MECHANISM SIMULATION WITH CONFIGURATION SPACES AND SIMPLE DYNAMICS

Elisha Sacks

CS-TR-367-92

March 1992

Mechanism Simulation with Configuration Spaces and Simple Dynamics

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

March 9, 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 much more efficient than traditional simulation. 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 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.

Contents

1 Introduction

This paper presents research in automating the analysis of rigid part mechanisms, such as feeders, locks, and brakes. Mechanism analysis derives the kinematics and the dynamics of a mechanism. The kinematics specify the behaviors that are consistent with the shapes of the parts, the contacts among parts, and the possible input motions. The dynamics specify the behaviors that are consistent with the forces acting on the parts, such as gravity, springs, and friction. Together, they determine the exact behavior for each input motion and initial configuration of the parts. 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 possible behaviors. 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 forces. It produces a compact, symbolic behavioral description and realistic, three-dimensional animation.

Our research advances the state of the art in mechanism analysis. The simulation algorithm covers most mechanisms in an engineering encyclopedia, including ones with complex part shapes, varying part contacts, and multiple input motions. The dynamical analysis yields a fuller description of the workings of mechanisms than does purely kinematic analysis. The simulation describes the behavior for specific input motions, thus complementing a description of the space of possible behaviors. The output combines the vividness of graphics with the precision of symbolic data.

The program is much more efficient than traditional simulation programs. 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 uses a simple model of dynamics that captures the steady-state effect of forces without the conceptual and computational cost of dynamical simulation. It analyzes in minutes mechanisms that take hours with general purpose dynamical simulators.

This 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.

We derive the behavior of a mechanism by kinematic simulation with simple dynamics. Kinematic simulation infers the effect of input motions on the parts of the mechanism, using

the physical principle that two rigid objects cannot be in the same place at the same time. If an input motion pushes part A against part B, part B will move rather than overlap part A. At each instant, the configuration (position and orientation) of part A and the shape of the contact surface determine the configuration of part B. By the same principle, part B can move other parts that it touches. The overall behavior of the mechanism depends on all the contacts among its parts.

We formalize this reasoning within the configuration space representation of mechanical engineering. Intuitively, the configuration space 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 configuration space 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.

Kinematic simulation captures the effect of input motions and part contacts on the behavior of mechanisms, but misses the effect of forces. Simple dynamics captures most of these effects without the overhead of full dynamical simulation. It assumes that forces impart fixed motions to parts, which act infinitely faster than input motions. Simple dynamics is a qualitative theory of steady-state motion that abstracts away transient acceleration. Applying a constant force to the center of mass of 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. We implement simple dynamics as motions that take precedence over input motions.

Figure 1 shows the relationship between the kinematic simulation program and our previous kinematic analysis program. 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.

The rest of the paper is organized as follows. We conclude this section with an informal

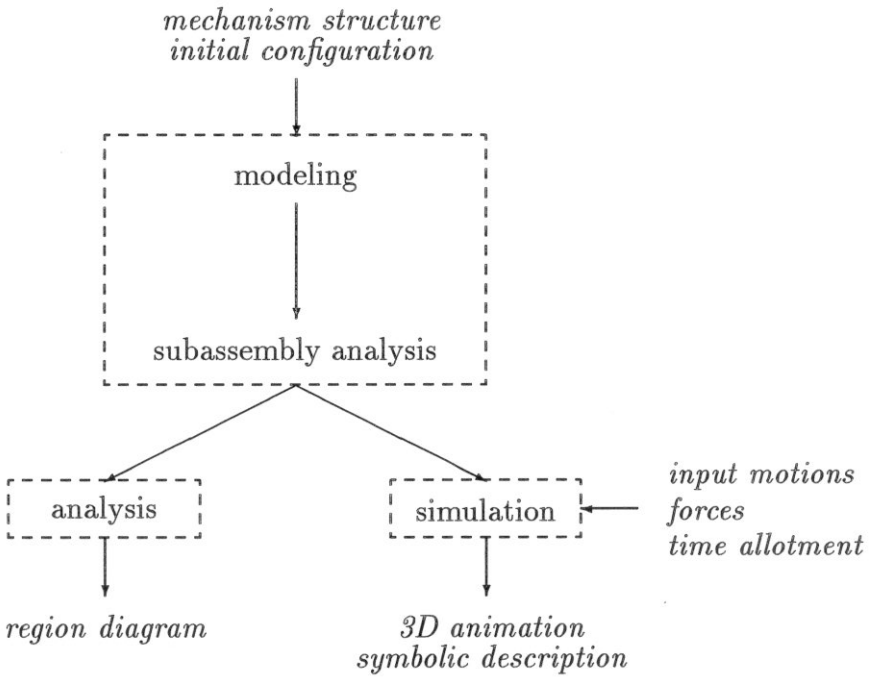


Figure 1: Mechanism analysis flowchart.

illustration of the kinematic simulation program. In Section 2, we review the configuration space formalization of kinematics, identify a class of mechanisms for which kinematic analysis is feasible, show that the feasible class covers most mechanisms, and describe the kinematic simulation algorithm. In Section 3, we define simple dynamics and show that it captures the dynamical behavior of about 80% of mechanisms from an engineering encyclopedia. In Section 4, we describe an efficient program that performs kinematic simulation of feasible mechanisms with simple dynamics. In Section 5, we demonstrate the program on additional examples. We conclude with a review of related work and a discussion of future work.

1.1 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 and motion type: fixed, fixed-axes, or linkage. Fixed-axes parts move along fixed

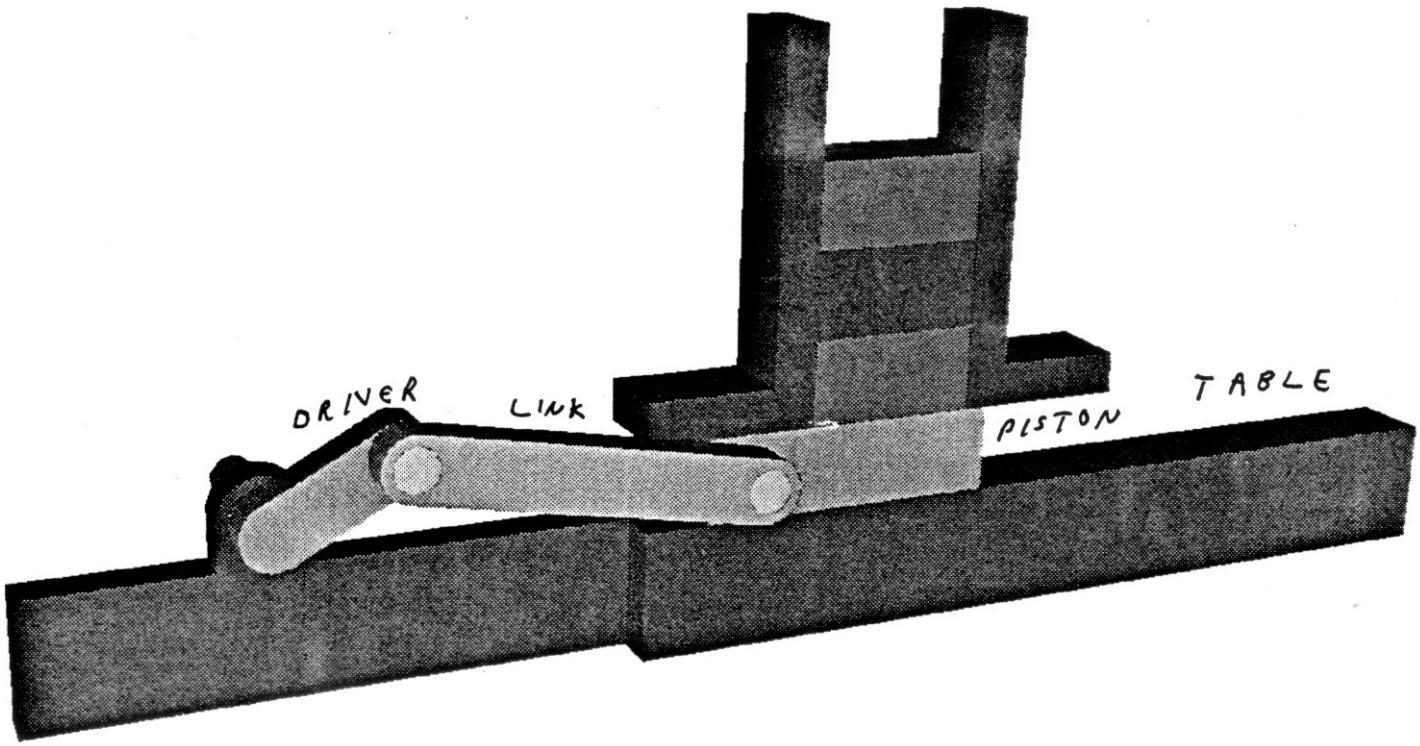


Figure 2: The feeder mechanism.

spatial axes, whereas linkage parts need not. 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 configuration space path that the mechanism traverses. Figure 3 shows one configuration from each of the first six segments in the path, which represent the first cycle of the feeder, and 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 4 shows the symbolic descriptions of the six segments of the path. Each description specifies the driving motion, how the driving motion propagates, and how the parts move. 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.

