ASME Design Theory and Methodology 1995, pp. 603 – 611.

TURNING SKETCHES INTO WORKING GEOMETRY

Thomas F. Stahovich, Randall Davis, Howard Shrobe[1]
Artificial Intelligence Laboratory
Massachussets Institute of Technology
Cambridge, Massachusetts

ABSTRACT
We describe a program called SketchIT that uses a description of desired behavior to transform a sketch of a mechanical device into a "BEP-Model," a parametric model with constraints that ensure the device produces the desired behavior. The program also generates alternative implementations for the design, each of which is represented as a BEP-Model. The program is based on qualitative configuration space, a novel representation for mechanical behavior.

INTRODUCTION
A common task in mechanical design is the search for component geometries that will provide a desired behavior. Parametric modelers are often useful in this search because they provide a convenient means for modifying geometry. However, the designer is still responsible for creating the parametric model in the first place, then for deciding what changes to make.

We are developing SketchIT, an intelligent design assistant that assists the designer in finding a geometry that provides a specified behavior.[2] Our program uses a description of desired behavior to transform a sketch of a mechanical device into a "BEP-Model," a parametric model with constraints that ensure the geometry produces the desired behavior. The program also generates alternative implementations for the design, each of which is represented as a BEP-Model. We use DesignView[3] to evaluate the BEP-Models produced by our program.

The constraints in each BEP-Model represent the regions in parameter space (i.e., the space spanned by the geometric parameters) that provide the desired behavior. Thus they define a family of design solutions which the designer can explore in order to satisfy other design requirements, such as requirements on size or cost.

We have tested the program on two design problems: a mechanical circuit breaker and the firing mechanism from a single action revolver. We believe the program is applicable to a wide class of devices described in the future work section.

Our techniques can be viewed as a natural complement to both bond graph based design techniques and the well known design techniques for fixed topology mechanisms.

In this paper we use the design of a circuit breaker to illustrate the program's operation. One specific implementation of the device is shown in Figure 1a. In normal use, current flows from the lever to the hook; current overload causes the bimetallic hook to heat and bend, releasing the lever and interrupting the current flow. After the hook cools, pressing and releasing the pushrod resets the device.

The designer describes the circuit breaker to SketchIT with the stylized sketch shown in Figure 1b. Because SketchIT is concerned only with the *functional geometry*, i.e., the faces where parts meet and through which force and motion are transmitted (lines f1 through f8), the designer's task is to indicate which pairs of faces are intended to engage each other. Consideration of the connective geometry (the surfaces that connect the functional geometry to make complete solids) is put off until later in the design process.

The designer describes the desired behavior of a device to SketchIT using a state transition diagram (figure 2b). Each node in the diagram is a list of the pairs of faces that

[1]Support for this project was provided by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

[2]See Stahovich (1995) for a more complete description of this work.

[3]DesignView is a commercial parametric modeler based on variational geometry.

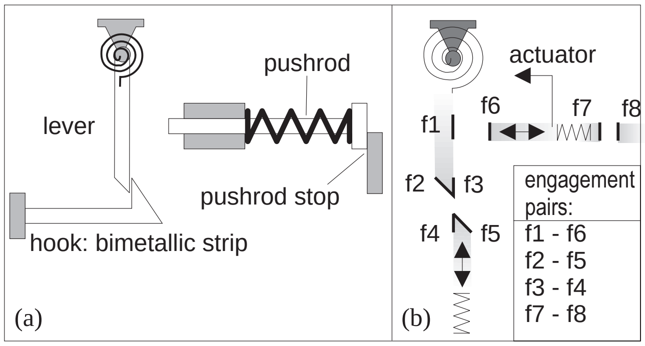


Figure 1: (a) One structure for the circuit breaker. (b) Sketch of circuit breaker as actually input to program. Engagement faces are in bold. The actuator applied to the pushrod represents the motion imparted by the user. For our convince we use labels to refer to engagement pairs: (f1 f6)=push-pair, (f2 f5)=cam-follower, (f3 f4)=lever-stop, (f7 f8)=pushrod-stop.

are engaged and the springs that are relaxed. (The pairs of faces not listed at a node are by default disengaged, the springs not listed are by default not relaxed.) The arcs are the external inputs that drive the device. Figure 2b, for instance, describes how the circuit breaker should behave in the face of heating and cooling the hook and pressing the reset pushrod.

