Interpreting the Engineer's Sketch: A Picture is Worth a Thousand Constraints

Thomas F. Stahovich*
CMU Mechanical Engineering Department
415 Scaife Hall
Pittsburgh, PA 15213
stahov@andrew.cmu.edu

Abstract

We describe a program called SKETCHIT that transforms a single sketch of a mechanical device into multiple families of new designs. To "interpret" a sketch the program first determines how the sketched device should have worked, then derives constraints on the geometry to ensure it works that way. The program is based on qualitative configuration space (qc-space), a novel representation that captures mechanical behavior while abstracting away the particular geometry used to depict this behavior. The program employs a paradigm of abstraction and resynthesis: it abstracts the initial sketch into qc-space then maps from qc-space to new geometries.

Introduction

Drawings have always been an important tool for engineers, with the sketch on a napkin an important and traditional means of thought and communication. Yet to date CAD software has been at best a drafting tool, producing carefully drawn pictures, but neither understanding them the way people do, nor capable of accepting as input an informal sketch of the sort engineers commonly create.

We are working to change that by developing a program that can read, understand, and use sketches of mechanical devices of the sort shown in Figure 1. Our program, called SKETCHIT, is capable of taking a single stylized sketch of a mechanical device and generalizing it to produce multiple new designs.[1]

Engineering sketches, by their very nature, are inaccurate descriptions of a device. Taken literally, the geometry in Figure 1, for example, may not actually produce the desired behavior. Nevertheless, a skilled engineer is able to see how a roughly sketched device was supposed to work and hence what the geometry should have been. In effect, achieving the correct behavior places constraints on the device's geometry. Therefore, to "interpret" the geometry of a sketch, our program first identifies what behaviors the parts should provide, then derives constraints on the geometry to ensure it produces these behaviors.

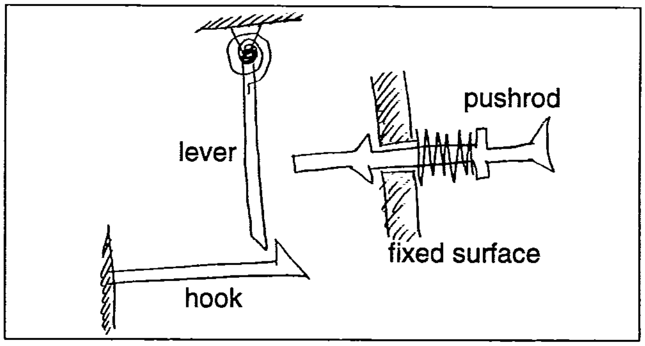


Figure 1: A sketch of a circuit breaker.

To identify the behaviors of the individual parts of a device the program transforms the sketch into a novel representation we call qualitative configuration space (qc-space). Qc-space captures the behavior of the original design while abstracting away the particular geometry used to suggest that behavior. If the sketch as drawn does not produce the desired behavior, the program adjusts the qc-space until it does. The program then uses a library of geometric interactions to transform each identified behavior into new geometry with constraints ensuring that behavior. The constraints define a family of geometries that all produce a particular kind of behavior. Thus, as the program transforms the qc-space back into geometry, it transforms the initial sketch into a family of designs. Because the library may contain multiple implementations for a particular kind of behavior, the program is capable of generating multiple families of new designs. The program represents each new family with what we call a behavior ensuring parametric model ("BEP-Model"): a parametric model augmented with constraints that ensure the geometry produces the desired behavior.[2]

*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.

[1] This paper reports on work previously published in [14].

[2] A parametric model is a geometric model in which the

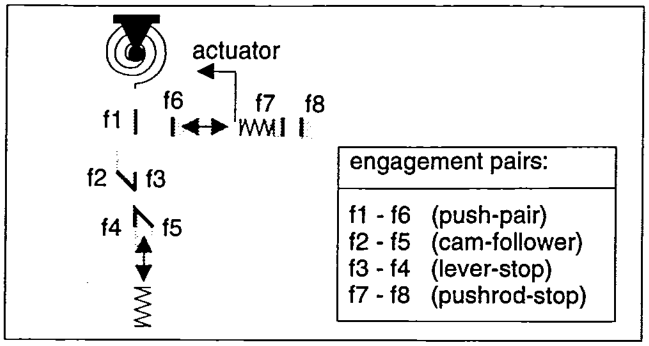


Figure 2: Sketch as actually input to program. Engagement faces are in bold. The actuator represents the reset motion imparted by the user.