2 Kinematic simulation

Kinematics studies the ways in which assemblies of rigid parts move in space. An isolated part can be in any configuration and is free to translate and rotate in any direction. But a part in an assembly can only occupy configurations that do not overlap other parts and can only move along or away from other parts. Parts moving together must satisfy the constraints imposed by their touching vertices, edges, and faces. In this section, we summarize the main concepts of kinematics and define kinematic simulation. Our previous paper [10] contains a detailed discussion of kinematics and its role in mechanism analysis.

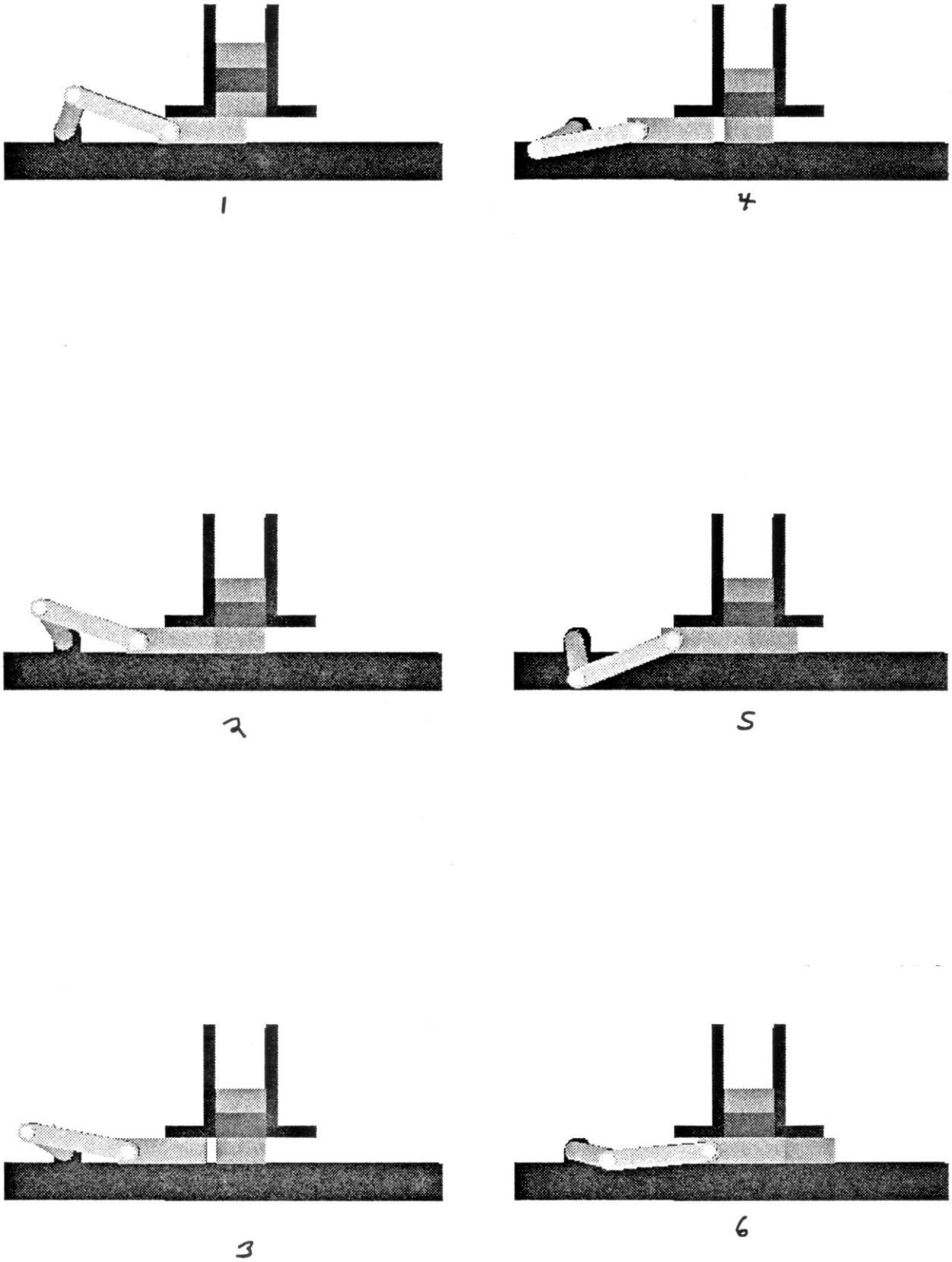


Figure 3: Configurations from a simulation of the feeder mechanism.

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 4: Symbolic description of the feeder simulation.

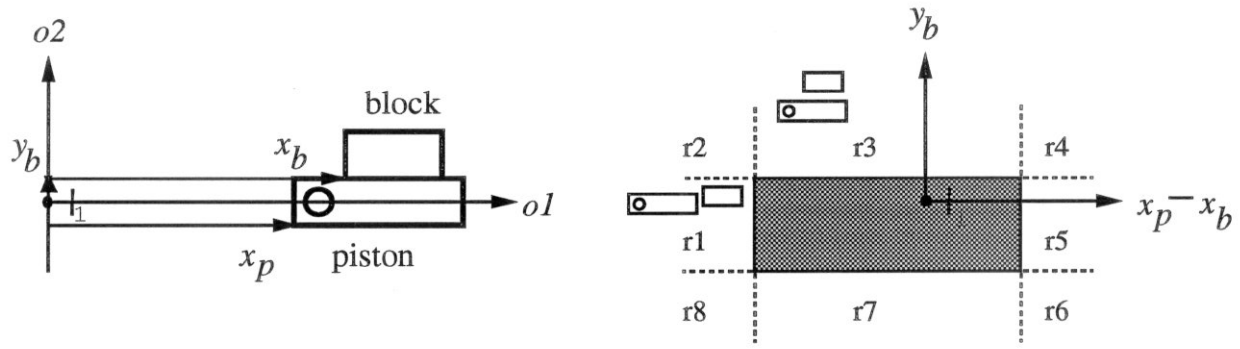


Figure 5: The piston/block pair and its region diagram. Shading indicates blocked space. Dashed lines indicate region boundaries.

2.1 Kinematics

We study the kinematics of mechanisms within the standard configuration space (CS) representation of mechanical engineering. The configuration of each part is specified as a six element vector, for example by the position and Euler angles of a local coordinate frame with respect to a global coordinate frame. The Cartesian product of the vectors forms the mechanism configuration space. The mechanism CS decomposes into *blocked space* where parts overlap and *free space* where no parts overlap. Only configurations in free space are physically realizable. The boundary of free space consists of the configurations where parts touch. All physically realizable motion of the parts take place in free space. A motion defines a continuous curve in free space consisting of the succession of configurations that the mechanism traverses.

We describe the kinematics of a mechanism with a partition of its free space into motion regions, called a region diagram. The region diagram is a complete and concise description of the space of possible behaviors. The regions define the operating modes of the mechanism. A realizable motion of the mechanism translates into a path in the region diagram. The path decomposes into a sequence of segments, each of which lies entirely in one region. The segments represent the uniform portions of the motion, whereas their boundary configurations represent shifts in operating mode.

We illustrate these concepts for the piston and a block from the feeder (Figure 5). The piston translates horizontally along axis $o1$ with coordinate x_p. The block can either translate horizontally along axis $o1$ with coordinate x_b or translate vertically along axis $o2$ with coordinate y_b. Figure 5 shows a region diagram for the pair. The diagram projects the 3D CS into the $(x_p - x_b, y_b)$ space for simplicity. Free space regions r_1 and r_3 are labeled by typical configurations. The upper horizontal CS boundary consists of the configurations where the block rests on the piston and the left hand vertical boundary consists of the con-

figurations where the piston touches the block on the left. The piston/block configuration moves from r_3 across the vertical boundary and into r_2 when the piston moves left from its initial configuration. It moves from r_2 across the horizontal boundary and into r_1 when the blocks drop.

