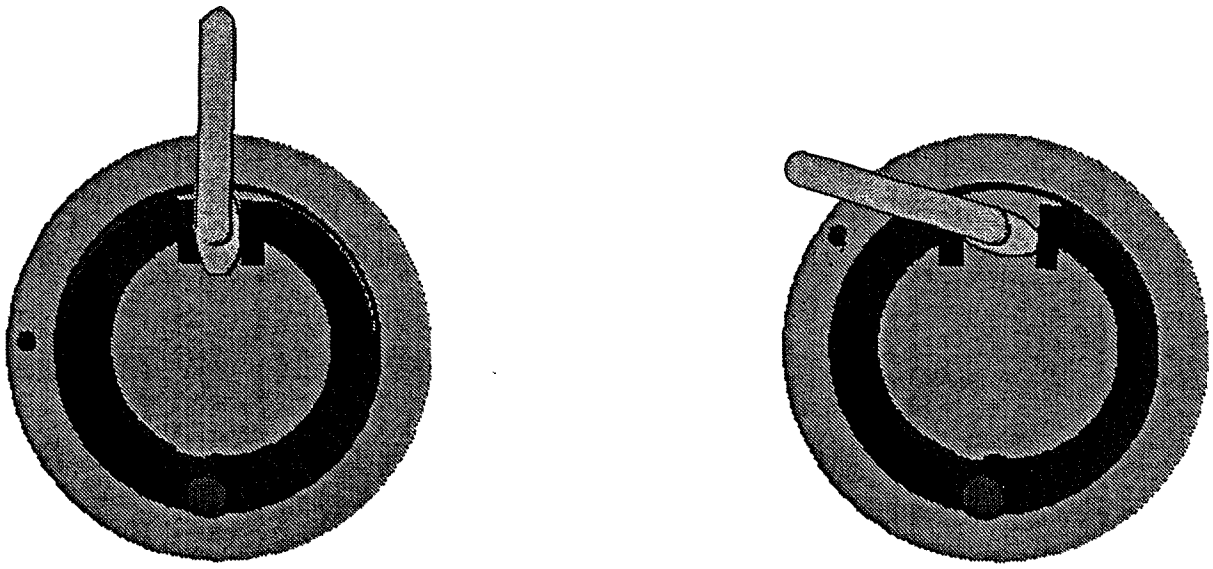
[5] G. A. Kramer, Solving geometric constraint systems. in: *Proceedings of the 8th National Conference on Artificial Intelligence*, Boston (1990) 708–714.

[6] Joskowicz, L. and Sacks, E. P. Computational kinematics. *Artificial Intelligence* **51** (1991) 381–416.