We use the design of a circuit breaker (Figure 1) to illustrate the program in operation. 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.

SKETCHIT takes as input a stylized sketch of a device and a state transition diagram describing the desired overall behavior of the device. The later provides guidance in identifying what behaviors the individual parts of the device should provide.

The designer describes the circuit breaker to SKETCHIT with the stylized sketch shown in Figure 2, using line segments for part faces and icons for springs, joints, and actuators. 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–f8). The designer's task is thus 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 overall behavior of the circuit breaker with the state transition diagram in Figure 3. Each node in the diagram is a list of the pairs of faces that are engaged and the springs that are relaxed. The arcs are the external inputs that drive the device. This particular state transition diagram describes how the circuit breaker should behave in the face of heating and cooling the hook and pressing the reset pushrod.

Figure 4 shows a portion of one of the BEP-models that SKETCHIT derives in this case. The top of the figure shows the parameters that define the sloped face on the lever (f2) and the sloped face on the hook (f5). The bottom shows the constraints that ensure this pair of faces plays its role in achieving the overall desired behavior: i.e., moving the lever clockwise pushes

shapes are controlled by a set of parameters.

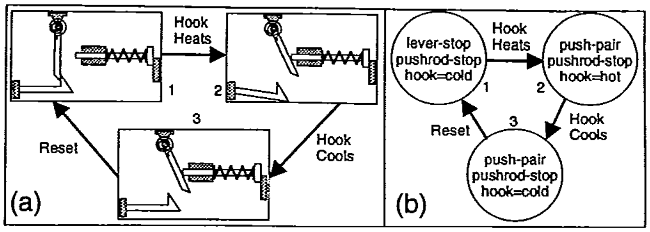


Figure 3: The desired behavior of the circuit breaker. (a) Physical interpretation. (b) State transition diagram. In each of the three states, the hook is either at its hot or cold neutral position.

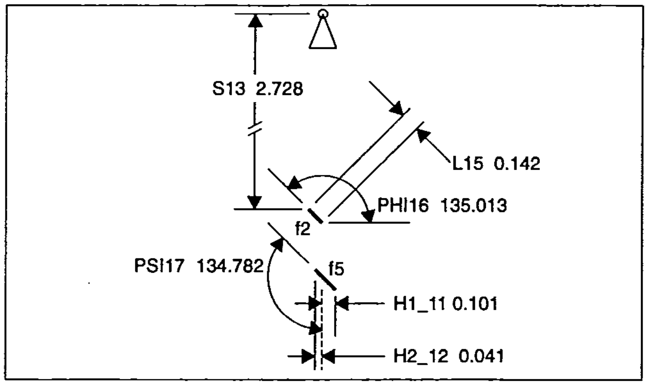
the hook down until the lever moves past the point of the hook, whereupon the hook springs back to its rest position. As one example of how the constraints enforce the desired behavior, the ninth equation, `0 > R14/TAN(PSI17) + H2_12/SIN(PSI17)`, constrains the geometry so that the contact point on face 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 parameter values shown in the top of Figure 4 are solutions to the constraints of the BEP-Model, hence this particular geometry provides the desired behavior. These specific values were computed by a program called DesignView, a commercial parametric modeler based on variational geometry. Using DesignView, we can easily explore the family of designs defined by this BEP-Model. Figure 5, for example, shows another solution to this BEP-Model. Because these parameter values satisfy 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 design approaches.

Figures 4 and 5 show members of just one of the families of designs that the program produces for the circuit breaker. SKETCHIT produces other families of designs (i.e., other BEP-Models) by selecting different implementations for the pairs of interacting faces and different motion types (rotation or translation) for the components. Figure 6 shows an example of selecting different implementations for the pairs of interacting faces: 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 6, the motions are parallel. Figure 7 shows a design obtained by selecting a new motion type for the lever: in the original design the lever rotates, here it translates.