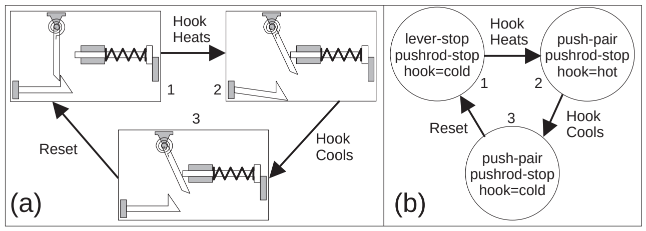
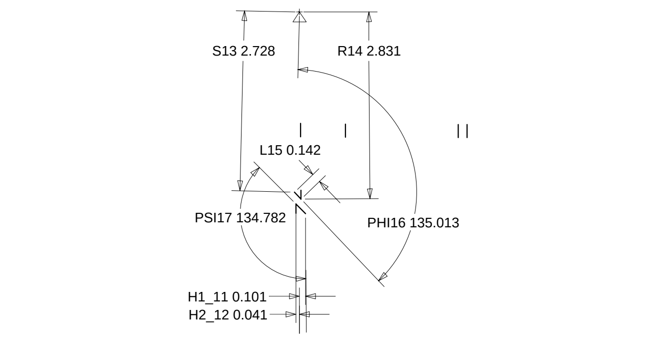

Hook Heats
1
2
3
Reset
Hook Cools
(a)
lever-stop pushrod-stop hook=cold
Hook Heats
push-pair pushrod-stop hook=hot
1
2
3
Reset
Hook Cools
push-pair pushrod-stop hook=cold
(b)

Figure 2: The desired behavior of the circuit breaker. (a) Physical interpretation. (b) State transition diagram. In each of the three states, the hook is either at its hot or cold neutral position.

Figure 3 shows a portion of one of the BEP-models SketchIT derives from the sketch of the circuit breaker (Figure 1b) and the desired behavior (Figure 2b). The top of the figure shows the parameters that define the sloped faces on the lever and the hook (faces f2 and f5 respectively). The bottom shows the constraints that ensure this pair of faces plays its role in achieving the overall desired behavior: moving the lever clockwise pushes the hook down until the lever moves past the point of the hook, whereupon the hook springs back to its rest position. As one example, the ninth equation, `0 > R14/TAN(PSI17) + H2_12/SIN(PSI17)`, constrains the geometric parameters so that the contact point on face f2 never moves tangent to face f5. This in turn ensures that when the two faces are engaged, clockwise rotation of the lever always increases the deflection of the hook.

The particular parameter values in the top of Figure 3 were computed by DesignView as solutions to the constraints of



```
H1_11 > 0      H2_12 > 0      S13 > H1_11
L15 > 0        PHI16 > 90     PHI16 < 180
PSI17 > 90     PSI17 < 180
0 > R14/TAN(PSI17) + H2_12/SIN(PSI17)
R14 = SQRT(S13^2 + L15^2 - 2*S13*L15*COS(PHI16))
```

Figure 3: Output from the program. Top: the parametric model; the decimal number next to each parameter is the current value of that parameter. Bottom: the constraints on the parameters. For clarity, only the parameters and constraints for faces f2 and f5 are shown.

the BEP-Model, and thus, this particular geometry provides the desired behavior.[4] Using DesignView, we can easily explore the family of designs defined by this BEP-Model. For example, Figure 4 shows another solution to the BEP-Model computed with DesignView. Because these parameter values satisfy the constraints of the BEP-Model, even this rather unusual geometry provides the desired behavior. As this example illustrates, the family of designs defined by a BEP-Model includes a wide range of design solutions, many of which would not be obtained with conventional approaches.

SketchIT also produces two kinds of design variants. One kind of variant is obtained by using a different implementation for a pair of engagement faces. Figure 5 shows an example: In the original implementation of the cam-follower engagement pair, the motion of face f2 is roughly perpendicular to the motion of face f5; in the new design of Figure 5, the motions are parallel. Conversely, the motions of faces f1 and f6, originally parallel, are now perpendicular.

SketchIT produces a second type of variant by changing rotating parts to translating ones, and vice versa. Figure 6 shows an example in which SketchIT replaces the rotating lever in Figure 1 with a translating part.

For each set of choices for the motion types (rotation or translation) of the components, the program may have several ways to implement each engagement pair; each variant of the second type has its own variants of the first type.

For the circuit breaker example the program currently generates 5 qualitatively distinct designs (shown in this paper); with very minor additions the program will produce a total of 13 distinct designs.

[4]We currently transfer the parametric geometry and constraints of the BEP-Model to DesignView manually; this can easily be accomplished automatically using DesignView's macro language.

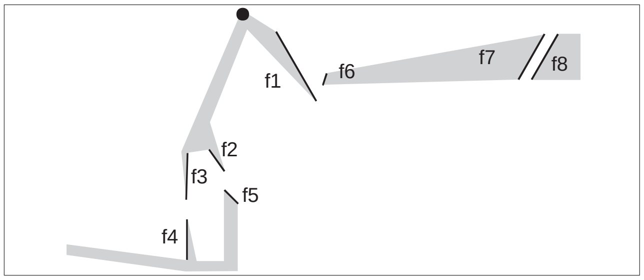


Figure 4: Another solution to the BEP-Model of Figure 3. Shading indicates how the faces might be connected to flesh out the components. This solution shows that the pair of faces at the end of the lever and the pair of faces at the end of the hook need not be contiguous.

