A Sketch-Based Tool for Analyzing Vibratory Mechanical Systems

Levent Burak Kara
Mechanical Engineering Department,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: lkara@andrew.cmu.edu

Leslie Gennari
ExxonMobil Chemical Company,
4500 Bayway Drive,
Baytown, TX 77520
e-mail: leslie.m.gennari@exxonmobil.com

Thomas F. Stahovich
Mechanical Engineering Department,
University of California,
Riverside, CA 92521
e-mail: stahov@engr.ucr.edu

Sketches are a ubiquitous form of communication in engineering design due to their simplicity and efficiency. However, because of a lack of suitable machine-interpretation techniques, they are virtually unusable with current computer-aided design and engineering tools. The informal nature of sketches and their inherent ambiguity present a number of challenges to the development of such techniques. Here we address one particular challenge, the task of reliably locating and recognizing the intended visual objects from a continuous stream of pen strokes. We present an integrated sketch parsing and recognition approach, based on a novel mark-group-recognize paradigm, which is tailored to the domain of mechanical systems. In the first step of processing, the sketch is examined to identify certain delimiting symbols called "markers." The remaining pen strokes are then partitioned into distinct clusters, each representing a single symbol. Finally, a trainable symbol recognizer is used to find the best interpretation of each cluster. We have used these techniques to build a sketch-based tool for designing and analyzing vibratory mechanical systems. This tool enables designers to analyze and animate vibratory systems by simply sketching them on a tablet computer. User studies indicate that even first-time users find our tool to be effective. [DOI: 10.1115/1.2965595]

Keywords: sketch understanding, pen computing, sketch parsing, symbol recognition, vibratory mechanical systems

1 Introduction

In many disciplines, sketches have great utility as a problem solving tool as they provide a convenient medium for recording elusive thoughts, visualizing and testing emerging ideas, and compactly and efficiently representing a variety of types of information such as spatial, temporal, and causal relationships. Sketches are particularly useful in the early stages of design, where their fluidity and ease of construction enable creativity and the rapid exploration of ideas [1]. In a seminal study of the importance of drawing in mechanical design, Ullman et al. [2] demonstrated that sketches are a particularly useful form of graphical representation and that "computer-aided (CAD) systems must allow for sketching input."

This is consistent with several recent empirical studies demonstrating that sketching has a positive impact on the final outcome of a design project. For example, in a study involving mechanical engineering graduate students, Schütze et al. [3] found a strong positive correlation between the number of sketches drawn in the early phases of design and the quality of the resulting design solution. Based on their studies, they concluded that "digital sketching tools… can create potentially large time and cost savings for computer-aided design in mechanical engineering." Similarly, in a study involving undergraduate engineering students, Yang [4] found a positive correlation between the number of dimensioned sketches drawn early in the design process and the quality of the resulting design. Likewise, Song and Agogino [5] found that the total number of sketches drawn by student design teams has an increasingly positive effect on the design outcome as the design proceeds from preliminary investigation, through conceptual design, to more detailed design. A more recent study by Yang and Cham [6] suggests that the design outcome depends on a broader range of factors than just the number of sketches and that the number of sketches drawn may depend on the particular designer's need for an external representation for visualizing behavior. Other works similarly emphasize the role of sketching in design [7].

Despite the evidence suggesting that sketching is an essential part of engineering design, most contemporary engineering software still cannot work effectively from sketch input. This is due to the lack of suitable machine-interpretation techniques. The informal nature of sketches and their inherent ambiguity present a number of challenges to the development of such techniques. In the present work, we address one particular challenge, the task of reliably locating and recognizing the intended visual objects from a continuous stream of pen strokes. We present an integrated sketch parsing and recognition approach tailored to the domain of mechanical systems. Here, we use the term "parsing" to refer to the task of locating visual objects. Our approach is based on a novel mark-group-recognize paradigm. In the first step of processing, the sketch is examined to identify certain delimiting symbols called "markers." Once these are identified, the remaining pen strokes are partitioned into distinct clusters, each representing a single symbol. Finally, a trainable symbol recognizer is used to find the best interpretation of each cluster.

To demonstrate the utility of our techniques, we have developed a sketch-based tool for designing and analyzing multibody vibratory mechanical systems. This tool allows designers to draw as they would on paper, with minimal constraints imposed by the sketch understanding engine. Unlike paper sketches, however, the sketches created with our tool are "live" in the sense that designers can interact with them to change various model parameters and observe the physical response through live animations of vibratory behavior. The tool's interface (Fig. 1) enables designers to sketch vibratory systems comprised of any number of masses, springs, dampers, forces, and grounds. These objects can be drawn in any order, and each can consist of multiple strokes. Because of our automated parsing technique, the designer is not required to provide explicit cues, such as button presses or pauses, to demarcate symbols. New components can be added to the sketch at any time.

This domain presents several challenges that make it an inter-

Contributed by the Design Theory and Methodology Committee of ASME for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received September 24, 2007; final manuscript received May 14, 2008; published online September 10, 2008. Review conducted by Yan Jin. Paper presented at the ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference (DETC2004), Salt Lake City, UT, September 28–October 2, 2004.