Representation: QC-Space

SKETCHIT's approach to its task is use a representation that captures the behavior suggested by the sketch while abstracting away the particular geometry used to depict this behavior. This allows the program to gen-




```
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 4: Output from the program (a BEP-Model). 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.

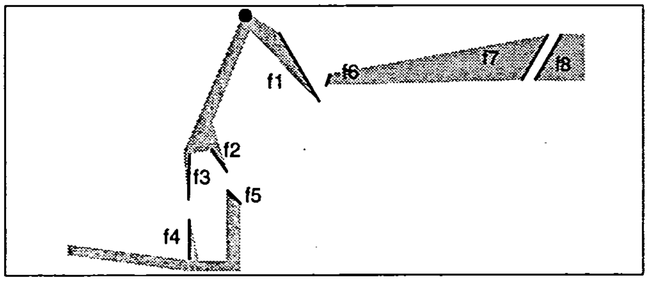


Figure 5: Another solution to the BEP-Model of Figure 4. Shading indicates how the faces might be connected to flesh out the components. This solution shows that neither the pair of faces at the end of the lever nor the pair of faces at the end of the hook need be contiguous.

eralize the initial design by selecting new geometries that provide the same behaviors.

For the class of devices that SKETCHIT is concerned with, the overall behavior is achieved through a sequence of interactions between pairs of engagement faces. Hence the behavior that our representation must capture is the behavior of interacting faces.

Our search for a representation began with configuration space (c-space), which is commonly used to represent this kind of behavior. Although c-space is capable of representing the behaviors we are interested in, it does not adequately abstract away their geometric implementations. We discovered that abstracting c-space into a qualitative form produces the desired effect; hence we call SKETCHIT's behavioral representation "qualitative configuration space" (qc-space).

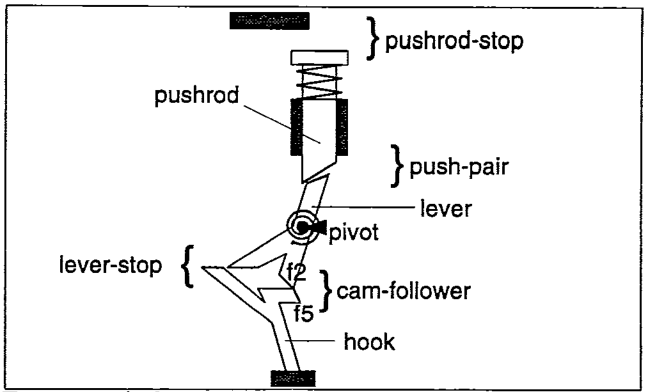


Figure 6: A design variant obtained by using different implementations for the engagement faces. In the position shown, the pushrod is pressed so that the hook is just on the verge of latching the lever.

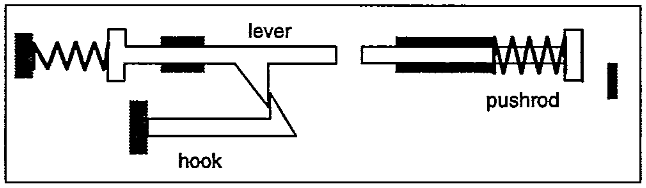


Figure 7: A design variant obtained by replacing the rotating lever with a translating part.

This section begins with a brief description of c-space, then describes how we abstract c-space to produce qc-space.

C-Space

Consider the rotor and slider in Figure 8. If the angle of the rotor U_R and the position of the slider U_S are as shown, the faces on the two bodies will touch. These values of U_R and U_S are termed a *configuration* of the bodies in which the faces touch, and can be represented as a point in the plane, called a configuration space plane (cs-plane).

If we determine all of the configurations of the bodies in which the faces touch and plot the corresponding points in the cs-plane (Figure 8), we get a curve, called a configuration space curve (cs-curve). The shaded region "behind" the curve indicates blocked space, configurations in which one body would penetrate the other. The unshaded region "in front" of the curve represents free space, configurations in which the faces do not touch.

The axes of a c-space are the position parameters of the bodies; the dimension of the c-space for a set of bodies is the number of degrees of freedom of the set. To simplify geometric reasoning in c-space, we assume that devices are fixed-axis. That is, we assume that each body either translates along a fixed axis or rotates about a fixed axis. Hence in our world the c-space for a pair of bodies will always be a plane (a

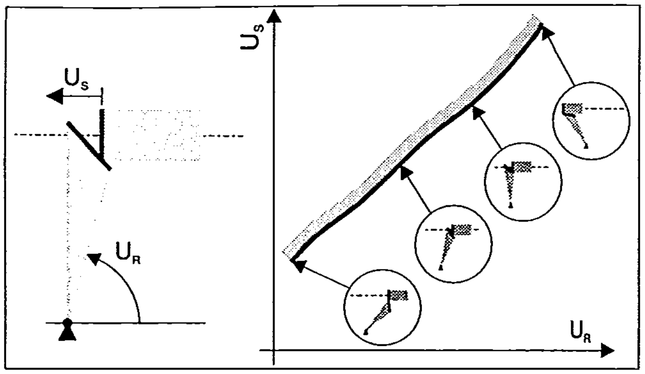


Figure 8: Left: A rotor and slider. The slider translates horizontally. The interacting faces are shown with bold lines. Right: The c-space. The inset figures show the configuration of the rotor and slider for selected points on the cs-curve.

cs-plane) and the boundary between blocked and free space will always be a curve (a cs-curve).[3] However, even in this world, a device may be composed of many fixed-axis bodies, hence the c-space for the device as a whole can be of dimension greater than two. The individual cs-planes are orthogonal projections of the multi-dimensional c-space of the overall device.

Abstracting to QC-Space