We obtain the region diagram of a mechanism by composing the contact constraints of its parts. The condition that a pair of parts does not overlap translates into algebraic equations in their coordinates. We can formulate the equations for the interacting pairs in the mechanism, construct pairwise region diagrams, and compose them into a mechanism region diagram. This process is inherently intractable for general mechanisms. We obtain a practical algorithm by restricting the shapes, motions, and interactions of parts. The resulting class of *feasible mechanisms* covers most mechanisms, yet supports efficient kinematic analysis.

The first type of feasible mechanism is the standard *linkage* of mechanical engineering. A linkage consists of parts permanently attached to each other by joints. The parts only interact via the joints. There are six types of joints, each of which constrains the attached parts to move in a simple manner. Linkages are modeled by one-dimensional rods whose coordinates are coupled by joint equations.

The second type of feasible mechanism is a *fixed-axes mechanism* whose parts move along fixed spatial axes. Parts are 2.5D, meaning that each part consists of a finite number of *slices* over a reference plane. A slice is the Cartesian product of an interval orthogonal to the reference plane and one or more closed curves in the reference plane. The closed curves consist of line segments and circular arcs. Each part can translate along axes parallel or orthogonal to the reference plane and can rotate around an axis orthogonal to it. We require that the kinematic pairs in fixed-axes mechanisms have CSs of dimension two or lower.

The third type of feasible mechanism is a collection of fixed-axes mechanisms connected by linkages. The linkages interact with the fixed-axes mechanisms via permanent connections to fixed-axes parts. The feeder is an example of such mechanism. The fixed-axes subassembly consists of the frame, driver, piston, and blocks, while the linkage consists of the driver, link, pins, and piston. The subassemblies interact via the driver and the piston.

Most real-world mechanisms are feasible. In a survey of 1912 mechanisms from a mechanical engineering encyclopedia [1], we found that 58% of mechanisms are feasible: about 30% are linkages, 22% are fixed axes, and 6% are fixed-axes connected by linkages.

2.2 Simulation

Although region diagrams fully specify the kinematics of mechanisms, they encode the information symbolically as complex, multivariate, algebraic equations. Inferring the behavior of a mechanism from this symbolic format takes some work for trained engineers and is daunting for novices. The region diagram often contains more information than users need: it encodes the entire mechanism CS, which represents the behaviors for all input motions, whereas users generally care about only one or two input motions. For example, the region

diagram of the feeder encodes a complex 4D subspace of a 12D space. The four degrees of freedom describe the motions of the feeder assembly and the three blocks under four independent input motions. We care only about the 1D subset of the region diagram in which the sole input is rotation of the driver.

Kinematic simulation derives the behavior of a mechanism under a specific input motion, rather than the space of possible behaviors. The input motion causes the mechanism to trace a path through its CS. Kinematic simulation generates that path and describes it symbolically and with a three-dimensional animation of the corresponding behavior. It generates only the regions that the mechanism traverses, which comprise a small portion of the full region diagram.

We could perform kinematic simulation by constructing the entire mechanism region diagram then tracing the path generated by the input motions. This approach is conceptually simple, but computationally impractical for complicated mechanisms. Instead, we produce the region diagram and the simulation simultaneously. We start by constructing the region containing the initial configuration. We trace the path generated by the first input motion from the initial configuration to the region boundary. We then construct the next region that the path enters and repeat the process. If the mechanism cannot move, we apply the next input motion or stop if none remains.

We specify a motion as a coordinate, a velocity, and a sampling rate. In the feeder example, the input motion is $(c_d, 1, 1/4)$, meaning that the driver rotates one radian per second and is sampled once every quarter second during animation. The path generated by a motion (x, v, s) within a region is constrained by the part contacts 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 $x(t) = x_0 + vt$. If the contact constraints determine a coordinate y as a function of x, $y = f(x)$, then the motion implicitly specifies y as a function of time $y = f(x_0 + vt)$. 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. After calculating the contact constraints and the exit time, we can trace the path without collision detection or other expensive computations.

For example in segment 4 of the feeder animation (Figures 3 and 4), the motion $(c_d, 1, 1/4)$ rotates the driver, moves the link, and slides the piston to the right because the coordinates of these parts are determined by c_d, but leaves the blocks in place because their coordinates are independent of c_d. The path leaves this region when the piston touches block 1. In segment 5, the same motion pushes block 1 to the right because the contact between the piston and block 1 determines x_{b1} as a function of c_d. The path leaves this region when the piston begins to slide left, breaking contact with block 1.

3 Dynamics

Kinematic simulation captures the effect of input motions and part contacts on the behavior of mechanisms, but misses the effect of forces, such as gravity, springs, and friction. Forces play an important role in the workings of many mechanisms. In the feeder, the blocks drop onto the table because of gravity as soon as the piston retracts, whereas kinematic simulation mistakenly shows the blocks suspended in the air. We must augment kinematic simulation with force analysis to obtain the correct behavior.

The standard tools for force analysis are classical mechanics and numerical analysis. If we model the parts of a mechanism as rigid objects, we can simulate its dynamics by formulating its laws of motion as mixed algebraic and ordinary differential equations and numerically integrating them starting from its initial configuration. Setting up the simulation is complicated and running it can take hours of computer time for realistic mechanisms. The simulation runs slowly because the mixed equations are stiff and because at each time step it must test whether any contact change has occurred since the previous time step, and if so reformulate the equations of motion. In the feeder, the equations change when the piston breaks contact with the bottom block, when the blocks hit the floor, and when the piston resumes contact with the bottom block. We discuss some of the extensive research on dynamical simulation in the review of literature.

We have developed a simple dynamics for mechanisms that captures the effect of forces without the conceptual and computational cost of dynamical simulation. Simple dynamics formalizes forces as motions akin to input motions. In the feeder, gravity imparts a negative velocity to y_{bi} $(i = 1, 2, 3)$, which drops the blocks onto the table. Simple dynamics is a qualitative theory of steady-state motion that abstracts away transient acceleration. Applying a constant force to the center of mass of 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. Applying a constant torque around an axis of rotation of an object actually accelerates it to a terminal angular velocity, but simple dynamics assumes that it reaches terminal velocity instantaneously.

Simple dynamics achieves its simplicity and efficiency by sacrificing the predictive power of Newtonian mechanics. It predicts the directions that parts move, but not their exact motion paths. The tradeoff is worthwhile for mechanism analysis because simple dynamics adequately describes the workings of most mechanisms. Simple dynamics 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. In the next section, we describe simple dynamics in detail and show how to integrate it with kinematic simulation. In the following section, we justify simple dynamics empirically by demonstrating that it reproduces most behavioral descriptions in a mechanical engineering encyclopedia.

3.1 Simple dynamics

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. We model gravity as a simple dynamics force, which implies that falling objects reach terminal velocity instantaneously. We model a spring as a simple dynamics force. We require that one end be fixed and that the other end never oscillate.

Friction constrains the relative motion of touching parts. Every surface has a coefficient of friction of 0 or 1, called smooth and sticky. Friction acts solely between touching pairs of sticky faces. If two parts touch along sticky faces, they move in tandem along axes parallel to those faces. We represent the coupling with a constraint on the coordinates, u and v, of the parts along these axes. If both parts translate along parallel axes or rotate around the same axis, the constraint is $u - v = \textit{constant}$. Otherwise, the constraints are $u = \textit{constant}$ and $v = \textit{constant}$. In the feeder, if the table and the piston were sticky, their x coordinates would satisfy the constraint $x_p - x_t = \textit{constant}$, but their y and z coordinates would not be constrained.

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 frictions as constraints akin to kinematic constraints.

We extend the kinematic simulation algorithm to apply all external motions before applying each input motion. An external motion does nothing if a constraint blocks it or an input motion opposes it. Otherwise, it moves the mechanism just like an input motion. In the feeder animation, gravity causes no motion initially because the piston supports the blocks. The input motion moves the piston left until it clears block 1 and a region transition occurs. Gravity then makes the blocks drop onto the table. This motion creates a region transition, after which the driver continues moving the piston left.