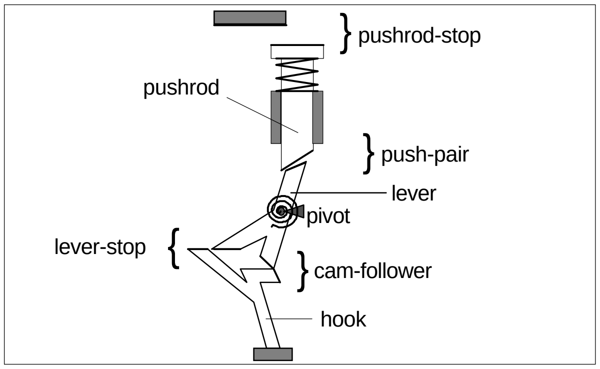


Figure 5: A design variant obtained by using different implementations for the engagement faces. The pushrod is pressed so that the hook is just on the verge of latching the lever.

THE REPRESENTATION

Qualitative c-space

Before SKETCHIT can compute the constraints on the geometry, it must determine what role each piece of geometry plays in achieving the desired overall behavior. This task is made difficult by the fact that sketches of conceptual designs are often incomplete and incorrect.

Sketches are incomplete in the sense that details are missing: a sketch may depict something about the kind of implementation the designer wants to use, rather than a specific, working implementation (e.g., the designer may want a cam and follower, but not necessarily with the exact dimensions and shapes shown in the sketch). A sketch can be incorrect in the sense that it contains a collection of fragments that implement pieces of the design, but those individual pieces do not produce the desired overall behavior.

Our first step in dealing with this problem was to use configuration space (c-space) because it represents the interaction between two parts rather than the shapes of the

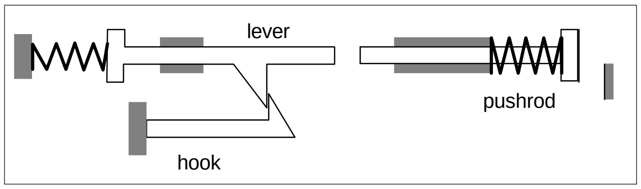


Figure 6: A design variant obtained by replacing the rotating lever with a translating part.

parts.[5] In so doing it allows us to move from representing shape to representing behavior.

Because the devices we are interested in are fixed axis,[6] the c-space for a pair of interacting parts is a plane, called a cs-plane, and the interaction of a pair of faces is a curve in the cs-plane (a cs-curve). In the same way that a solid has an inside and an outside, a cs-curve has a front and back. The back side is called blocked space, and represents configurations of the parts which would cause one part to penetrate the surface of the other.[7]

Each cs-curve represents a family of shapes (i.e., a family of interacting faces) that all produce the same behavior. However, we can identify a much larger family of shapes that produce the same behavior by abstracting the numerical cs-curves to obtain a *qualitative* c-space. In qualitative c-space (qc-space) we represent cs-curves by their qualitative slopes and the locations of the curves relative to one another. By qualitative slope we mean the obvious notion of labeling monotonic curves as diagonal, vertical, or horizontal; by relative location we mean relative location of the end points.[8]

This representation is useful because the qualitative slope of a cs-curve is an indicator of the behavior of the corresponding interaction. A cs-curve that is vertical or horizontal, for example, indicates what we call "stop behavior," in which the extent of motion of one part is limited by the position of another. A diagonal cs-curve indicates an interaction in which one part pushes the other.

The key, more general, insight here is that *if the cs-curves are monotonic, the first order dynamics of the device are completely determined by the qualitative slopes of the cs-curves and the locations of the cs-curves relative to one another.* By first order dynamics we mean the dynamic behavior obtained when the motion is assumed to be inertia-free and the collisions are assumed to be inelastic

[5]See, for example, Caine (1993).

[6]Fixed axis devices are composed of components that either translate along a fixed axis or rotate about a fixed axis (i.e., single degree of freedom components).

[7]For the most part, interactions between two finite surfaces produce cs-curves with finite extent. Each spring neutral position and actuator limit contributes an infinite boundary to c-space, as does each engagement pair in which one body is ground. Thus, a cs-plane contains both cs-curves and infinite boundaries. For shorthand convenience we refer to both of these together as cs-curves.

[8]We assume that all interactions have monotonic cs-curves.

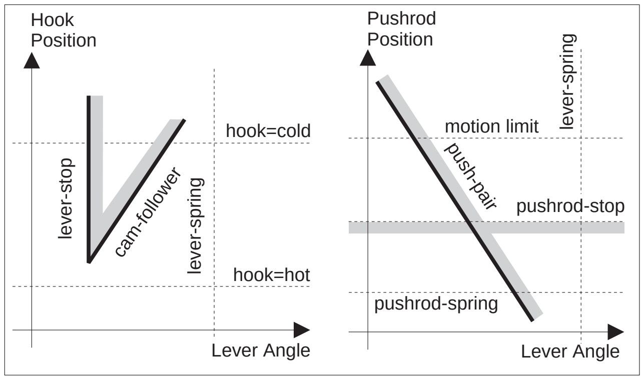


Figure 7: Qualitative c-space from Figure 1. For drawing convenience, qcs-curves are shown as straight line segments; they can have any shape as long as they are monotonic. Neutral positions of springs, limits of actuator movement, and interactions with the ground body (e.g., the pushrod-stop) are all shown as dashed infinite lines.

and frictionless.[9] The consequence of this general insight is that qc-space captures all of the relevant physics, and hence serves as a design space for behavior. It is a particularly convenient design space because it has only two properties: qualitative slopes and relative locations.