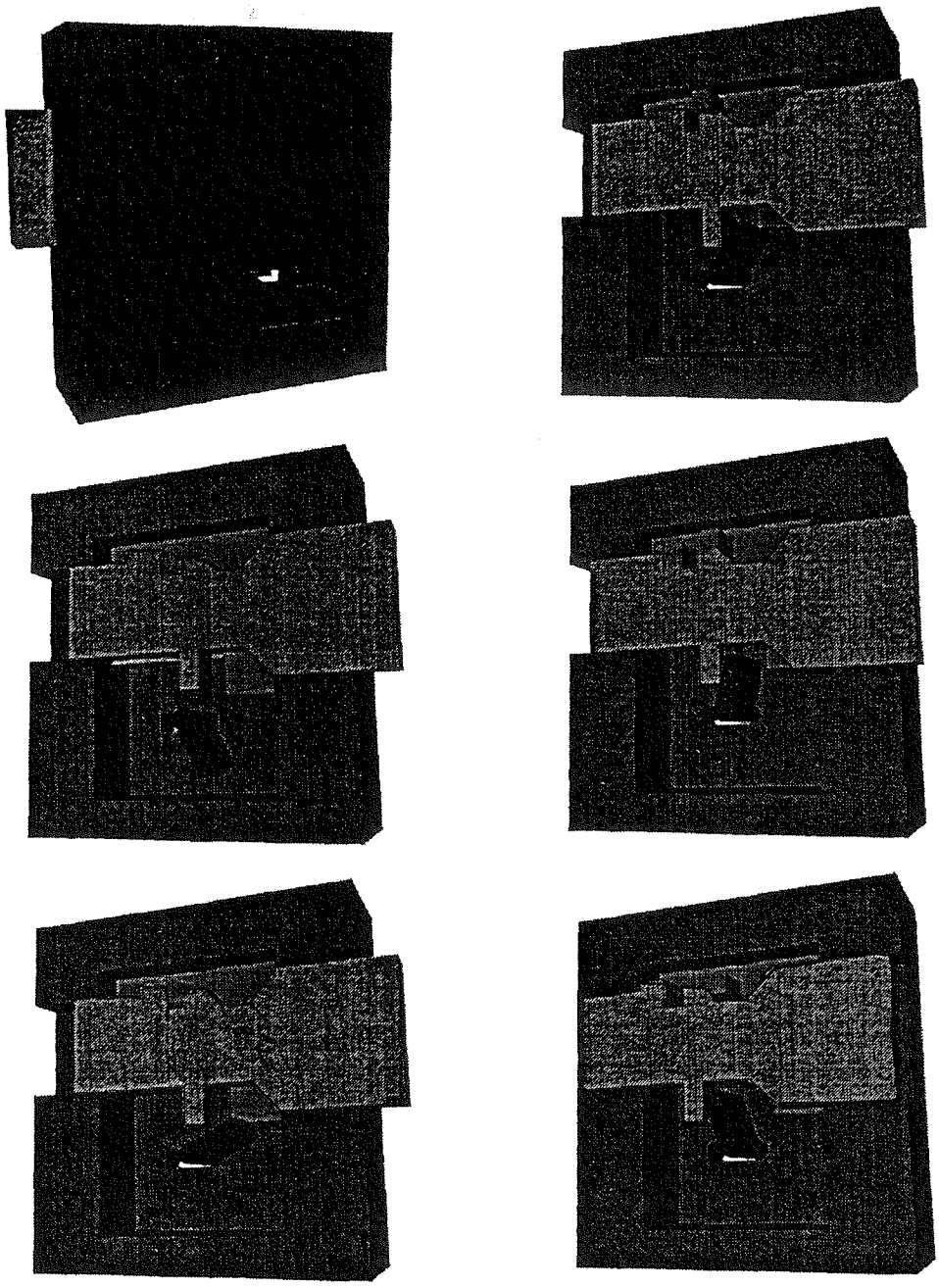
[7] Sacks, E. P., and Joskowicz, L. Mechanism Simulation using Configuration Spaces and Simple Dynamics. Technical Report, March 1992.

[8] Sacks, E. P. Hierarchical reasoning about inequalities. in: *Proceedings of the National Conference on Artificial Intelligence*, 1987.

Appendix: more examples



Shoe brake: braking sequence of a shoe brake. The brake consists of a hollow drum rotating around its center, two spring-loaded shoes mounted at their edges to a fixed pin, and an activating lever. In the initial (left) configuration, the drum rotates freely and the cam is rotated clockwise. As the lever is turned, it pushes open the shoes. When the shoes touch the internal surface of the drum, friction makes the drum stop rotating.



Rim lock: unlocking sequence of a rim lock. The lock consists of a frame, a key, a rim, a latch, and a spring (not shown) that pushes the latch against the rim. The top left snapshot shows a back view of the initial locked configuration. The following snapshots show a front view of an unlocking sequence. In the initial configuration, the latch blocks the horizontal motion of the rim, thus barring unauthorized entry. As the key rotates counterclockwise, it raises the latch (countering the effect of the spring), disengages the rim, and pushes the rim back. When the key breaks contact with the latch, the spring pushes the latch against the rim, causing the latch to follow the contour of the rim. Rotating the key clockwise (not shown) pushes the rim out, which locks the door.

	feeder	transmission	rim lock	shoe brake
moving parts	8	8	3	4
part faces	77	446	80	60
linkages	1	0	0	0
CS dimension	17	9	4	4
dynamics	gravity	none	spring	friction and springs
input motions	1	1	1	2
mechanism DOF	4	2	4	2
potential regions	373,248	31,360,000	2352	64
explored regions	2115	49	337	16
nonempty regions	217	13	79	16
traversed regions	48	2	22	1
quadrangles	1168	3626	217	8350
snapshots	88	32	87	6
runtime (secs.)	487	2	49	12

Summary of the analyses of four mechanisms *. The first four rows characterize the structure of each mechanism: the number of moving parts; the number of part faces, which measures geometric complexity; the number of linkages; and the CS dimension, which equals the number of potential degrees of freedom. The next two rows describe the forces and the input motions. The next five rows describe the region diagram of the mechanism: the maximal region dimension, which equals the actual degrees of freedom; the number of potential regions, which equals the product of the number of reachable regions in the pairwise region diagrams; the number of regions explored; the number of nonempty regions, which represent realizable configurations; and the number of regions traversed during kinematic simulation. The last three rows are the number of quadrangles in the linearization, which measures graphical complexity; the number of snapshots in the animation; and the time required to produce the kinematic simulation and the animation.

** a color videotape of these animations is available in VHS format*