C-space is already an abstraction of the original geometry. For example, any pair of faces that produces the cs-curve in Figure 8 will produce the same behavior (i.e., the same dynamics) as the original pair of faces. Thus, each cs-curve represents a family of interacting faces that all produce the same behavior.

We can, however, identify a much larger family of faces 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 (with positive or negative slope), vertical, or horizontal; by relative location we mean relative location of the curve end points.[4]

To see how qualitative slope captures something essential about the behavior, we return to the rotor and slider. The essential behavior of this device is that the slider can push the rotor: positive displacement of the slider causes positive displacement of the rotor. If the motions of the rotor and slider are to be related in this fashion, their cs-curve must be a diagonal curve with positive slope. Conversely, any geometry that maps to a diagonal curve with positive slope will produce the same kind of pushing behavior as the original design.

Their are eight types of qualitative cs-curves, shown in Figure 11. Diagonal curves always correspond to pushing behavior; vertical and horizontal curves correspond to what we call "stop behavior," in which the extent of motion of one part is limited by the position of another.

The key, more general, insight here is that *for monotonic cs-curves, the qualitative slopes and the relative locations completely determine the first order dynamics of the device.* 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 and frictionless.[5] The consequence of this general insight is that qc-space captures *all* of the relevant physics of the overall device, and hence serves as a design space for behavior. It is a particularly convenient design space because it has only two properties: qualitative slope and relative location.

Another important feature of qc-space is that it is constructed from a very small number of building blocks, viz., the different types of qcs-curves in Figure 11. As a consequence we can easily map from qc-space back to geometric implementation using precomputed implementations for each of the building blocks. We show how to do this in Section "Selecting Geometries."

The SKETCHIT System

Figure 9 shows a flow chart of the SKETCHIT system with its two main processes: "Behavior Extraction" and "Constraint & Geometry Synthesis."

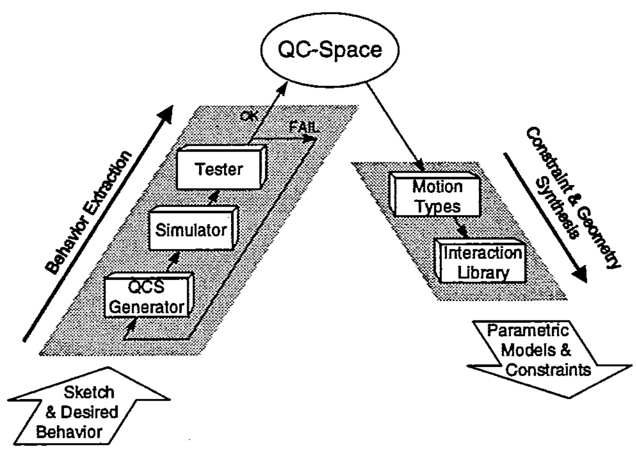


Figure 9: Overview of SKETCHIT system.

[3]The c-space for a pair of fixed-axis bodies will always be 2-dimensional. However, it is possible for the c-space to be a cylinder or torus rather than a plane. See Section "Selecting Motion Type" for details.

[4]We restrict qcs-curves to be monotonic to facilitate qualitative simulation of a qc-space.

[5]"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.

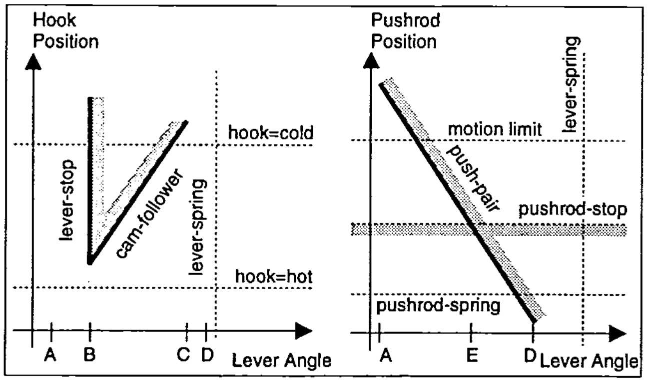


Figure 10: Candidate qc-space for the circuit breaker.

Behavior Extraction Process

SKETCHIT uses generate and test to abstract the initial design into one or more working qc-spaces, i.e., qc-spaces that provide the behavior specified in the state transition diagram.

The generator produces multiple candidate qc-spaces from the sketch, each of which is a possible interpretation of the sketch. The simulator computes each candidate's overall behavior (i.e., the aggregate behavior of all of the individual interactions), which the tester then compares to the desired behavior.

The generator begins by computing the numerical c-space of the sketch, then abstracts each numerical cs-curve into a qcs-curve, i.e., a curve with qualitative slope and relative location.

As with any abstraction process, moving from specific numerical curves to qualitative curves can introduce ambiguities. For example, in the candidate qc-space in Figure 10 there is ambiguity in the relative location of the abscissa value (E) for the intersection between the push-pair curve and the pushrod-stop curve. This value is not ordered with respect to B and C, the abscissa values of the end points of the lever-stop and cam-follower curves in the hook-lever qcs-plane: E may be less than B, greater than C, or between B and C.[6]