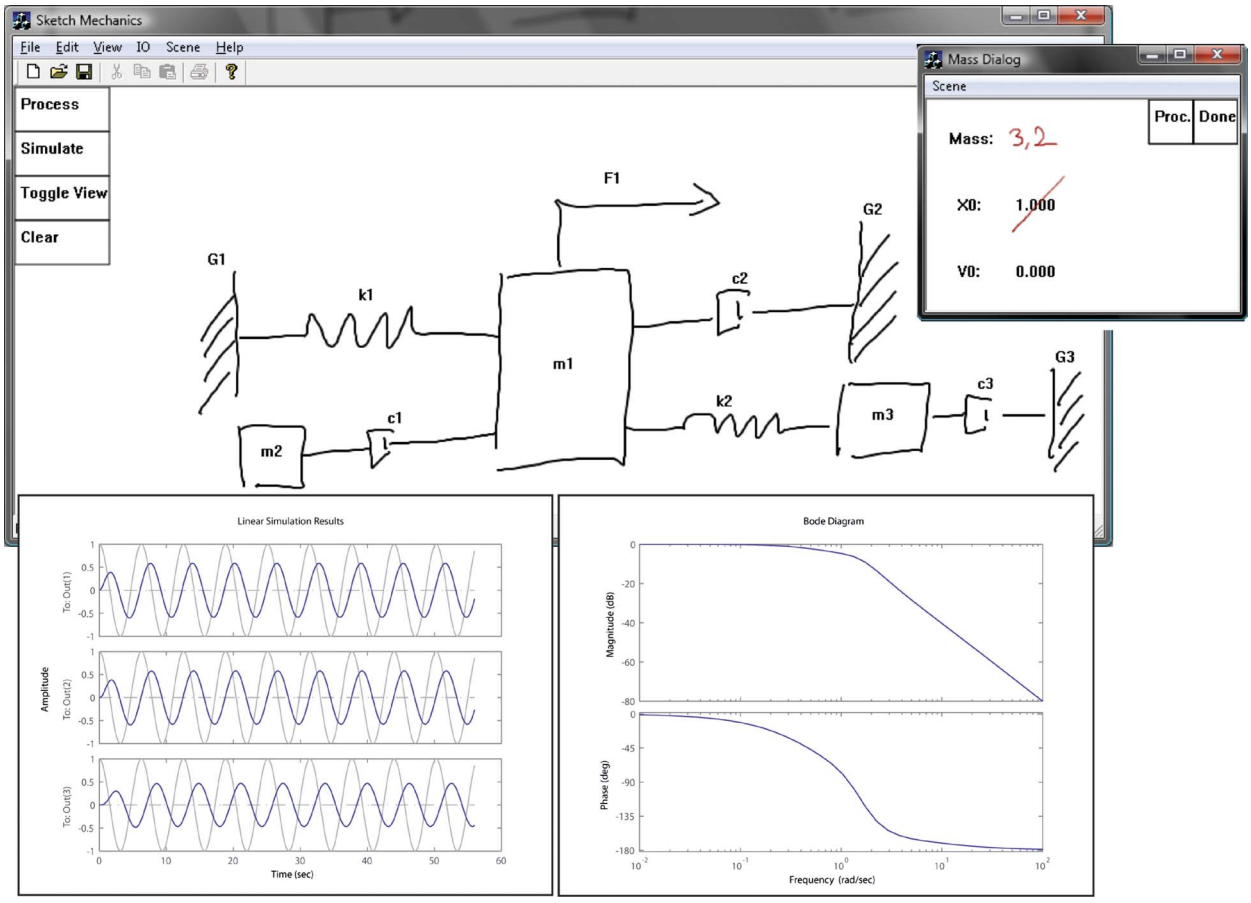

Fig. 1 A typical vibratory system created with our software. The program interprets the sketch, performs a simulation of it, and displays the results in the form of live animations and graphs of performance variables. The user can interact with recognized objects to interactively change their parameters.

esting test of our techniques. First, it features a variety of graphical objects such as masses, springs, and dampers that are typically spatially adjacent and, at times, overlapping. This makes the task of separating objects difficult. Second, different instances of the same object, say, a spring, may exhibit relatively large differences in shape, even within a single sketch drawn by one particular designer. Hence, the shape recognizers used in our system must reliably accommodate such variations, while retaining the ability to differentiate between different types of objects. Third, a wide variety of vibratory systems can be constructed using these objects, thus making sketch interpretation a nontrivial task. Finally, the designer's need to examine the vibratory behavior of the sketched system and interactively adjust the design parameters presents interesting user interface design issues.

In addition to its utility for design, our software may also have applications in engineering education. For instance, our software could enhance the lecture experience by allowing an instructor to analyze and animate vibratory systems by simply sketching them on an electronic whiteboard, just as one would ordinarily sketch them on a conventional blackboard. Similarly, our software enables students to solve problems using the same graphical representations they ordinarily use when solving problems on paper, thus allowing them to focus on problem solving rather than on the details of operating an analysis tool. Although we have not yet assessed the educational value of our software, other recent work suggests that pen-based user interfaces can be effective instructional tools [8].

The remainder of this paper is organized as follows. Section 2 provides a more extensive description of the challenges in automatic sketch interpretation and presents an overview of our solution to those challenges. This is followed by a survey of previous research on sketch-based systems, with an emphasis on parsing and recognition techniques. Next, the details of our parsing and recognition techniques are presented, along with an analysis of their computational complexity. The user interface of our sketch-based design tool is then presented, along with results of a user study evaluating the tool's usability. Finally, a discussion of the work and conclusions are presented.

2 Challenges and Proposed Approach

One of the fundamental challenges in sketch-based computer interaction that distinguishes it from traditional interaction mechanisms has to do with the difficulty of interpreting hand drawings. Unlike text-based or WIMPy (Windows, Icons, Menus, Pointer) input, hand drawing tends to be highly informal, inconsistent, and ambiguous. Thus, for a sketch-based system to be of practical utility, it must robustly cope with the variations and ambiguities inherent in hand drawings so as to interpret the visual scene the way the user intended.