SKETCHIT uses generate and test to identify the role each pair of engagement faces plays in achieving the desired overall behavior. It computes the numerical c-space from the sketch, abstracts this into a qualitative c-space representation, and then uses qualitative simulation to test if that qc-space provides the specified behavior.

If qualitative simulation demonstrates that this qc-space does produce the correct behavior, we are done with this step. If not, SKETCHIT uses this initial guess as the starting point in a search for a working qc-space. SKETCHIT tries new choices for the relative locations of the qualitative cs-curves (qcs-curves) until it finds an arrangement that produces the desired behavior. If this fails, SKETCHIT will (in the future) also try other choices for the slopes of the qcs-curves (this is not implemented as of this writing, but is a straightforward extension in progress). If SKETCHIT still cannot find a qc-space that produces the correct behavior, the conceptual design is not capable of producing one.

Figure 7 shows the qualitative c-space abstracted from the numerical c-space of the sketch in Figure 1b.

Searching qc-space

The brute force approach—exhaustively enumerating all possible values for the location and slope of each qcs-curve—is clearly impractical in any non-trivial design task. As a result we have identified two sources of knowledge that allow us to prune the search space: we use knowledge about the mapping from the sketch to qc-space and knowledge about trajectories in qc-space.[10]

Mapping the sketch to qc-space. There are circumstances in which small variations in the geometry of the sketch produce qualitative changes in the behavior. When SKETCHIT detects these conditions, it asks the designer about his or her intent.

One such set of circumstances occurs when a numerical cs-curve is nearly vertical or nearly horizontal. If the cs-curve really is vertical or horizontal, the interaction produces stop behavior; but if the cs-curve really is sloped, the interaction produces pushing behavior (i.e., one part pushes the other). A small change in the slope of the cs-curve thus produces a large change in behavior. When SKETCHIT identifies a numerical cs-curve that is nearly vertical or nearly horizontal, it asks the designer if the interaction is intended to provide stop behavior.

Another set of circumstances occurs when the end points of two numerical cs-curves are coincident (or nearly so). This is likely to have been intentional, because it corresponds to a common and useful piece of behavior: if two cs-curves connect end to end, the device can slip from one engagement pair to the next with no disengagement in between. Hence if two numerical cs-curves (nearly) connect end to end SKETCHIT asks the designer if a smooth transition between engagement pairs was intended.

The “V” in the left of Figure 7 illustrates the use of these questions in the circuit breaker example. SKETCHIT notices that the cs-curve for the lever-stop engagement pair is nearly vertical and asks if we intended stop behavior; in this case we did. SKETCHIT then asks if the end points of the lever stop cs-curve and the cam-follower cs-curve are intended to be coincident; again they are, in order to allow the device to slip from one pair of engagement faces to the other without disengaging. Finally, SKETCHIT notices that the other ends of the two cs-curves have a coordinate in common and asks us if this was intended, but this is just coincidental. The “V” in the figure reflects the answers to these questions.

Knowledge about trajectories in qc-space. The desired behavior of a mechanical device can be described by a path through its c-space, hence the topology of the c-space can have a strong influence on whether the desired path (and hence the desired behavior) is easy, or even possible. This is equally true in the qc-space.

We are exploring the use of (but have not yet implemented) debugging rules that examine *why* a particular qc-space fails to produce the correct behavior, based on its topology. The qc-space may, for example, contain a funnel-like topology that “traps” the device, preventing it from traversing the desired path. If we can diagnose these kinds

[9] “Inertia-free” refers to the circumstance in which the inertia terms in the equations of motion are negligible compared to the other terms. One important property of inertia-free motion is that there are no oscillations. This set of physical assumptions is also called quasi-statics.

[10] As a consequence we have focused on creating this more informed generator and have not built an exhaustive generator.

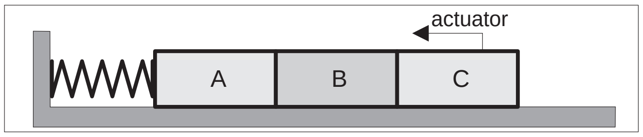


Figure 8: The three blocks slide along a frictionless horizontal surface. The spring pushes block A to the right, the actuator pushes block C to the left.

of failures to produce the desired behavior, we may be able to generate a new qc-space by judicious repair of the current one.

DYNAMIC SIMULATION

We use qualitative simulation to test the behavior of each qualitative c-space. SKETCHIT begins by identifying the forces on each body, and then uses a qualitative version of Newton's laws to compute the motion of each body. That motion continues until some event (e.g., a collision) changes the nature of the forces. At this point SKETCHIT stops, recomputes the forces, and then continues simulating. Because this is a qualitative simulation, ambiguities can arise concerning what will happen next; our simulator determines all possible sequences of events.