3.2 Coverage

We surveyed an encyclopedia of mechanisms to determine the percentage of practical mechanisms covered by kinematic simulation with simple dynamics and to identify significant exceptions. As in our previous paper [10], we chose Artobolevsky's four-volume *Mechanisms in Modern Engineering Design* [1] because of its scope, uniform format, and comprehensiveness. In the previous paper, we found that 59% of the mechanisms are feasible, hence amenable to kinematic simulation. Our current survey shows that 79% of the mechanisms are

		Coverage		
Volume	total	dynamics	kinematics	both
Lever mechanisms 1	685	544	443	368
Lever mechanisms 2	732	603	486	414
Gear mechanisms	399	324	168	135
Cam mechanisms	283	159	98	61
Pairs (all volumes)	549	466	358	296
Total	2648	2096	1553	1274
Percentage	—	79%	59%	48%

Table 1: Survey of mechanisms.

covered by simple dynamics and that 48% are both feasible and covered by simple dynamics. Hence, kinematic simulation with simple dynamics covers roughly half of all mechanisms.

The survey focuses on the motions of the parts under the specified input motions and forces. We use the mechanism descriptions accompanying each figure as the guideline to determine if simple dynamics captures the workings of the mechanism. The description focuses on the aspects of the mechanism relevant to its function and abstracts away other aspects. We deem that simple dynamics covers the 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 effects of friction, or the transient accelerations. It shows that the simulation in Figure 3 captures the workings of the feeder.

The assessment that simple dynamics covers a mechanism is subjective, since it relies on our interpretation of Artobolevsky's text, whereas the kinematic classification of mechanisms is objective, since it relies solely on precise mathematical definitions. Thus, the results of the dynamics survey are weaker than those of the kinematics survey. We compensate for this weakness by only classifying a mechanism as covered if we are certain that it is.

We surveyed 2648 mechanisms. With the exception of one-dimensional springs, we excluded mechanisms with flexible parts, such as belts and chains, because our dynamical model assumes rigid parts. We also excluded mechanisms for which kinematic simulation is

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

1. modeling
2. subassembly analysis
3. kinematic simulation with simple dynamics
   (a) find current region
   (b) traverse path segment in current region
   (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 6: Kinematic simulation with simple dynamics.

meaningless, such as curve generators and analog computers. Our model of dynamics does not account for vibrations, deformations, lubrication, wear and tear, stress, or tolerancing. Table 1 summarizes the results.

4 Implementation

We have defined kinematic simulation with simple dynamics and demonstrated that it captures the behavior of most mechanisms. We now present an efficient implementation for feasible mechanisms. The program covers about half of all mechanisms both dynamically and kinematically. Figure 6 shows the program organization. The inputs are a mechanism, an initial configuration, a sequence of input motions, and a time allotment. The mechanism is specified by its frame parts, fixed-axes parts, linkages, external motions, and sticky surfaces. The outputs are a symbolic description of the CS path that the mechanism traverses, an animation, and a region diagram of the regions that the path goes through.

The modeling module identifies the axes of motion, motion parameters, and possible part interactions. The subassembly analysis module constructs the subassembly region diagrams. The kinematic simulation module generates the CS path that the mechanism traverses under the input motions during the allotted time, along with a region diagram of the regions that the path goes through. The animation module transforms the CS path into a three-dimensional animation and displays it on a graphics workstation.

4.1 Modeling and subassembly analysis

The modeling and subassembly analysis modules extend our previous implementation from fixed-axes mechanisms to general feasible mechanisms consisting of fixed-axes subassemblies connected by linkages. The modeling module decomposes the fixed-axes subassemblies into interacting kinematic pairs, identifies their axes of motion, and constructs their region diagrams, as explained in our previous paper [10]. Each region diagram consists of convex regions whose boundaries are accurate linearizations of the true contact curves.

The program handles linkages with one degree of freedom, one input coordinate, and one output coordinate. Such a linkage satisfies n equalities in its $n+1$ mobile coordinates, which jointly define its configuration as a function of its input. Its region diagram contains a single region, which is the graph of the configuration function.

The program derives the equations of the linkage by the standard Cartesian coordinates method [13]. It eliminates the coordinates that are linear functions of other coordinates, such as the pin coordinates. It calculates the configuration at evenly spaced points throughout the input range, using the AUTO continuation package [4] on the remaining equations, and stores the results in a table. The continuation takes seconds. The elimination step is worthwhile because AUTO's running time grows cubically in its input size. The program calculates the configuration at intermediate points by table lookup followed by linear interpolation. A lookup takes logarithmic time in the length of the table, which amounts to milliseconds. The table and the lookup algorithm jointly define the region diagram of the linkage.

The feeder linkage consists of the driver, link, pins, and piston. The input is the rotation of the driver, c_d, the output is the translation of the piston, x_p, and the reduced Cartesian equations are

$$\begin{aligned} x_l &= 3\cos c_d \\ y_l &= 3\sin c_d \\ x_p &= x_l - 1 + 8\cos c_l \\ 0 &= y_l + 8\sin c_l. \end{aligned}$$

where x_l, y_l and c_l are the motion parameters of the link. A configuration table with 301 entries covers the input range of $(-\pi, \pi)$ with accuracy exceeding 10^{-5}. Figure 7 shows the input/output function obtained from the table by linear interpolation. For a full rotation of the driver, the piston travels 6 units, from 4 to 10.

4.2 Kinematic simulation with simple dynamics

The kinematic simulation module traces the CS path that the mechanism traverses under the input motions during the allotted time. Figure 8 shows the algorithm. The program generates the path by repeatedly constructing the region containing the current configuration, applying the external motions, and applying the current input motion. The resulting

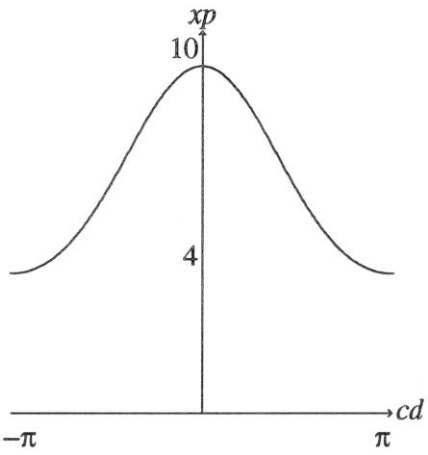


Figure 7: The feeder linkage and its input/output function.

path segment consists of a sequence of configurations and of a symbolic description of the motion within the region. The program shifts to the next input motion when the current one yields no motion. It stops when the time allotment is exhausted or no input motions remain.

Boundary configurations complicate the basic algorithm. The program must consider the possibility that the configuration c will instantaneously leave the containing region and enter an adjacent region whose closure contains c. For example, consider a 1D CS with regions $x < 0$ and $x \geq 0$. The configuration 0 lies in $x \geq 0$, but the motion $(x, -1, 1)$ moves it into $x < 0$. In the feeder, the configuration where the piston breaks contact with the bottom block lies in the initial region, but the input motion moves it into the next region where the blocks fall. The program handles boundary configurations by retrieving (in steps 3 and 4) every region whose closure contains c then selecting a region in which motion occurs.

The basic algorithm handles mechanisms that receive one input motion at a time. The program also handles mechanisms with simultaneous input motions, such as a transmission with a gear shift, a clutch, and a drive shaft. It reduces simultaneous motions to one motion by introducing by parameterizing them with respect to a new parameter. For example, it converts the motions $(x, 1, 1)$ and $(y, 2, 1)$ to the motion $(t, 1, 1)$ with the parameterization $x = t$ and $y = 2t$. Section 5 contains an example of simultaneous input motions. The rest of the paper ignores them for simplicity.

The path segment generated by a motion (x, v, s) within a region depends on the region constraints and on the linkages, which jointly determine a subset of the mobile coordinates as functions of x. The constraints determine certain coordinates as functions of x. If these coordinates include the input of a linkage, then the linkage determines its other coordinates as a function of x. The constraints may then determine additional coordinates as functions of the linkage output, hence as functions of x, and so on. The program finds the dependent

1. Initialize c to the initial configuration, p to an empty path, and d to an empty region diagram.

2. If the alloted time is exhausted or no input motions remain, return p.

3. Retrieve from d the region r containing c, constructing it if necessary. Apply all external motions to c in r. Update p with the resulting path segment and set c to its last configuration.

4. Retrieve from d the region r containing c, constructing it if necessary. Apply the next input motion to c in r. If no motion occurs, discard the first input motion. Otherwise, update p with the resulting path segment and set c to its last configuration.

5. Go to step 2.

Figure 8: Algorithm for kinematic simulation with simple dynamics.

coordinates by repeatedly testing the constraints for dependent parameters then propagating the results through the linkages. It sets the other coordinates to their initial values, since the motion leaves them constant.