Physically, point E is the configuration in which the lever is against the pushrod and the pushrod is against its stop; the ambiguity is whether in this particular configuration the lever is (a) to the left of the hook (i.e., $E < B$) (b) contacting the hook (i.e., $B < E < C$), or (c) to the right of the hook (i.e., $C < E$). When the generator encounters this kind of ambiguity, it enumerates all possible interpretations, passing each of them to the simulator.

The relative locations of these points are not ambiguous in the original, numerical c-space. Nevertheless, SKETCHIT computes all possible relative locations, rather than taking the actual locations directly from the numerical c-space. One reason for this is that it offers one means of generalizing the design: The original locations may be just one of the possible working designs; the program may be able to find others by enumerating and testing all the possible relative locations.

A second reason the program enumerates and tests all possible relative locations is because this enables it to compensate for flaws in the original sketch. These flaws arise from interactions that are individually correct, but whose global arrangement is incorrect. For example, in Figure 2 the interaction between the lever and hook, the interaction between the pushrod and the lever, and the interaction between the pushrod and its stop may all be individually correct, but the pushrod-stop may be sketched too far to the left, so that the lever always remains to the left of the hook (i.e., the global arrangement of these three interactions prevents the lever from actually interacting with the hook.) By enumerating possible locations for the intersection between the pushrod-stop and push-pair qcs-curves, SKETCHIT will correct this flaw in the original sketch.

Currently, the candidate qc-spaces the generator produces are possible interpretations of ambiguities inherent in the abstraction. The simulator and tester identify which of these interpretations produce the desired behavior. We are also working on repairing more serious flaws in the original sketch, as we describe in the Future Work section.

SKETCHIT employs an innovative qualitative simulator designed to minimize branching of the simulation. See [13] and [15] for a detailed presentation of the simulator. The simulator computes the motion of the parts of a device as a trajectory through qc-space.[7]

Constraint & Geometry Synthesis

In the synthesis process, the program turns each of the working qc-spaces into multiple families of new designs. Each family is represented by a BEP-Model.

Qc-space abstracts away both the motion type of each part and the geometry of each pair of interacting faces. Hence there are two steps to the synthesis process: selecting a motion type for each part and selecting a geometry for each pair of engagement faces.

Selecting Motion Type SKETCHIT is free to select a new motion type for each part because qc-space abstracts away this property. More precisely, qc-space abstracts away the motion type of parts that translate and parts that rotate less than a full revolution.[8]

[6]We do not consider the case where $E = B$ or $E = C$.

[7]This process is itself a diagrammatic reasoning task.

[8]Qc-space cannot abstract away the motion type of parts that rotate more than a full revolution because the topology of the qc-space for such parts is different: If one of a pair of parts rotates through full revolutions, its motion will be 2π periodic, and what was a plane in qc-space will become a cylinder. (If both of the bodies rotate through full revolutions the qc-space becomes a torus.) Hence, if a pairwise qc-space is a cylinder or torus, the design must