Such difficulties give rise to a variety of challenges in sketch understanding. One concerns *sketch parsing*, the task of grouping a user's pen strokes into the intended symbols without requiring the user to indicate when one symbol ends and the next one begins. This is a difficult problem as the strokes can be grouped in many different ways, and moreover, the number of stroke groups to consider increases exponentially with the number of strokes. To avoid these difficulties, many current systems require the user to

explicitly indicate the intended partitioning of the ink by pressing a button on the stylus or by pausing between symbols [9,10]. Alternatively, some systems require each object to be drawn in a single pen stroke [11]. However, these sorts of constraints typically result in a less than natural drawing experience.

Our approach to parsing is based on a mark-group-recognize paradigm. The first step involves a preliminary recognition process in which the stream of pen strokes is examined to identify markers, symbols that are easily and reliably extracted from a continuous stream of input. The special-purpose recognizers employed in this step achieve high reliability because they are based on the drawing conventions specific to the domain. Once the marker symbols are identified, the remaining strokes are partitioned into distinct clusters. This is done using a general-purpose clustering algorithm that takes as input a cloud of unprocessed pen strokes and groups them into distinct clusters, each corresponding to a single symbol. Note that our parser is provided with no information about the sizes and shapes of the clusters, the number of strokes they might contain, or even the overall number of clusters to be identified.

Another challenge in sketch understanding is *symbol recognition*, the task of recognizing individual hand-drawn objects such as geometric shapes, glyphs, and symbols. The task of differentiating between, say, a damper and a spring symbol is the focus of symbol recognition. In this work, we employ a multistroke, domain-independent, trainable symbol recognizer we developed previously [12]. The recognizer computes the best interpretation of each cluster by comparing it to a database of definition symbols. One advantage of this recognizer is that it can learn a new symbol definition from only a few training examples—even as few as ten examples may be adequate—thus making it possible for users to customize the system to their own drawing styles.

In previous work, we implemented a mark-group-recognize approach that was suitable only for networklike diagrams consisting of symbols linked together with arrows [13]. This approach was used to create a sketch-based interface for MATLAB's SIMULINK software. For that problem, arrows are the marker symbols and clustering relies on grouping each pen stroke with the nearest arrow head or tail. In the current work, we have substantially generalized the approach by the introduction of a clustering method that does not rely on arrows. This greatly extends the usefulness of the approach beyond networklike diagrams.

3 Related Work

Researchers have explored a wide variety of techniques for parsing sketches and other graphical images. For example, Saund et al. [14] presented a system that uses Gestalt principles to determine the salient objects represented in a line drawing. Their work concerns only the grouping of the strokes and does not employ recognition to verify whether the identified groups are, in fact, the intended ones. Jacobs [15] described a system to recognize objects with straight-line perimeter representations. The system uses a number of heuristic rules to group edges that likely come from a single object, and then uses simple recognizers to identify the objects represented by the edges. However, because the techniques require straight-line segments and sharp corners, they may not be well suited to informal hand-drawn sketches. LaViola and Zeleznik [16] described an interface for recognizing handwritten mathematical expressions, but they require explicit gestures to demarcate the various lines of an expression. Notowidigdo and Miller [17] described a system for interpreting structured diagrams such as flowcharts, but their techniques are intended for off-line computation. Costagliola and Deufemia [18] presented an approach based on "left-to-right" (LR) parsing for the construction of visual language editors. They employed "extended positional grammars" to encode the attributes of the graphical objects and presented a set of production/reduction rules for the grammar.

Inspired by the advances in speech recognition, some approaches require visual objects to be drawn with a predefined sequence of pen strokes [19,20]. While useful at reducing computational complexity, the strong temporal dependency of these methods forces the user to remember the correct order in which to draw the pen strokes. Other approaches employ constrained search methods in which a multitude of partial interpretations of an image are generated, and then additional evidence is used to either support or refute the interpretations [21]. Such approaches often face difficulties resulting from nonoptimal thresholds that either prematurely terminate a promising search path or retain a futile one for too long.

Shilman et al. [22] presented an approach to ink parsing that relies on a manually coded visual grammar. The grammar defines composite objects hierarchically in terms of lower level objects. The lowest level objects—individual pen strokes—must be recognizable in isolation (with Rubine's method [23]), although ambiguity can be tolerated. Our approach does not rely on single-stroke shapes. In more recent work, Shilman and Viola [24] improved upon this approach by first generating a multitude of candidate stroke groups, and then evaluating each candidate using a fast bitmap-based recognizer. Their parsing scheme is similar to ours in that strokes are grouped only if they are spatially proximate. Our preliminary recognition step, however, helps us produce only the salient stroke groups whereas their approach uses a produce-many, eliminate-many strategy.

Alvarado and Davis [25] described a parsing approach based on dynamically constructed Bayesian networks. Based on a set of competing hypotheses originating from a set of structural shape descriptions, input strokes are used as evidence that favor certain hypotheses over others. Because of a reliance on precise structural shape definitions during hypothesis evaluation, missing or extra strokes can adversely affect performance. Our parsing and recognition techniques are less sensitive to noise at that level because our system separates the task of parsing and recognition. Our parser requires only that a subset of the symbols—the marker symbols—be recognized accurately.