In segment 1 of the feeder animation (Figures 3 and 4), 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, so the search for dependent coordinates ends. 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 5, 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.

After determining the dependent coordinates and fixing the other coordinates, the program finds the endpoint of the motion within the region by augmenting the region constraints with the constraint $x = x_0 + vt$ and calculating the maximum of t. It traces the CS path by solving the constraints for the coordinates as functions of t then substituting the values $0, s, 2s, \ldots, t_{max}$ for t. The motion in segment 1 of the feeder animation ends after 2.1268 seconds when the piston breaks contact with block 1. The values of c_d and x_p along the path segment are

c_d	0	0.25	0.5	0.75	1	1.25	1.5	1.75	2	2.1268
x_p	10	9.8722	9.5023	8.9292	8.2121	7.4223	6.6315	5.9008	5.2721	5

and the other fixed-axes coordinates are constant.

The program derives the symbolic description of the path segment from the input motion and the coordinate dependencies. The description specifies the driving motion, which parts

move because of contacts, and the motion types of the parts. The description of a part specifies it name, its motion type, and the initial and final values of its mobile coordinates. The mobile coordinates of a part determine its motion type: *stationary* if it does not move, *translates* if it translates, *rotates* if it rotates, *helical* if it translates and rotates in a coupled manner, and *rotates-and-translates* if it translates and rotates independently. Segment 1 in Figure 4 shows the description of the first motion segment of the feeder. The first line describes the input motion of the driver, the second line describes how the driver moves the piston, and the following lines describe the motions of the fixed-axes parts.

4.2.1 Region diagram construction

The kinematic simulation module incrementally generates the region diagram of the mechanism based on requests for the regions that contain certain configurations. It starts with an empty region diagram. Given a request, it retrieves the regions in the current diagram and returns those that contain (actually, whose closures contain) the input configuration. A configuration lies in a region if it satisfies the constraints that define the region. If no current region contains the input 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.

In the feeder, the fixed-axes parts are the frame, driver, piston, and blocks with mobile coordinates c_d, x_p, x_{bi}, and y_{bi} $(i = 1, 2, 3)$. Figure 5 shows the piston/block region diagram. The block/block diagram has the same structure, but with different size regions. Each block/frame diagram consists of two regions: a region $y_{bi} = -1$ where block i lies on the table and a region $y_{bi} > -1$ where block i is above the table. The piston/frame and driver/frame diagrams each consists of a single region that encompasses the entire CS, hence imposes no constraints. The other pairs impose no constraints because they do not interact. The initial configuration is $c_d = 0$, $x_p = 10$, $x_{b1} = 12$, $y_{b1} = 1$, $x_{b2} = 12$, $y_{b2} = 3$, $x_{b3} = 12$, and $y_{b3} = 5$.

This configuration yields one containing region per pairwise diagram:

pair	region	constraints
frame/block 1	r_1	$x_{b1} = 12;\ \ y_{b1} > -1$
frame/block 2	r_1	$x_{b2} = 12;\ \ y_{b2} > -1$
frame/block 3	r_1	$x_{b3} = 12;\ \ y_{b3} > -1$
piston/block 1	r_3	$-7 \leq x_p - x_{b1} \leq 4;\ \ y_{b1} \geq 1$
piston/block 2	r_3	$-7 \leq x_p - x_{b2} \leq 4;\ \ y_{b2} \geq 1$
piston/block 3	r_3	$-7 \leq x_p - x_{b3} \leq 4;\ \ y_{b3} \geq 1$
block 1/block 2	r_3	$-4 \leq x_{b2} - x_{b1} \leq 4;\ \ y_{b2} - y_{b1} \geq 2$
block 1/block 3	r_3	$-4 \leq x_{b3} - x_{b1} \leq 4;\ \ y_{b3} - y_{b1} \geq 2$
block 2/block 3	r_3	$-4 \leq x_{b3} - x_{b2} \leq 4;\ \ y_{b3} - y_{b2} \geq 2.$

Intersecting these regions yields the fixed-axes subassembly region defined by the constraints:

$$\begin{array}{l} x_{b1} = 12;\ \ x_{b2} = 12;\ \ x_{b3} = 12 \\ y_{b1} \geq 1;\ \ y_{b2} \geq y_{b1} + 2;\ \ y_{b3} \geq y_{b2} + 2 \\ 5 \leq x_p \leq 16. \end{array} \tag{1}$$

The first three constraints show that the magazine prevents the blocks from moving left or right, the next three constraints show that the piston supports block 1 which supports block 2 which supports block 3, and the final constraint shows the range over which the piston supports block 1.

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 on y by scanning the table that encodes the region diagram and recording the minimum and maximum of y for x between x_l and x_u. It calculates the x constraints by scanning the table for the widest interval that encloses the initial value of x and for which $f(x)$ lies between y_l and y_u. Both calculations requires linear time in the length of the table.

The feeder has a single linkage with input c_d, output x_p, and input/output function shown in Figure 7. The fixed-axes constraints of the initial region (Eq. (1)) restrict c_d to $[-\infty, \infty]$ and x_p to $[5, 16]$. The linkage restricts x_p to $f([5, 16]) = [4, 10]$ and c_d to $f^{-1}([5, 10]) = [0, 2.1268]$. The program composes these constraints with the fixed-axes constraints, yielding $5 \leq x_p \leq 10$ and $0 \leq c_d \leq 2.1268$. It constructs the initial region of the mechanism by augmenting the fixed-axes constraints with these constraints.

As the final step in region construction, the program asserts the frictional constraints for the touching pairs of sticky surfaces. It determines which surfaces touch from the pairwise

region diagrams, obtains their axes of motion from the region, and adds the appropriate constraint to the region constraints. We illustrate friction in Section 5.

The program uses a subset of the BOUNDER inequality prover [14] 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.

4.3 Animation

The animation module animates CS paths by converting them into the format that a standard graphics package displays. It makes minimal assumptions about the underlying graphics package, just that it can display triangles and quadrangles in three dimensions and that it can position objects with transformation matrices. We chose the MINNEVIEW graphics package from the University of Minnesota Geometry Center, which runs on the IRIS workstation. We could adapt the program to another graphics package by changing the format of the part descriptions and transformation matrices.

The program first approximates the part shapes with quadrangles. It does this once per mechanism, not once per animation, so the running time is insignificant (seconds to minutes). It converts a part slice by slice. Figure 9 illustrates the process. The program piecewise linearizes the arc segments in the slice boundary, since few graphics packages handle arcs. It quadrangulates the area enclosed in the piecewise linear boundary, using a vertical line sweep algorithm. It represents the top and bottom of the slice by embedding the quadrangles in the corresponding planes. It represents the sides of the slice by forming quadrangles from corresponding pairs of top and bottom boundary segments. The part description consists of a list of triangles and quadrangles along with optional color and lighting information.

Given a quadrangulated mechanisms and a CS path, the program animates the mechanism by generating snapshots of the configurations in the CS path then displaying them consecutively with the MINNEVIEW program. The snapshots encode the configurations of the parts of the mechanism as homogeneous transformation matrices. MINNEVIEW takes the part descriptions and the snapshots as inputs and generates the animation. Typical animations contain about 100 snapshots and run in real time.

4.4 The feeder revisited

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

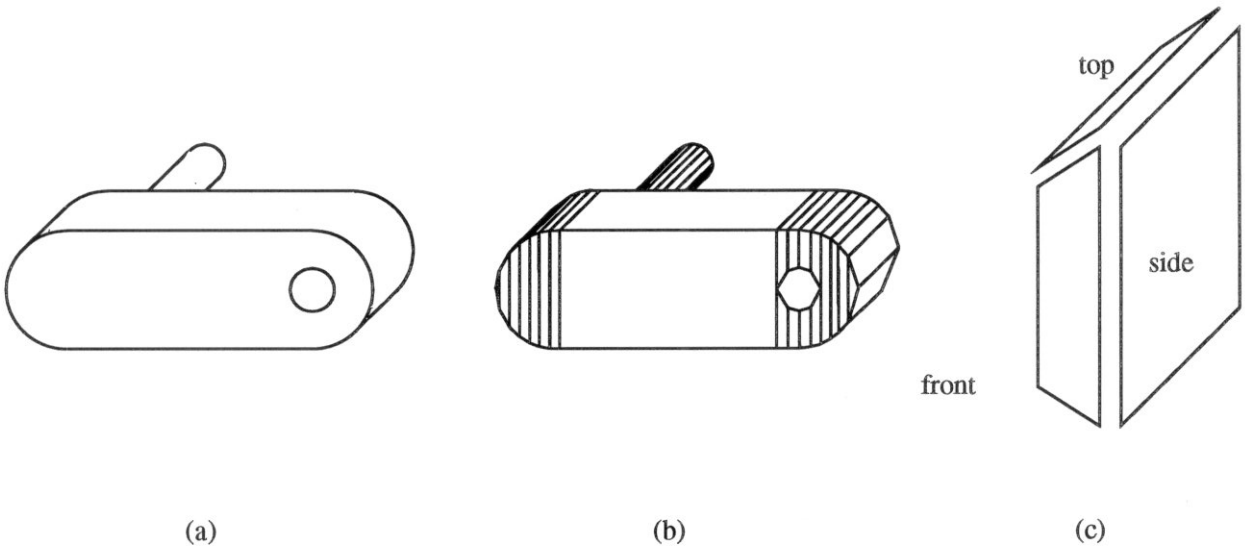


Figure 9: Slicing the driver into quadrangles.

containing 90 configurations and a region diagram containing 16 regions. It runs in 10 minutes on a DEC workstation, while MINNEVIEW 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 [10] 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.