Computing Motion

With the Newtonian formulation of the equations of motion one begins by drawing a free-body diagram for each body, using force vectors to replace all of the engagements from other bodies. Newton's laws then produce a system of algebraic and differential equations relating the sum of the forces to the time rate of change of the momentum of the body. The solution to this system of equations contains values for the forces and the motion of the body.

Unfortunately, the Newtonian formulation does not work as easily with a qualitative representation. If we simply used qualitative versions of the engagement force vectors, there would be a significant amount of ambiguity in the force sums. Instead we represent each engagement by the type of constraint it applies to the body. Consider, for example, the three blocks in Figure 8. The spring pushes block A to the right, the actuator pushes block C to the left. In SKETCHIT's world, all actuators are assumed to be motion sources, that is, they assign position as a function of time. Block B, the block in the middle, experiences two engagement forces, one from A and one from C. Because the forces are in opposite directions, the qualitative sum of these forces is ambiguous.

However, we know that B will move to the left. Why is this? We know that the force C applies to B is whatever force is necessary for C to achieve its assigned motion. We call this kind of engagement a "motion constrained engagement" because it constrains the motion of the body to which it is applied. We can think of this kind of engagement as a kinematic constraint.

The spring's deflection determines the spring's force on A. In an inertia-free world, A will transmit the spring force to B. Thus, A applies a known force to B. In contrast to a motion constrained engagement which assigns motion, this kind of engagement assigns force. We call this kind of engagement a "compliant engagement" because it assigns force in the same way that a compliant member (e.g., a spring) does.

One of our basic principles is that *a motion constrained engagement overpowers a compliant engagement.* Hence, because there is a motion constrained engagement pushing B to the left and a compliant engagement pushing B to the right, B must move to the left.

Another basic principle is that, because we are restricting our attention to fixed axis bodies, the only component of a force that has any effect on the motion of a body is the projection of the force along the degree of freedom of the body. By reasoning about the projection of a force, rather than the whole force, SKETCHIT can substantially reduce ambiguity in force sums.

In an inertia free world, the motion of a body is in the direction of the net force. Hence, SKETCHIT determines the motion of a body by determining the net force on the body.

Events

Because the motion of a rigid body depends on the applied forces, the motion changes when there is a change in the nature of the forces. There are four kinds of events that can change the nature of the forces: an engagement is broken, an engagement is made, a spring passes through its neutral position, or an actuator reaches its limit. The first kind of event occurs when a trajectory in qc-space following a qcs-curve reaches the end of that curve; the other kinds of events occur when the trajectory hits a new curve.

To determine which kind of event will happen next, SKETCHIT examines the qc-space diagram. If the motion of a body in a qualitative cs-plane (qcs-plane) is along an engaged qcs-curve, or is horizontal or vertical, SKETCHIT can simply look along the trajectory to see what is reached first: a new qcs-curve or the end of the engaged qcs-curve.

If the motion is diagonal and does not follow a qcs-curve, we have to solve a geometric reasoning task in order to determine what event happens next. In our world the motion of every rigid body will remain either strictly positive, strictly negative, or zero over a time step, hence trajectories in qc-space will be monotonic. Thus, SKETCHIT's task is to find all of the qcs-curves that can be reached with a given monotonic trajectory. If there is a continuous chain of qcs-curves that block the trajectory, such as the chain composed of the lever-stop and the "hook=cold" qcs-curves in Figure 9, we can be certain that the trajectory will not reach any qcs-curve outside the chain (i.e., outside the shaded region). The qcs-curves composing the chain and any qcs-curves inside the chain (i.e., inside the shaded region) can be reached.

Because the qc-space is a multi-dimensional space, the trajectory through each qcs-plane is a two-dimensional projection of the trajectory through the qc-space. However we want to know the actual trajectory, i.e., the trajectory through the full multi-dimensional space. To do this SKETCHIT uses constraint propagation to reassemble the full trajectory from the projections and thus determine which of

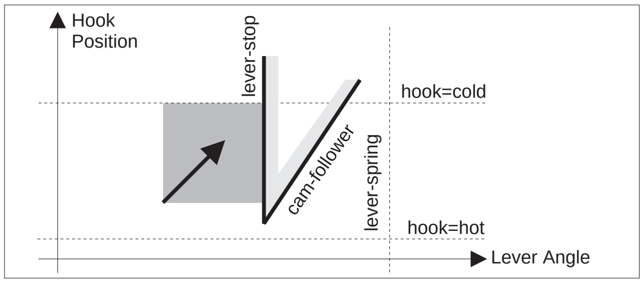


Figure 9: A diagonal trajectory in the qcs-plane.

the events predicted by the individual planes can happen first. SKETCHIT may find, for example, that the trajectory reaches an event in one qcs-plane before it reaches an event in any of the other qcs-planes.[11]

CONSTRUCTING BEP-MODELS

Once SKETCHIT has a qc-space that produces the desired behavior, it constructs a BEP-Model by mapping the qcs-curves to a library of implementations. Figure 10 shows all possible types of qcs-curves. Because there are eight types, and the coordinates q_1 and q_2 can be either rotation or translation, we need a library capable of implementing 32 different types of interactions.