Researchers have developed a wide variety of approaches to shape and symbol recognition. Some approaches require the entire shape or symbol to be drawn in a single stroke [23,26]. For example, the method in Ref. [1] uses conic sections to classify pen strokes as lines, arcs, and corners. Systems such as those in Refs. [27,20] allow for multistroke symbols but require shapes to be drawn with a consistent pen stroke order. Some other multistroke recognizers allow for variable stroke order but are hard coded [28,29]. Systems such as those in Refs. [9], [30], and [31] are trainable but typically require a multitude of training examples. In contrast, our recognizer in insensitive to stroke order is suitable for both single and multistroke symbols and can learn a definition from only a few prototype examples. Kara and Stahovich [32] presented a multistroke recognizer based on a down-sampled bitmap representation. The approach is well suited to "sketchy" drawings, such as those with overstroking. However, as the method is based on geometry and not on topology, it is sensitive to nonuniform scaling and large variations in shape.

4 Overview of the Approach

Figure 2 illustrates the overall process for interpreting a sketch of a vibratory system. The first step is a preliminary recognition process that examines the stream of pen strokes to identify marker symbols. Once these are identified, the remaining strokes are partitioned into clusters, each representing a single symbol. Next, a general-purpose symbol recognizer is used to determine which component each cluster represents. Finally, the connectivity of the components is examined, enabling a mathematical model of the vibratory behavior of the sketched system to be derived. Sections 5–8 describe each of these steps in detail.

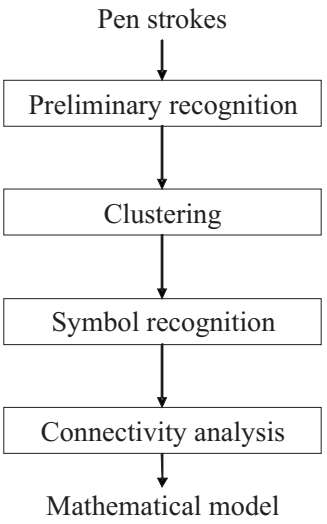

Fig. 2 Process for interpreting a sketch of a vibratory system

5 Preliminary Recognition

One key to successful sketch understanding lies in the ability to establish the ground truths about the sketch early on before costly interpretation errors occur. Our approach is based on the use of marker symbols, symbols that are easy to recognize and that can guide the interpretation of the remainder of the sketch. This approach is similar in spirit to the construction of "islands of certainty" in the Hearsay-II speech understanding system [33].

In the domain of discrete vibratory systems, we have found mass and ground symbols to be good marker symbols as they possess a number of unique geometric and temporal characteristics that facilitate their recognition. (Note that our work is focused on schematic rather than on pictorial sketches of devices.) For example, masses are invariably drawn as closed loops, while ground symbols are characterized by a sequence of short parallel line segments corresponding to the hatches. In the first step of analysis we exploit these features to identify the masses and grounds in the sketch.

Recognizing Masses. Identifying mass symbols involves finding sets of consecutively drawn strokes that connect end to end to form closed loops. To determine if a particular set of strokes forms such a loop, our program constructs a fully connected *stroke chain* comprised of the original strokes and a set of hypothetical linkages between them. The latter are formed by joining each stroke endpoint to the nearest endpoint of some other stroke. For example in Fig. 3, endpoint A of stroke 1 is connected to endpoint A' of stroke 2 with the hypothetical linkage AA' because these two endpoints are closer to each other than to any other endpoints. In a perfect closed loop, all strokes would be connected precisely at their endpoints and the total "linkage length," defined

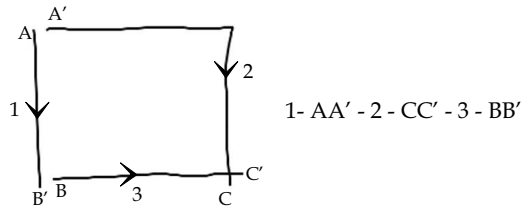

Fig. 3 Masses are identified as "stroke chains." The mass on the left is described by the stroke chain on the right. The mass recognizer is insensitive to drawing order and direction. (The numbers next to the strokes indicate the drawing order, and the arrowheads indicate the drawing direction.)

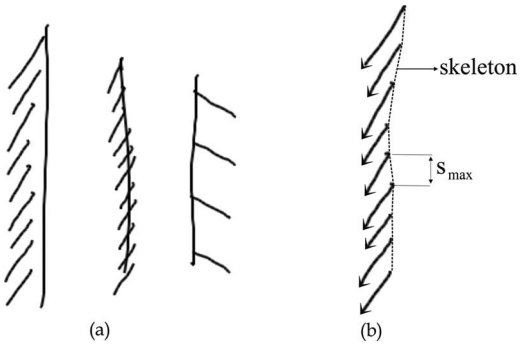

Fig. 4 (a) Examples of ground symbols our system can recognize. (b) The definition of a ground symbol is based on the length of the skeleton and the orientation of and separation between hatches.

as the sum of the lengths of the hypothetical linkages, would be zero. (In Fig. 3, the total linkage length is $|AA'|+|BB'|+|CC'|$.) However, to account for "sketchiness," we use a thresholded criterion that accepts a closure if the total linkage length is less than or equal to 10% of the total length of the pen strokes.[1] For strokes that do not form a closed loop, the linkage length is typically much larger. For example, it is 100% for a straight line and can be greater than 100% for an arbitrary stroke set.

Using this approach, our program identifies closed loops comprised of up to five consecutively drawn strokes,[2] including single-stroke loops. Although the strokes comprising a particular mass must be drawn consecutively, they can be drawn in any order and direction. Also, this approach can successfully recognize arbitrarily shaped bodies and is not restricted to simple patterns such as rectangles. After identifying the closed loops, our program instantiates the mass objects they represent and marks the associated strokes as processed to prevent them from later being considered as parts of other components.