Changing translating parts to rotating ones, and vice versa, permits SKETCHIT to generate a rich assortment of new designs.

Selecting Geometries The general task of translating from *c-space* to geometry is intractable [1]. However, *qc-space* is carefully designed to be constructed from a small number of basic building blocks, 40 in all. The origin of 32 of these can be seen by examining Figure 11: there are four choices of qualitative slope; for each qualitative slope there are two choices for blocked space; and the qc-space axes q_1 and q_2 can represent either rotation or translation. The other 8 building blocks represent interactions of rotating or translating bodies with stationary bodies.

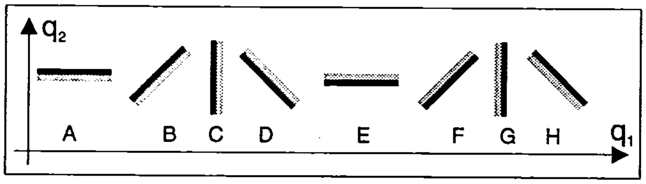


Figure 11: For drawing convenience, diagonal qcs-curves are shown as straight line segments; they can have any shape as long as they are monotonic.

Because there are only a small number of basic building blocks, we were able to construct a library of implementations for each building block. To translate a qc-space to geometry, the program selects an entry from the library for each of the qcs-curves.

Each library entry contains a pair of parameterized faces and a set of constraints that ensure that the faces implement a monotonic cs-curve of the desired slope, with the desired choice of blocked space. Each library entry also contains algebraic expressions for the end point coordinates of the cs-curve.

For example, Figure 12 shows a library entry for qcs-curve F in Figure 11, for the case in which q_1 is rotation and q_2 is translation. For the corresponding qcs-curve to be monotonic, have the correct slope, and have blocked space on the correct side, the following ten constraints must be satisfied:

$w > 0 \qquad L > 0 \qquad h > 0$

$s < h \qquad r > h \qquad \pi/2 < \phi \leq \pi$

$\psi > 0 \qquad \psi < \arcsin(h/r) + \pi/2$

$\arccos(h/r) + \arccos(\frac{L^2+r^2-s^2}{2Lr}) < \pi/2$

$r = (s^2 + L^2 - 2sL\cos(\phi))^{1/2}$

The end point coordinates of the cs-curve are:

$\theta_1 = \arcsin(h/r) \qquad x_1 = -r\cos(\theta_1)$

$\theta_2 = \pi - \arcsin(h/r) \qquad x_2 = -r\cos(\theta_2)$

Figure 13 shows a second way to generate qcs-curve F, using the constraints:

employ rotating parts (one for a cylinder, two for a torus) rather than translating ones.

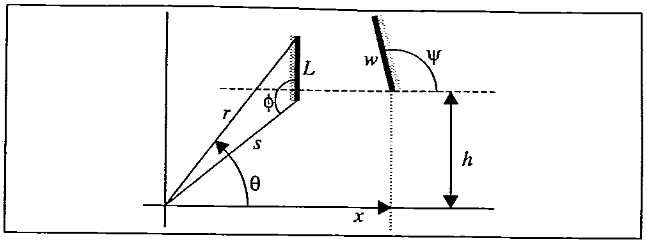


Figure 12: The two faces are shown as thick lines. The rotating face rotates about the origin; the translating face translates horizontally. θ is the angle of the rotor and x, measured positive to the *left*, is the position of the slider.

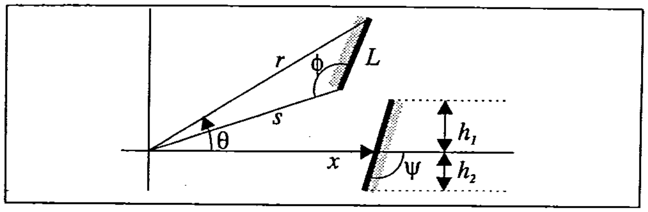


Figure 13: The two faces are shown as thick lines. The rotating face rotates about the origin; the translating face translates horizontally. θ is the angle of the rotor and x, measured positive to the *left*, is the position of the slider.

$h_1 > 0 \qquad h_2 > 0$

$s > h_1 \qquad L > 0$

$\pi/2 < \phi < \pi \qquad \pi/2 < \psi < \pi$

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

The end point coordinates of this cs-curve are:

$\theta_1 = -\arcsin(h_2/r)$

$x_1 = -r\cos(\theta_1) + h_2/\tan(\psi)$

$\theta_2 = \arcsin(h_1/s) + \arccos(\frac{s^2+r^2-L^2}{2sr})$

$x_2 = -s\cos(\arcsin(h_1/s)) - h_1/\tan(\psi)$

In the first of these designs 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.[9] The two designs thus represent qualitatively different implementations for the same qcs-curve.

To generate a BEP-Model for the sketch, we select from the library an implementation for each qcs-curve. For each selection we create new instances of the parameters and transform the coordinate systems to match those used by the actual components. The relative locations of the qcs-curves in the qc-space are turned into constraints on the end points of the qcs-curves. We assemble the parametric geometry fragments and constraints of the library selections to produce the parametric geometry and constraints of the BEP-Model.