Figure 10 shows the sequence of regions that the feeder traverses. The regions are specified by their components in the pairwise region diagrams. The driver rotates counterclockwise at one radian per second, which makes the linkage slide the piston left and right. Each time the piston slides out from under the bottom block, a region transition occurs. Gravity drops the stacked blocks onto the table, producing another region transition. The piston slides left then right, producing a region transition when it touches the bottom block. It continues right, pushing the bottom block and supporting the rest of the stack. Region transitions occur when the bottom block moves out from under the stack, when it hits the left side of the block that dropped on the previous cycle, and when the piston starts moving left to begin the next cycle. The piston pushes the three blocks in the stack onto the table during its first three cycles. A higher stack would take more cycles and generate more regions, but would not affect the simulation significantly.

motion	f/b_1	f/b_2	f/b_3	p/b_1	p/b_2	p/b_3	b_1/b_2	b_1/b_3	b_2/b_3
p slides left	r_1	r_1	r_1	r_3	r_3	r_3	r_3	r_3	r_3
blocks drop	r_1	r_1	r_1	r_2	r_2	r_2	r_3	r_3	r_3
p slides left	r_2	r_1	r_1	r_1	r_2	r_2	r_3	r_3	r_3
p slides right	r_2	r_1	r_1	r_1	r_2	r_2	r_3	r_3	r_3
p pushes b_1	r_2	r_1	r_1	r_1	r_3	r_3	r_3	r_3	r_3
b_1 clears b_2	r_2	r_1	r_1	r_1	r_3	r_3	r_2	r_2	r_3

Figure 10: The regions that the feeder traverses during the first cycle of the driver. The labels b_i, p, and f stand for block i, the piston, and the frame.

4.5 Program extensions

The current program covers roughly half of all mechanisms and analyzes a mechanism in minutes. A few straightforward extensions would significantly extend the coverage and reduce the running time.

The program handles *nondegenerate* linkages in which the linkage equations are nonsingular. This excludes mechanisms with redundant linkages or with degenerate linkages. The program can handle mechanisms with redundant linkages if the user identifies and omits them. Degenerate linkages pose a harder problem because the continuation algorithm cannot handle them. We are testing a simplicial algorithm, which handles degeneracy, but have yet to integrate it into the program.

We can speed up the program by changing the implementation language and by improving the algorithms. Simply switching from LISP to C would reduce the running time by a factor of 1000, based on empirical measurements. Given the work involved, it seems more practical to rewrite only the most costly modules. The prime candidate is the constraint manager, which performs roughly 90% of the computation. We plan to replace the BOUNDER implementation with a special-purpose linear constraint reasoner [8] that runs in milliseconds. The change will speed up the entire program by a factor of ten. The special-purpose program eliminates redundant constraints from constraint sets, whereas BOUNDER does not. For example, BOUNDER specifies the initial region of the feeder with 20 constraints even though 8 suffice. Part approximation is also worth speeding up, since it takes the bulk of the remaining run time. A better algorithm written in C will reduce it from minutes to milliseconds.

5 Examples

We have tested our program on a dozen examples, including the feeder, a transmission [10], 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 it region diagram and contains a spring that opposes the input motion. The shoe brake has simultaneous input motions, springs, and friction.

Table 2 summarizes the analyses of the 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. We now discuss the rim lock and the shoe brake.

	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; 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	4585
snapshots	88	32	87	6
runtime (min.)	10	0.3	2	0.2

Table 2: Summary of the analyses of four mechanisms.

Figure 11 shows the unlocking sequence of a rim lock for a door. The lock consists of a frame, a key, a rim, a latch, and a spring (not shown) that pushes the latch agains the rim. In the initial, locked 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.

Figure 12 shows the braking sequence of a drum with 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 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.

6 Beyond simple dynamics

In this section, we discuss ways to extend the coverage of kinematic simulation with simple dynamics. In our previous paper, we describe methods for extending the kinematic coverage from 59% to about 90% while maintaining reasonable computational efficiency. These extensions would raise the overall coverage from 48% to about 72%, since simple dynamics covers 80% of the mechanisms. We now discuss extensions to simple dynamics. We consider improved modeling, steady-state dynamics, and dynamical simulation.

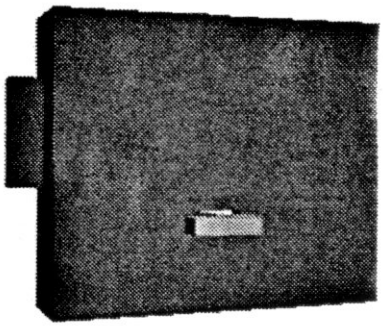
The survey in Table 1 shows that simple dynamics covers 2096 out of 2648 mechanisms. We can extend the coverage by more innovative modeling. The biggest payoff comes from replacing 2D springs by 1D springs where possible [1] This covers an additional 117 mechanisms. Of the remaining springs, 76 require a 2D model and 10 are used in shock absorbers or vibrators. This approach invests increased modeling effort for ease of analysis.

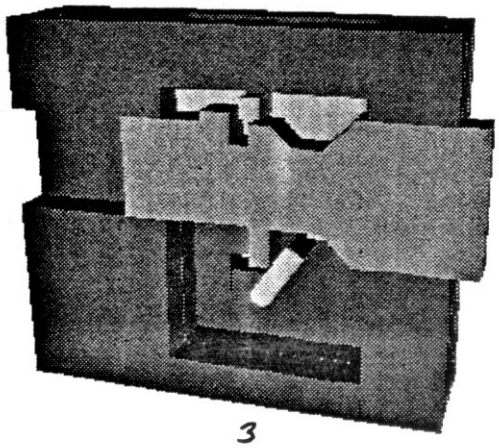
Fig 13a shows a switching lever with a true 2D spring that cannot be modeled with 1D springs. Lever 1 rotates about fixed axis A. Spring 2 holds lever 1 against one of the stops a. The lever has two stable equilibrium positions at the two extremes and an unstable equilibrium position in the middle.

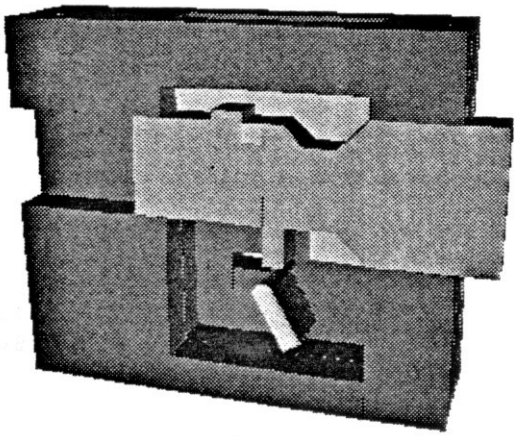
Steady-state analysis covers an additional 269 of the remaining 435 mechanisms. Steady-state analysis abstracts away transient acceleration and vibration, but derives the precise steady-state effect of forces, masses, moments of inertia, and friction. The coverage includes friction mechanisms (86), such as rollers, friction drives and friction clutches; mechanisms with competing forces or inertia (91), such as governors and tripping mechanisms, brakes (26), wedges and clamps (36); and measuring instruments (30), such as balances and dynamometers.