Recognizing Grounds. After identifying the masses in the sketch, our program focuses attention on the ground symbols. These are distinguished by a set of short parallel line segments (i.e., the hatches) that are aligned approximately along a straight line (Fig. 4). Moreover, the segments are almost always drawn consecutively and in a uniform orientation, such as from top to bottom or vice versa. Our program thus searches for such geometric and temporal patterns to locate the ground symbols. To prevent arbitrary parallel strokes from being recognized as grounds, we require a minimum of four hatches before a ground symbol is conjectured. To test whether a group of strokes constitutes a ground symbol, the program determines if (1) they are roughly uniformly separated, (2) they are more or less parallel, and (3) the imaginary polyline formed by connecting their starting points, which we call the skeleton, is approximately a straight line. The first requirement is satisfied if the separation between the pair of most distant consecutive strokes (s_{max}) is less than twice the average separation distance. The second requirement is satisfied if the vectors defined by connecting each stroke's first point to its last point are all oriented toward the same quadrant, for example, southwest in Fig. 4(*b*). The last requirement is satisfied if the length of the skeleton is within 5% of that of a line connecting the starting point of the first stroke to the starting point of the last stroke. Once a sequence of four strokes satisfying these require-

[1]This threshold was selected empirically. A larger threshold would make the system more tolerant of drawing errors but could lead to false positives. A smaller threshold would tend to reduce false positives but could increase false negatives.

[2]In an informal study involving a few engineering students, we observed that users typically draw masses with two or three strokes. An upper limit of five strokes was selected as a means of reducing computational cost without substantial risk of false negatives. This limit has worked well in practice but could be increased if false negatives become a problem.

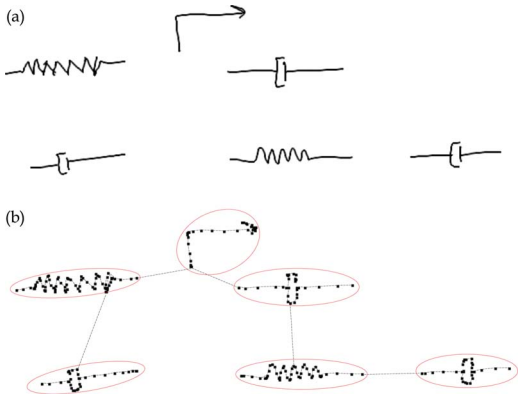

Fig. 5 (a) The objects remaining to be identified once the masses and the grounds in Fig. 1 have been recognized. (b) The clustering algorithm is run until a single cluster is obtained. The intended clusters, which are encircled with ellipses, are then identified by examining the distance between the two clusters merged at each iteration.

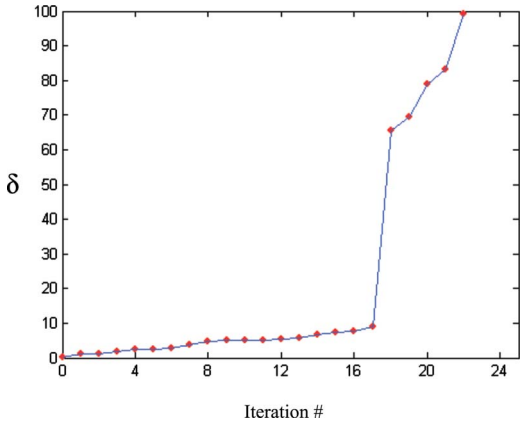

Fig. 6 The dissimilarity score δ increases monotonically with the number of iterations. Sharp leaps, such as the one at iteration 17, usually correspond to forced merges and thus can be used to determine the number of natural clusters.

ments has been found, the program determines the extent of the pattern by appending subsequent strokes, one at a time, until the pattern is disrupted. Finally, the stroke representing the skeleton is located and added to the pattern. After identifying the ground symbols, the program instantiates the ground objects they represent and marks the associated strokes as processed to prevent them from later being considered as parts of other components.

6 Clustering

In the previous step, the masses and grounds are identified, but the remainder of the sketch, which consists of springs, dampers, and forces, is left uninterpreted. Eliminating the masses and grounds from the sketch tends to make these other symbols more distinct. This is a particularly important property of the mark-group-recognize approach. Figure 5(*a*), for example, shows what is left after the masses and grounds are removed from the sketch in Fig. 1. We divide the task of identifying the remaining symbols into two subtasks. The first is *stroke clustering*, in which the strokes are grouped into clusters corresponding to distinct symbols. The second is *recognition*, in which each cluster is classified using the symbol recognizer described in Sec. 7.

The goal of clustering is to group the unprocessed strokes (i.e., those not belonging to masses or grounds) into clusters, such that each cluster contains only strokes belonging to a single symbol. For example, in Fig. 5, there are six clusters corresponding to the two springs, the three dampers, and the force. There are four issues that complicate the problem. The first is that distinct clusters can be arbitrarily close to one another. Hence there is no fixed threshold distance below which two strokes would be considered part of the same cluster. Second, clusters can have arbitrary sizes and shapes. Third, each cluster may contain an arbitrary number of strokes. Fourth, and most importantly, there is no means of knowing, a priori, the number of clusters to be located.

Our clustering approach relies on the fact that once the masses and grounds have been removed from the sketch, the remaining clusters typically occur at spatially distinct regions and do not overlap. Furthermore, while the distance between two clusters is arbitrary, it is usually greater than the distance between the strokes within a cluster. Hence, our clustering technique partitions the strokes into clusters, such that each cluster contains strokes that are nearer to one another than they are to other clusters. To implement this idea, we have adopted the agglomerative hierarchical clustering algorithm described in Ref. [34]. Note that while this algorithm has been applied to other problems, it is useful in this domain precisely because the preliminary recognition process accentuates the separation between clusters.