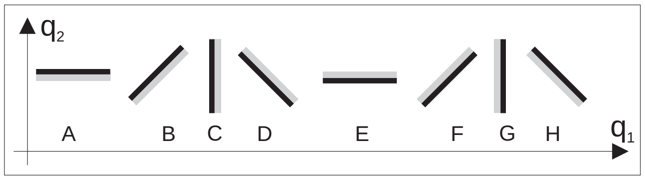


Figure 10: For drawing convenience, qcs-curves are shown as straight line segments; they can have any shape as long as they are monotonic.

Each library entry contains a parameterized drawing and a set of constraints that ensure that the geometry implements a monotonic cs-curve of a particular slope. We chose monotonic cs-curves because they ensure that the behavior will be as predicted by the simulator.[12] Because the qc-space places constraints on the relative locations of the qcs-curves, each library entry also contains algebraic expressions for the end points of the cs-curve that the geometry implements.

For example, Figure 11 shows a library entry for qcs-curve H in Figure 10, for the case in which q_1 is rotation and q_2 is translation (i.e., rotation in the negative direction causes translation in the positive direction). For the corresponding qcs-curve to be monotonic and have the correct slope, the following ten constraints must be satisfied:

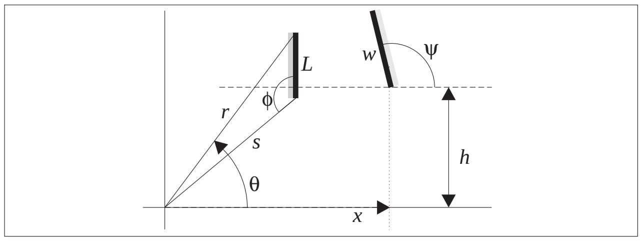


Figure 11: The two faces are shown as thick lines. The rotating face rotates about the origin; the translating face translates horizontally. θ is the angle of the rotor and x is the position of the slider

$w > 0$ $\quad L > 0$ $\quad h > 0$
$s < h$ $\quad r > h$ $\quad \pi/2 < \phi \leq \pi$
$\psi > 0$ $\quad \psi < \arcsin(h/r) + \pi/2$
$\arccos(h/r) + \arccos(\frac{L^2+r^2-s^2}{2Lr}) < \pi/2$
$r = (s^2 + L^2 - 2sL\cos(\phi))^{1/2}$

The end points of the corresponding cs-curve are defined as:

$\theta_1 = \arcsin(h/r)$ $\quad x_1 = r\cos(\theta_1)$
$\theta_2 = \pi - \arcsin(h/r)$ $\quad x_2 = r\cos(\theta_2)$

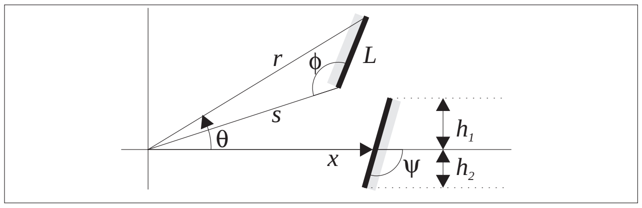


Figure 12: The two faces are shown as thick lines. The rotating face rotates about the origin; the translating face translates horizontally. θ is the angle of the rotor and x is the position of the slider

Figure 12 shows a second implementation for qcs-curve H, with constraints:

$h_1 > 0$ $\quad h_2 > 0$
$s > h_1$ $\quad L > 0$
$\pi/2 < \phi < \pi$ $\quad \pi/2 < \psi < \pi$
$0 > r/\tan(\psi) + h_2/\sin(\psi)$ $\quad r = (s^2 + L^2 - 2sL\cos(\phi))^{1/2}$

The end points of the corresponding cs-curve are given by:

$\theta_1 = -\arcsin(h_2/r)$
$x_1 = r\cos(\theta_1) - h_2/\tan(\psi)$
$\theta_2 = \arcsin(h_1/s) + \arccos(\frac{s^2+r^2-L^2}{2sr})$
$x_2 = s\cos(\arcsin(h_1/s)) + h_1/\tan(\psi)$

The slider in Figure 11 does not pass through the pivot but the one in Figure 12 does. In the first design the motion of the slider is approximately parallel to the motion of the rotor, while in the second the motion of the slider is approximately perpendicular to the motion of the rotor.[13] The two designs thus represent qualitatively different implementations for the same qcs-curve.

[11] There may be several possible next events. For example, in Figure 9, there are two possible next events: the lever could reach the hook, or the hook could reach its neutral position. When there is more that one next event, the simulator considers all possibilities.

[12] There may be other types of solutions, but we do not look for them.

[13] The first design is a cam with offset follower, the second is a cam with centered follower.

To generate the BEP-Model for the sketch, we select an implementation for each qcs-curve from the library. For each selection we create a new instance of the parameters and transform the coordinate systems to match those used by the actual components. Interactions with ground, such as the pushrod-stop in Figure 1b, are handled in a similar way. The locations of the qcs-curves in the qc-space are turned into constraints on the end points of the qcs-curves. Figure 3 (bottom of figure) shows the algebra SKETCHIT generates when the cam-follower of the circuit breaker is implemented with the geometry from figure 12.