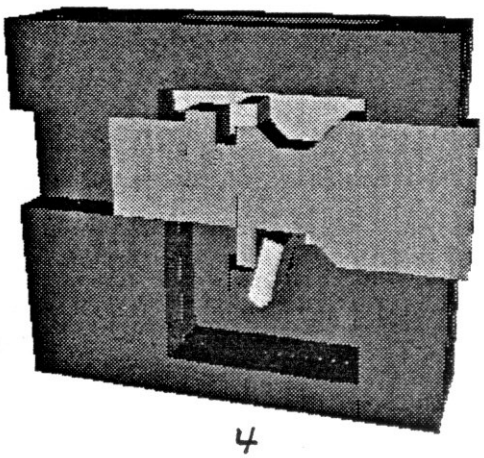
Fig 13b shows a mechanism covered by steady-state analysis: a self-disengaging friction clutch. Cross-shaped disk 6 is mounted on input shaft 1. Four spring-loaded pads 7 are symmetrically mounted on the disk. The springs pull the pads inward when the driving shaft does not rotate. As the driving shaft accelerates, the centrifugal forces of inertia on the pads increase, pushing the pads toward driven disk 8. The pads are pressed against the

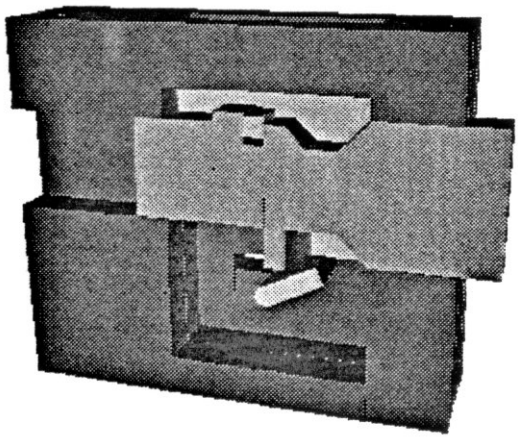
[1] Springs play an important role in mechanisms: of the 2648 mechanisms, 565 or 21% contain one or more springs.











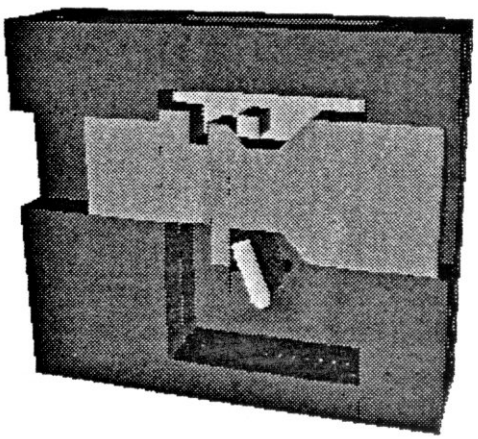


Figure 11: Animation of a rim lock. The top left snapshot shows a back view of a locked configuration. The following snapshots show a front view of an unlocking sequence.

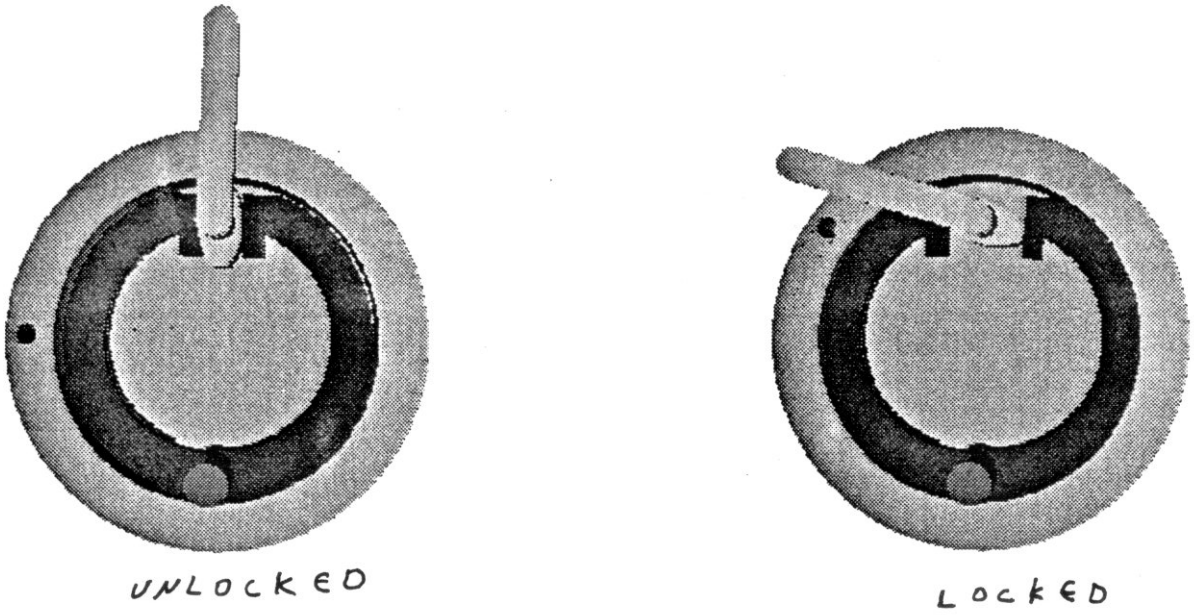


Figure 12: Animation of a shoe brake. The brake rotates in the first configuration and is locked in the second.

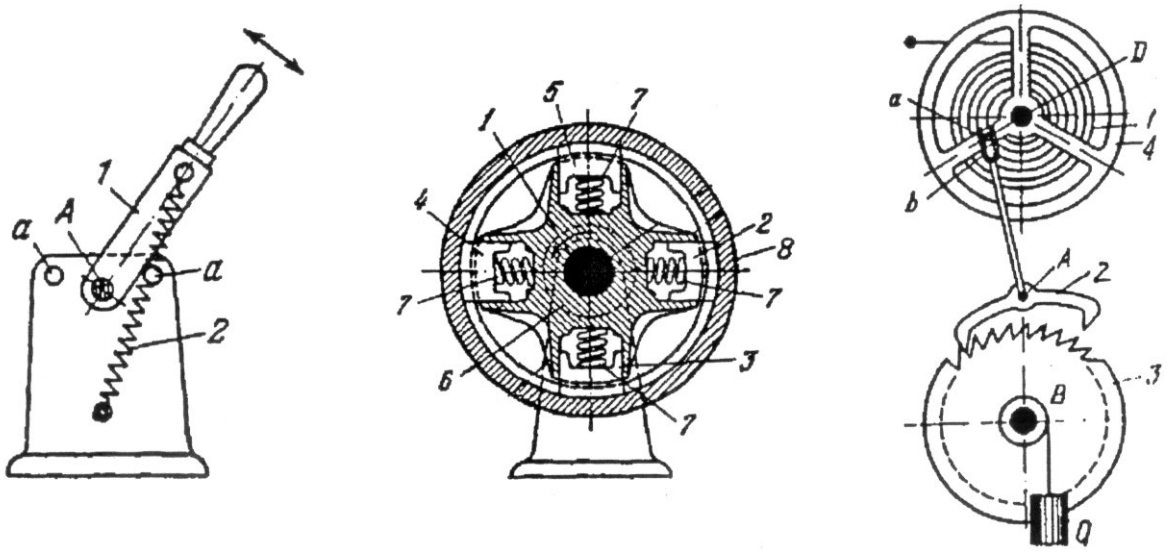


Figure 13: Sample mechanisms not covered by simple dynamics: (a) a switching lever (343), (b) a friction clutch (3405); (c) a clock escapement (2749).

driven disk, thereby transmitting the rotation and increasing the torque. If the speed of the shaft falls below a threshold, the springs disengage the pads and the rotation is no longer transmitted.

Further dynamical analysis is necessary for the remaining 106 (4%) mechanisms. The only solution we can see is detailed dynamical simulation. This is necessary for simulating feeding and sorting mechanisms (34), escapements (19), vibrators (20), springs used as vibrating links (10), measuring devices such as accelerometers, tachometers, and dynamographs (12), and others (11). The hardest devices to simulate are feeding and sorting mechanisms.

Fig 13c shows a mechanism that requires further dynamical analysis: an escapement-type governor controlled by a balance. Weight Q applies a clockwise torque to escapement wheel 3 about fixed axis B. Balance 4 oscillates about fixed axis D by the spiral spring 1 and has a pin a sliding in fork b of anchor 2. This contact oscillates anchor 2 about fixed axis A and its pallets allow intermittent rotation of escape wheel 3. In this mechanism, the precise transient motion achieves the desired function.