The clustering procedure is facilitated if the sketch is viewed as a collection of data points rather than of pen strokes. In this representation, each data point initially forms a distinct seed cluster. The algorithm takes as input these seed clusters and repeatedly merges them until a single all-encompassing cluster is obtained. At each iteration, the two nearest clusters are merged, resulting in a bigger cluster containing the data points from the two clusters. Each iteration thus reduces the number of clusters by 1.

For the purpose of merging clusters, we define the distance between two clusters as the minimum distance between a point in one cluster and a point in the other. The distance between two clusters A and B can be expressed more formally as

$$d(A,B) = \min_{a \in A, b \in B} \|a - b\| \tag{1}$$

where $\|a-b\|$ is the Euclidean distance between point a in cluster A and point b in cluster B. In this formulation $d(A,B)$ is known as the *nearest-neighbor* distance. At each iteration of the clustering algorithm, the program computes this distance for all pairs of clusters and merges the pair having the minimum distance. Although other distance measures could be used, such as farthest neighbor, we have found the nearest-neighbor measure to be most suitable as it favors thin and elongated clusters due to a phenomenon called "chaining" [34]. The springs, dampers, and forces in our domain often benefit from this effect.

Because we assume that each pen stroke belongs to a single symbol, we accelerate the clustering process by merging all seed clusters originating from a given pen stroke. Once this is done, the usual merging process continues as before. We have found this to greatly reduce the amount of computation needed for clustering.

The lack of a priori knowledge about the number of clusters to be identified presents a challenge for this analysis. If this number were known, the clustering algorithm could be terminated when the desired number of clusters was located. In our case, however, this number must be determined automatically. Fortunately, the clustering algorithm provides a means to accomplish this. The distance between the pair of clusters merged at each iteration can be stored as a dissimilarity score δ. Because the nearest clusters are merged at each iteration, δ monotonically increases with the number of iterations. The key, however, is that a large increase in δ is usually indicative of a "forced merge" [34]—a merge that combines two distant clusters—and thus can be used as a stopping criterion.

We exploit this observation to determine the number of distinct symbols in a sketch. Consider Fig. 6 that shows the dissimilarity

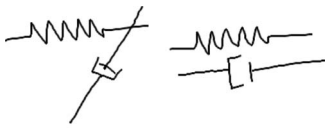

Fig. 7 The clustering algorithm fails when symbols overlap or when intrasymbol distances are comparable to intersymbol distances

score versus the iteration number obtained from the sketch in Fig. 5. The large jump from iteration 17 to 18 corresponds to the merging of the force symbol with the nearby damper. The subsequent iterations combine the remaining clusters until a single cluster is obtained. Clearly the intended clusters are those obtained by iteration 17. By finding such sharp "leaps" in δ, we can thus identify the best stopping iteration.

Our criterion for the stopping iteration i^* is as follows:

$$i^* = \arg\max_i \left[\frac{\delta_{i+1} - \delta_i}{\delta_i - \delta_{i-1}} \cdot (\delta_{i+1} - \delta_i) \right] \tag{2}$$

The ratio term in this expression compares the increase in δ observed in one iteration to that observed in the previous iteration. This is useful for detecting sharp leaps in δ such as the one that occurs at iteration 17 in Fig. 6. However, because the ratio measures only the relative increase, if the increase in the previous iteration was minute, even a small increment in the subsequent iteration may undesirably extremize the ratio. This often occurs during the initial iterations. To prevent such occurrences from dictating the stopping iteration, we favor globally large leaps over smaller ones by using the absolute amount of leap $(\delta_{i+1} - \delta_i)$ as a scaling factor.

The clustering method described above works best when the symbols form compact clusters at spatially distant locations. It naturally allows symbols to be drawn with an arbitrary number of strokes and is insensitive to the absolute angular orientation of a symbol or the angular orientation of one symbol relative to another. However, it is not well suited to situations in which different symbols overlap (Fig. 7(*a*)), or when an internal gap in a symbol is comparable in size to the distance to a neighboring symbol (Fig. 7(*b*)). In the first case, the algorithm will simply produce erroneous clusters. In the second case, the right number of clusters will not be determined reliably as the leap from intracluster merges to intercluster merges will not be as distinct as in Fig. 6. Although the first of these issues is uncommon in our domain (springs, dampers, and forces usually do not overlap), occasionally the second issue does cause errors. We have found that most of these errors can be alleviated by keeping the gaps in dampers to a minimum. Although currently not implemented, our system could exploit the *temporal* proximity of the strokes to resolve such problems (strokes belonging to different symbols are likely to be temporally separated) or could provide the user with a means to interactively rectify mistakes when they occur. We plan to explore these alternatives in the future.

7 Symbol Recognition

Once the clusters have been identified, we must determine which symbols they represent. We have developed a general-purpose trainable symbol recognizer for this task [12]. The recognizer takes as inputs the raw strokes in a cluster and outputs the domain object that best matches those strokes. Because masses and grounds are already identified in the preliminary recognition step, the symbol recognizer presented here is used for distinguishing between springs, dampers, and forces only. However, the relatively small number of patterns to consider in our working example should not obscure the utility of the recognizer. For example, Kara [35] reported on results of an earlier version of the recognizer with a library of 24 engineering symbols.

Segmentation. Our recognizer first decomposes the raw strokes into line and arc segments that match the original ink. This process, called segmentation, provides compact descriptions of the pen strokes that facilitate recognition. Segmentation involves searching along each stroke for "segment points," points that divide the stroke into different geometric primitives. These points are distinguished by both the kinematics of the pen tip during drawing and the shape of the resulting ink. Segment points are generally points at which the pen speed is a minimum, the ink exhibits high curvature, or the sign of the curvature of the ink changes (the details can be found in Ref. [36]). Once the segment points have been identified, a least squares analysis is used to fit lines and arcs to the ink between the segment points.