Our library contains implementations that use flat faces.[14] We have at least one implementation for each of the 32 kinds of interactions shown in Figure 10. We have begun work on implementations using circular faces.

DESIGN VARIANTS

One of the ways that SKETCHIT generates design variants is by selecting different implementations for a qcs-curve. For example, SKETCHIT created two designs for the cam-follower interaction of the circuit breaker by using the two implementations shown in Figures 11 and 12. SKETCHIT creates two designs for the push-pair in the same way. With two implementations for each of two interactions, SKETCHIT was able to create four of its design variants for the circuit breaker (Figure 5 shows one of the four).

SKETCHIT also generates design variants by changing the motion type of a component from translation to rotation (or vice versa). If we assume that components rotate less the a full revolution, changing the motion type does not change the qualitative c-space, thus the behavior computed by the simulator is still valid.[15] When SKETCHIT selects geometric implementations from the library, it simply selects entries that match the particular choices of motion types. For example, by changing the lever of the circuit breaker from a rotor to a slider, SKETCHIT generates the design shown in Figure 6.

RELATED WORK

Our techniques can be viewed as a natural complement to the bond graph techniques of the sort developed by Ulrich (1988). Our techniques are useful for computing geometry that provides a specified behavior, but because of the inertia-free assumption employed by our simulator, our techniques are effectively blind to energy flow. Bond graph techniques, on the other hand, explicitly represent energy flow but are incapable of representing geometry.

Our techniques focus on the geometry of devices which have time varying engagements (i.e., variable kinematic topology). Therefore, our techniques are complementary to the well know design techniques for fixed topology mechanisms, such as the gear train and linkage design techniques in Erdman and Sandor (1984).

There has been a lot of recent interest in automating the design of fixed topology devices. A common task is the synthesis of a device which transforms a specified input motion to a specified output motion: Kota and Chiou (1992) use a matrix to represent the desired motion transformation, then use matrix decomposition to decompose it into basic building blocks. Subramanian and Wang (1993) use iteratively deepening search to compose a sequence of "abstract mechanisms" that achieves the desired motion transformation. Welch and Dixon (1994) use behavior graphs (similar to bond graphs), and include components in the fluid and electrical domains. For the most part, these techniques synthesize a design using an abstract representation of behavior, then use library lookup to map to implementation. However, because our library contains interacting faces, while theirs includes complete components, we can design interacting geometry, while they cannot. Like SKETCHIT, these techniques produce design variants.

To construct the BEP-Model, we map from qc-space to geometry. Joskowicz and Addanki (1988) and Caine (1993) have also explored the mapping between geometry and c-space. However, they directly modify the shapes of parts using numerical techniques while we use library lookup.

Our work is similar in spirit to research exploring the mapping from shape to behavior. Joskowicz et al (1995) use kinematic tolerance space (an extension of c-space) to examine how variations in the shapes of parts affect their kinematic behavior. Their task is to determine how a variation in shape affects behavior, ours is to determine what constraints on shape are sufficient to ensured the desired behavior. Faltings (1992) examines how much a single geometric parameter can change, all others held constant, without changing the place vocabulary (topology of c-space). Their task is to determine how much a given parameter can change without altering the current behavior, ours is to determine the constraints on all the parameters sufficient to obtain a desired behavior.

More similar to our task is the work of Faltings and Sun (1993). They describe an interactive design system that modifies a user selected parameter until there is a change in the place vocabulary, and hence a change in behavior. Then, they use qualitative simulation to determine if the resulting behavior matches the desired behavior. They modify c-space by modifying geometry, we modify qc-space directly. Because there are many changes in geometry that map to the same change in c-space, their search space is larger than ours. Also, our tool is automatic while theirs is interactive, and we can generate design variants while they do not.

Gupta and Jakiela (1994) describe a novel technique by which a known component "carves out" the shape of an unknown mating component. They require that one of the interacting shapes is known, but we do not.

There is little previous work in sketch understanding. Narayanan et al. (1994) use a diagram of a device to reason about its behavior, but they use a pre-parsed description of

[14]Circular faces are used when rotors act as stops.

[15]If parts rotate through a full revolution, qc-space must have periodic boundary conditions. For example, the qcs-plane for the interaction between a rotating component and translating component becomes a cylinder. Hence, we would not be able to replace rotating parts with translating ones.

the behaviors of each component while we reason directly from the geometry of the interacting faces.

Our work builds upon the research in qualitative simulation, particularly, the work of Faltings (1990), Forbus et al. (1991), and Sacks and Joskowicz (1993). Our techniques for computing motion are similar to the constraint propagation techniques of Stallman and Sussman (1976).