7 Related work

Spatial reasoning about moving objects is an active research topic that spans mechanical engineering, model-based reasoning, robotics, and graphics. Each field has its own goals and emphasis. Mechanical engineering and model-based reasoning aim to simulate the behavior of mechanisms accurately and efficiently. Robotics aims to plan the motion of robots in cluttered environments. Graphics aims to produce realistic animations of interesting motions by complex shapes. Our goals are closest to those of mechanical engineering and model-based reasoning, although we share the other goals to some extent.

Traditional mechanical engineering simulators have been around the longest and are commercially available. Some are tailored specifically to linkages, but others (such as ADAMS) also handle fixed topology mechanisms with user-specified pairwise kinematics. The packages automatically derive either the Newton-Euler or the Lagrange equations of motion, a mixture of algebraic and differential equations, then numerically integrate them for a given initial condition. They presuppose that all contacts are permanent and hence that the equations are fixed and independent of the geometry of the mechanism. They uniformly consider six degrees of freedom per part, regardless of the real degrees of freedom. Haugh [7] surveys the commercially available packages of this type and describes test runs on large systems. In one chapter, Chace [2] reports the simulation of a vehicle base with 42 degrees of freedom and 580 motion equations.

Traditional simulators have several limitations. The user must provide the initial model. The simulators cannot handle more general mechanisms in which the geometry of parts affects behavior and the part contacts vary. Nor can they interpret the simulation results, requiring the user to recognize periodic behavior and other stopping conditions.

Model-based simulators address these limitations by incorporating kinematic analysis into dynamical simulators. Cremer [3] describes a simulator that uses a model-based three-dimensional part representation and that handles contact changes, friction, and collisions. The program only models a few shapes, such as spheres and polyhedra. It sets up the Newton-Euler equations for a set of objects in an initial configuration, simulates until two objects collide, modifies the equations to reflect the new configuration, and continues simulating. The output of the program is an animation of the objects and graphs of the time evolution of their configurations, velocities, and accelerations. Gelsey [5] describes a similar simulator that also outputs behavioral summaries. Kramer [11] describes a linkage simulation program that symbolically derives the kinematic equations and calculates the linkage configurations for particular values of the input motion parameters.

Our program improves upon previous model-based simulators by being faster, more robust, and cover more mechanisms. We produce a region diagram of the mechanism kinematics, which supports symbolic reasoning about its behavior, in addition to a simulation. Our program is more efficient than previous ones, both because the region diagram eliminates the need for collision detection during simulation and because simple dynamics eliminates the need for differential equations. Region diagrams also make our program more robust than previous ones. Previous programs run the risk of jumping over narrow boundaries, especially those with variable step-size integrators. For example, they can deduce that a rolling ball crosses a narrow chasm. Our program will not miss the chasm because it is a different region.

Robotics research in motion planning addresses spatial reasoning problems akin to those in mechanism analysis [12]. The basic task is to plan the motion of a robot between two configurations in a cluttered environment. The field also studies extensions to the basic task, including multiple robots, moving obstacles, constraints on legal paths, and path optimization. Both motion planning and mechanism analysis rely heavily upon configuration space, computational geometry, and differential topology. They differ in their task domains, hence in the ways they use these tools. Motion planning searches the CS for a single path, while simulation traverses the CS in the direction dictated by the input motion. Motion planning occurs in a complex, low dimensional CS with little structure, whereas simulation occurs in a low dimensional CS embedded in a high dimensional space of potential motions. Our simulation strategy is inappropriate for motion planning because its underlying assumptions apply to typical mechanisms, not to robots.

Computer graphics produces realistic looking (not necessarily true to physics) animations of physical phenomena. For example, Hahn [6] describes a program that animates complex rigid object motions, such as a chair falling down a staircase. The program formulates and simulates the full dynamical equations. Although some of the graphics techniques are relevant to our work, our tasks are quite different. We derive an explicit representation of the behavior of a mechanism, whereas they visualize unstructured collections of objects. We use animation as a tool for describing CS pathes and region diagrams, whereas they study it in its own right.

8 Conclusions

In this paper, we present a practical simulation algorithm 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. We believe that the symbolic output can serve as the basis for automating other tasks such as design and diagnosis.

We develop simple, efficient algorithms that handle the salient behavioral aspects of most mechanisms, rather than attempting to handle all aspects of all mechanisms. We formulate a class of feasible mechanisms whose kinematic analysis is tractable. The class contains linkages, fixed-axes assemblies, and combinations of the two. We develop a simple model of dynamics that captures the steady-state effect of forces in most mechanisms without the conceptual and computational cost of dynamical simulation. We assess the coverage of feasible mechanisms and of simple dynamics with a survey of 2500 mechanisms from an encyclopedia of mechanisms. We find that 59% of the mechanisms are feasible, 79% are covered by simple dynamics, and 48% are both feasible and covered by simple dynamics. Hence, kinematic simulation with simple dynamics covers roughly half of all mechanisms.

The guiding principle in our research is to exploit the structure and function of mechanisms. Mechanisms are designed to perform specific tasks. These tasks impose structural constraints on the shapes, configurations, and interactions of the parts of the mechanism. Every step of our algorithm relies upon this structure. Modeling and kinematics simplify analysis because mechanisms have few degrees of freedom and because parts interact in simple ways. Simple dynamics is informative because mechanisms are designed for regular behavior. Our strategy is ineffective for unstructured systems, such as rockslides. Modeling and kinematics cannot reduce the complexity of a rockslide because the rocks move in every direction, have irregular shapes, and collide often. Simple dynamics tells us nothing because the rocks slide erratically.

Our analysis algorithm is limited by the types of mechanisms it can analyze, by the dynamical phenomena it can model, and by the quality of explanations it produces. In Section 6, we discussed possible extensions that would significantly extend the kinematic and dynamical coverage of the analysis. We now discuss full dynamical analysis and explanation.

Full dynamical analysis is necessary for correctly simulating mechanisms not covered by simple dynamics and for accurately simulating covered mechanisms. Kinematic simulations with simple dynamics sets the stage for full dynamical analysis. Modeling identifies the relevant CS coordinates and possible part interactions. Subassembly analysis and simulation compute part interactions and coordinate dependencies. This information simplifies and speeds up full dynamical simulation.

We can formulate the full dynamical equations in CS coordinates instead of in part coordinates, which typically reduces the number of equations from six per part to one per part. Eliminating the redundant coordinates makes the remaining equations less stiff. 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.

More sophisticated interpretation of the simulation output is necessary to hide irrelevant details, such as gear chatter, and to identify behavior patterns, such as cycles and repetitions. We can produce more focused descriptions by using the CS simplification and abstraction operators developed by Joskowicz [9]. Simplification operators suppress irrelevant information by adding constraints and assumptions. The constraints restrict the type and range of input motions based on the operating context of the mechanism and on dynamical considerations. Abstraction operators suppress details by defining multiple levels of resolution. Simplification and abstraction are essential for many common tasks, such as comparing mechanisms or classifying mechanisms by behavior. We can infer behavior patterns in kinematic simulations with simple pattern matching techniques.

References

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

[2] Chace, M. Methods and experience in computer aided design of large-displacement mechanical systems. In Haugh [7].

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

[4] Doedel, E. AUTO 86 user manual: software for continuation and bifurcation problems in ordinary differential equations. Technical report, Princeton University, Feb. 1986.

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

[6] Hahn, J. K. Realistic animation of rigid bodies. *Computer Graphics* **22** (1988) 299–308.

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

[8] Huynh, T., Joskowicz, L., Lassez, C., et al. Reasoning about linear constraints using parametric queries. in: *Proc. 10th International Conference on Foundations of Software Technologies and Theoretical Computer Science*, Bangalore, India, 1990.

[9] Joskowicz, L. Simplification and abstraction of kinematic behaviors. in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989. Reprinted in [15].

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

[11] Kramer, G. A. Solving geometric constraint systems. in: *Proceedings of the National Conference on Artificial Intelligence.* American Association for Artificial Intelligence, 1990.

[12] Latombe, J.-C. *Robot Motion Planning.* (Kluwer Academic Publishers, 1991).

[13] Nikravesh, P. E. *Computer-Aided Analysis of Mechanical Systems.* (Prentice Hall, New Jersey, 1988).

[14] Sacks, E. P. Hierarchical reasoning about inequalities. in: *Proceedings of the National Conference on Artificial Intelligence*, 1987. Reprinted in [15].

[15] Weld, D. S. and de Kleer, J. (Eds.). *Readings in Qualitative Reasoning about Physical Systems.* (Morgan Kaufman, San Mateo, Ca., 1990).