Training. Our recognizer uses a feature-based statistical learning technique to learn new symbol definitions. To train the recognizer, the user draws several examples of a symbol. Each example can be sketched using any number of strokes drawn in any order. The examples need not be drawn the same size or at the same orientation since the recognizer is insensitive to size and rotation and is robust to moderate nonuniform scaling.

A set of nine geometric features is extracted from the segmented version of each training example. The first eight features include the number of pen strokes, the number of line segments, the number of arc segments, the number of endpoint (L) intersections, the number of midpoint (X) intersections, the number of endpoint-to-midpoint (T) intersections, the number of pairs of parallel lines, and the number of pairs of perpendicular lines. To account for the sketchiness of a drawing, tolerances are used when determining if two segments intersect or if two segments are parallel or perpendicular.

The final feature is the average distance between endpoints of the segments. This feature gives information about the relative size and spacing of segments. This average distance is computed by determining the distance from each endpoint of every segment to each endpoint of every other segment. This value is averaged and is then normalized by the maximum distance between any two endpoints, thus accounting for scaling. The average distance between endpoints is insensitive to rotation. Unlike the other eight features, which can assume only discrete values, this feature is continuous valued.

Once these nine features have been computed for each of the training examples of a symbol, a statistical definition model is constructed. We assume that the training features are distributed normally, i.e., they can be modeled as Gaussian distributions. A Gaussian model naturally accounts for variations in the training examples. However, because eight of the features assume only discrete values, and moreover we aim to use only a handful of training data, the continuous Gaussian models we use are not theoretically appropriate. Nevertheless, our empirical results show that these models produce highly favorable recognition rates for the types of symbols considered.

Recognition. The first step in recognizing an unknown symbol, S, is to extract the same nine features used to describe the training examples. The values of these features are then compared to those of each learned definition, D_i. Symbol S is classified by the definition D^* that maximizes the probability of match:

$$D^* = \arg\max_i P(D_i|S) \tag{3}$$

Here we assume that all definitions are equally likely to occur; hence we set the prior probabilities of the definitions to be equal. We also assume that the nine geometric features (x_j) are independent of one another. Otherwise, a much larger number of training examples would be required for classification. With these assumptions, Bayes' rule states that the definition that best classifies the

symbol is the one that maximizes the likelihood of observing the symbol's individual features:

$$D^* = \arg\max_i \prod_j P(x_j|D_i) \tag{4}$$

As stated above, we assume each statistical definition model $P(x_j|D_i)$ to be a Gaussian distribution with mean $\mu_{i,j}$ and standard deviation $\sigma_{i,j}$:

$$P(x_j|D_i) = \frac{1}{\sigma_{i,j}\sqrt{2\pi}} \exp\left[-\frac{(x_j - \mu_{i,j})^2}{2\sigma_{i,j}^2} \right] \tag{5}$$

Because the features are assumed to be independent, this is referred to as a naive Bayesian classifier. This type of classifier is commonly thought to produce optimal results only when all features are truly independent. This is not a proper assumption for our system since some of the features we use are interrelated. For example, the number of intersections in a symbol frequently increases with the number of lines and arcs. However, Domingos and Pazzani [37] suggested that the naive Bayesian classifier does not require independence of the features to be optimal. While the actual values of the probabilities of match may not be accurate, the rankings of the definitions will most likely be correct. Our empirical studies have shown this to be the case for our problem.

Because of our assumption of a Gaussian distribution, definitions in which the training examples show no variation in one or more features cause difficulty during recognition. This situation is a common occurrence because we often rely on only a small number of training examples and because eight of the features used for classification can assume only discrete values. To prevent definitions from becoming overly rigid in this way, we require that all features, with the exception of the continuously valued average distance between endpoints, have a standard deviation of at least 0.3. This significantly increases recognition rates, especially when only a few examples have been used for training.

8 Connectivity Analysis

The final interpretation step is determining how the recognized components are connected to one another. This is accomplished in a straightforward fashion by considering geometric proximity. The ends of a spring or damper are assumed to be connected to the nearest masses and grounds. Likewise, each force is assumed to be connected to the mass nearest its end. For example, in Fig. 1, the right end of spring $k1$ is connected to mass $m1$ because no other masses or grounds are closer. For this analysis, we define the distance between an endpoint and a mass or ground as the Euclidean distance between that point and the center of the coordinate-aligned bounding box of the mass or ground. We require that each end of a spring or damper be connected to precisely one mass or one ground symbol. Each mass or ground, however, may have an arbitrary number of springs or dampers attached to it. Currently, our analysis excludes the case in which springs and dampers are connected end to end.

The connectivity analysis is designed to avoid the pitfalls that occur when a sketch is interpreted literally. Our goal is to infer the *intended* vibratory model, despite errors and ambiguities in the sketch. For instance, in Fig. 1, although $c1$ and $m1$ are not actually attached, our program decides, just as anybody seeing the sketch would, that the two are intended to be connected. A literal interpretation, on the other hand, would consider the two components to be disconnected.

After determining the connectivity between components, our program constructs the equations of motion. For the discrete, linear, and time-invariant systems we consider, these equations are conveniently described in terms of mass, damping, and stiffness matrices; the displacement and forcing vectors; and the initial position and velocity vectors. To simplify the generation of these equations, we assume that each mass has 1D motion along the horizontal direction. This assumption is not a limitation of our sketch interpretation techniques but rather avoids issues related to computing simulations, which is not the focus of this work. The system equations are simulated by MATLAB, which runs in the background. The solution is a displacement vector, the elements of which are the displacements of each of the masses as a function of time. These results are displayed to the user in the form of graphs and as an animation of the user's sketch in which masses translate and dampers and springs stretch and compress (Fig. 1).