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

Our library contains geometries that use flat faces, although we have begun work on using circular faces.[10] We have at least one library entry for each of the 40 kinds of interactions. We are continuing to generate new entries.

SKETCHIT is able to produce different BEP-Models (i.e., different families of designs) by selecting different library entries for a given qcs-curve. For example, Figure 5 shows a solution to the BEP-Model SKETCHIT generates by selecting the library entry in Figure 13 for the cam-follower qcs-curve. Figure 6 shows a solution to a different BEP-Model SKETCHIT generates by selecting the library entry in Figure 12 for the cam-follower. As these examples illustrate, the program can generate a wide variety of solutions by selecting different library entries.

RELATED WORK

There is little previous work in sketch understanding. Narayanan et al. [10] use a diagram of a device to reason about its behavior, but they use a pre-parsed description of the behaviors of each component while we reason directly from the geometry of the interacting faces.

Faltings [5] suggests that a sketch is not a single qualitative model but rather represents a family of precise models. He demonstrates that taking a sketch as a qualitative metric diagram it is possible to compute the "kinematic topology" (an abstraction of the "place vocabulary" [3]). The kinematic topology may contain ambiguities suggesting behaviors that may be obtained by modifying the geometry. Methods for determining which modifications will yield these other behaviors is an open issue.

Our work is closely related to work in design automation. Our techniques can be viewed as a natural complement to the bond graph techniques of the sort developed in [18]. 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 [2].

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 ([11], [17] [19]). 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 contain complete components, we can design interacting geometry, while they cannot. Like SKETCHIT, these techniques produce design variants.

To construct new implementations (BEP-Models), we map from qc-space to geometry. [8] and [1] have also explored the problem of mapping between c-space and geometry. They obtain a geometry that maps to a desired c-space by using numerical techniques to directly modify the shapes of parts. However, we map from qc-space to geometry using library lookup.

Our work is similar in spirit to research exploring the mapping from shape to behavior. [9] uses 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. [4] 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 in [6]. They describe an interactive design system that modifies user selected parameters until there is a change in the place vocabulary, and hence a change in behavior. Then, just as we do, 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. They do a form of generalization by generating constraints capturing how the current geometry implements the place vocabulary; we generalize further by constructing constraints that define new geometries. Finally, our tool is intended to generate design variants while theirs is not.

Our work [15] builds upon the research in qualitative simulation, particularly, the work in [3], [7], and [12]. Our techniques for computing motion are similar to the constraint propagation techniques in [16].

FUTURE WORK

As Section "Behavior Extraction Process" described, the current SKETCHIT system can repair a limited range of flaws in the original sketch. We are continuing to work on techniques for repairing more serious kinds of flaws.

Because there are only two properties in qc-space that matter — the relative locations and the qualitative slopes of the qcs-curves, to repair a sketch, even one with serious flaws, the task is to find the correct relative locations and qualitative slopes for the qcs-curves.