FUTURE WORK

A drawback of our current implementation is that our program will accept a qc-space if there is at least one branch in the simulation that passes through the specified states in the specified order. If one or more of the branches does not provide the desired behavior, obtaining the correct behavior depends on the choice of masses, springs, and actuators, as well as the geometry. In this case, the constraints of the BEP-Model are insufficient to ensure the desired behavior.

A second drawback is that, because our program assumes inertia-free motion with inelastic collisions, the simulator computes approximate dynamics. However, Sacks and Joskowicz (1993) demonstrated that these assumptions are not very restrictive: They examined 2500 mechanisms in an encyclopedia and found that about 80% could be accurately modeled with these assumptions.

Our current solution to both of these drawbacks is to use numerical simulation to verify the final design. Thus we use approximate dynamics during conceptual design and progress to more accurate dynamic analysis (which is computationally more expensive) as more of the design details are selected.

Currently, our techniques are restricted to fixed axis devices, but we believe that this constitutes a significant portion of the variable topology devices used in actual practice. Sacks and Joskowicz (1993) examined 2500 mechanisms in an encyclopedia and found that about 25% of all the devices were fixed axis devices; hence fixed axis devices would account for more than 25% if the examination were restricted to variable topology devices. In addition, we are making progress on extensions to our system which allow it to handle a specific class of non-fixed axis devices. We have identified a commonly occurring class of devices in which a pair of parts has three degrees of freedom (instead of two degrees of freedom as in a fixed axis device) but the configuration space is still tractable.

This work is clearly at an early stage; we have yet to determine how well our techniques will scale to design problems that are more complex than the two working examples reported here.

CONCLUSION

Given the intimate connection between shape and behavior, design of mechanical artifacts is typically conceived of as the modification of shape to achieve behavior. But if changes in shape are attempts to change behavior, and if the mapping between shape and behavior is quite complex (Caine, 1993), then, we suggest, why not manipulate a representation of behavior? Our qualitative c-space is just such a representation. We suggest that it is complete and yet offers a far smaller search space. It is complete because any change in shape will produce a c-space that maps to one of the qc-spaces that our program will consider. Qc-space is far smaller precisely because it is qualitative. Finally, it is an appropriate level of abstraction because it isolates the differences that matter: changes in the topology of qc-space are changes in behavior.

REFERENCES

Caine, M. E., 1993, "The Design of Shape from Motion Constraints," MIT AI Lab. TR 1425, September.

Erdman, A. and Sandor, G., 1984, *Mechanism Design: Analysis and Synthesis*, Vol. 1, Prentice-Hall, Inc., NJ.

Faltings, B., 1990, "Qualitative Kinematics in Mechanisms," *Artificial Intelligence*, Vol. 44, pp. 89–119.

Faltings, B., 1992, "A Symbolic Approach to Qualitative Kinematics," *Artificial Intelligence*, Vol. 56, pp. 139–170.

Faltings, B. and Sun, K., 1993, "Computer-aided Creative Mechanism Design," *Proceedings IJCAI-93*, pp. 1451–1457.

Forbus, K., Nielsen, P., and Faltings, B., 1991, "Qualitative Spatial Reasoning: The CLOCK Project," Northwestern Univ., The Institute for the Learning Sciences, TR #9.

Gupta, R. and Jakiela, M., 1994, "Simulation and Shape Synthesis of Kinematic Pairs via Small-Scale Interference Detection," *Research in Engineering Design*, Vol. 6, pp. 103–123.

Joskowicz, L. and Addanki, S., 1988, "From Kinematics to Shape: An Approach to Innovative Design," *Proceedings AAAI-88*, pp. 347–352.

Joskowicz, L., Sacks, E., and Srinivasan, V., 1995, "Kinematic Tolerance Analysis," *3rd ACM Symposium on Solid Modeling and Applications*, Utah.

Narayanan, N. H., Suwa, M., and Motoda, H., 1994, "How Things Appear to Work: Predicting Behaviors from Device Diagrams," *Proceedings AAAI-94*, Vol. 2, Aug., pp. 1161–1167.

Kota, S. and Chiou, S., 1992, "Conceptual Design of Mechanisms Based on Computational Synthesis and Simulation of Kinematic Building Blocks," *Research in Engineering Design*, Vol. 4, #2, pp. 75–88.

Sacks, E. and Joskowicz, L., 1993, "Automated Modeling and Kinematic Simulation of Mechanisms," *Computer Aided Design*, Vol. 25, #2, Feb., pp. 106–118.

Stahovich, T., 1995, "SKETCHIT: a Sketch Interpretation Tool for Conceptual Mechanical Design," MIT Ph.D. Thesis.

Stallman, R. and Sussman, G., 1976, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," MIT AI Lab. TR 380.

Subramanian, D., and Wang, C., 1993, "Kinematic Synthesis with Configuration Spaces," *The 7th International Workshop on Qualitative Reasoning about Physical Systems*, May, pp. 228–239.

Ulrich, K, 1988, "Computation and Pre-parametric Design," MIT AI Lab. TR-1043.

Welch, R. V. and Dixon, J. R., 1994, "Guiding Conceptual Design Through Behavioral Reasoning," *Research in Engineering Design*, Vol. 6, pp. 169–188.