We can do this using the same generate and test

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

paradigm described earlier, but for realistic designs this search space is still far too large. We are exploring several ways to minimize search such as debugging rules that examine *why* a particular qc-space fails to produce the correct behavior, based on its topology. The desired behavior of a mechanical device can be described by a path through its qc-space, hence the topology of the qc-space can have a strong influence on whether the desired path (and the desired behavior) is easy, or even possible. For example, the qc-space may contain a funnel-like topology that "traps" the device, preventing it from traversing the desired path. If we can diagnose these kinds of failures, we may be able to generate a new qc-space by judicious repair of the current one.

We are also working to expand the class of devices that SKETCHIT can handle. Currently, our techniques are restricted to fixed-axis devices. Although this constitutes a significant portion of the variable topology devices used in actual practice (see [12]), we would like extend our techniques to handle particular kinds of non-fixed-axis devices. We are currently working with a commonly occurring class of devices in which a pair of parts has three degrees of freedom (rather than two) but the qc-space is still tractable.

CONCLUSION

We have developed a computer program capable of transforming a stylized sketch of a mechanical device into multiple families of new designs. To "interpret" a sketch, our program first identifies what behaviors the parts should provide, then derives constraints on the geometry to ensure it produces these behaviors. In effect, the program uses physical reasoning to understand the geometry. We have used the program to design a range of devices that includes a circuit breaker and a yoke and rotor mechanism.

One reason this work is important is that sketches are ubiquitous in design. They are a convenient and efficient way to both capture and communicate design information. By working directly from a sketch, SKETCHIT takes us one step closer to CAD tools that speak the engineer's natural language.

REFERENCES

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

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

[3] Faltings, B., 1990, "Qualitative Kinematics in Mechanisms," *JAI*, Vol. 44, pp. 89–119.

[4] Faltings, B., 1992, "A Symbolic Approach to Qualitative Kinematics," *JAI*, Vol. 56, pp. 139–170.

[5] Faltings, B., 1992, Qualitative Models in Conceptual Design: A Case Study, *Reasoning with Diagrammatic Representations*, Papers from the 1992 Spring Symposium, Technical Report SS-92-02, AAAI Press, pp. 69–74.

[6] Faltings, B. and Sun, K., 1996, "FAMING: Supporting Innovative Mechanism Shape Design," *Computer-Aided Design*, 28, pp. 207–216.

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

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

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

[10] 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.

[11] 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.

[12] Sacks, E. and Joskowicz, L., 1993, "Automated Modeling and Kinematic Simulation of Mechanisms," *CAD*, Vol. 25, #2, Feb., pp. 106–118.

[13] Stahovich, T., 1996, "SKETCHIT: a Sketch Interpretation Tool for Conceptual Mechanical Design," MIT AI Lab. TR 1573, March.

[14] Stahovich, T. F., Davis, R., and Shrobe, H., 1996, "Generating Multiple New Designs from a Sketch," *Proceedings AAAI-96*, pp. 1022–29.

[15] Stahovich, T. F., Davis, R., and Shrobe, H., 1997, "Qualitative Rigid Body Mechanics," *Proceedings AAAI-97*, pp. 1038–44.

[16] 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.

[17] 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.

[18] Ulrich, K, 1988, "Computation and Preparametric Design," MIT AI Lab. TR-1043.

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