9 Complexity Analysis

The preliminary recognition step identifies the mass and ground symbols in the sketch. We assume that these symbols are formed by *consecutively* drawn strokes. With this assumption, the computational complexity of detecting these symbols is $O(k \cdot n)$, where k is the maximum number of strokes that a symbol can contain and n is the total number of strokes in the sketch. k is typically a small number. For instance, the mass recognizer searches for closed contours containing up to five strokes, and hence k is 5. Likewise, although the ground recognizer does not have an explicit upper limit on the number of strokes to be considered, in practice the number of hatches in a ground symbol rarely exceeds about seven strokes.

If the requirement of temporal consecutiveness were relaxed in the detection of these symbols, all stroke groups containing up to k strokes would need to be considered. In that case, the cost of identifying the masses and grounds would be $\Sigma_{i=1}^{k}\binom{n}{k} = O(n^k)$, resulting in an inefficient procedure. Although the requirement of consecutiveness imposes some constraint on drawing, we believe that this constraint is balanced by the significant reduction in computational complexity that is achieved. Note that this requirement applies only to marker symbols. The remaining symbols need not be drawn with consecutive strokes.

The complexity of the stroke clustering algorithm is $O(n^3)$. This is because at each iteration of the algorithm, we identify the nearest two clusters in a naive way by considering all cluster pairs. Each iteration thus has $O(n^2)$ complexity. (Although not implemented, it is possible to compute the two nearest clusters with more efficient algorithms that run in $O(n \log n)$.) This process is repeated until a single all-encompassing cluster is obtained, which requires $O(n)$ iterations. In actuality, the number of clusters is reduced by 1 at each iteration; thus the overall complexity is subcubic.

Finally, the complexity of the recognition step is $O(c \cdot d)$, where c is the number of stroke clusters and d is the number of domain symbols. This is because the recognizer compares every cluster to every symbol definition.

To provide a sense of the kind of performance actually achieved in practice, Fig. 8 shows performance data from three different sketches. These sketches contain between 18 and 148 pen strokes. For comparison, the sketch in Fig. 1 contains 44 strokes, 6 marker symbols, and 6 clusters, which is slightly less complex than Case 2 in the table. The table lists the amount of CPU time required for preliminary recognition, clustering, and symbol recognition. It is clear from these examples that the performance is quite acceptable for interactive use. Even with 148 pen strokes, processing took only a little more than 3 s on a CPU that was not particularly fast.

10 User Interaction

Our software is deployed on a 9×12 in.^{2} Wacom Cintiq digitizing tablet with a cordless stylus. This tablet is also a liquid crystal (LCD) display, which enables users to see virtual ink directly under the stylus, thus providing a working environment similar to pen and paper. The tablet provides time stamped data packets containing the coordinates of the stylus tip. Additionally, the stylus has two buttons located along its shaft, which provide functionality similar to that of mouse buttons.

After completing the drawing, the user initiates interpretation of the sketch by tapping the "process" button located at the top left

		Preliminary Recognition	Stroke Clustering	Symbol Recognition
Case 1	Number of Strokes = 18 Number of Markers= 2 Number of Clusters = 3	152 ms.	Less than 14 ms.	326 ms.
Case 2	Number of Strokes = 52 Number of Markers= 6 Number of Clusters = 6	420 ms.	55 ms.	581 ms.
Case 3	Number of Strokes = 148 Number of Markers= 20 Number of Clusters = 18	1392 ms.	1170 ms.	627 ms.

Fig. 8 Processing times of the various program modules for three different sketches. All times are in milliseconds. "Number of markers" includes both ground and mass symbols. Experiments conducted on a 1.7 GHz Pentium 4 machine with 256 Mbytes of RAM.

corner of the drawing surface. At this point, the program processes the collection of strokes and identifies the mechanical components. The program demonstrates its understanding by displaying unique text labels next to the identified components. The labels are similar to those an engineer might use. For example, $k1$ indicates that the component is a spring and, furthermore, that it was the first spring drawn. Similarly, $m3$ indicates that the component was the third mass drawn. A default value of 1 is assigned to the relevant properties of each spring, damper, and external force. For example, each spring is assigned a stiffness of 1 N/m, and each damper is assigned a damping constant of 1 N s/m. External forces have the form $F_o \cos(\omega \cdot t)$, with $F_o = 1$ N and $\omega = 1$ rad/s. Masses are assigned mass values proportional to their size. The geometrically largest mass symbol is assigned a mass of 1 kg, while the remaining ones receive proportionally smaller values. For example, a mass symbol half the size of the largest one is assigned a mass of 0.5 kg.

Once the sketch is interpreted, users can study the system behavior directly from the sketch interface. For example, the user can run a live animation by tapping the "simulate" button on the drawing surface. When the user does this, the sketch itself is animated: The masses move and the springs and dampers stretch and compress. The simulation results are also displayed in the form of graphs. As shown in Fig. 1, the graphical output consists of position versus time plots and the frequency response of the system.

The user can change the default parameters of an object by pointing to it with the stylus and clicking one of the stylus' buttons. Figure 1 shows an example. Here, the user has clicked on a mass symbol, bringing up a dialog box specialized to masses. This dialog box contains fields for editing the mass value, the initial position, and the initial velocity (the latter are 0 by default). Interaction in these dialog boxes is also sketch based in that users can change existing parameter values by crossing out the old ones with a delete gesture (a stroke through the number) and simply writing in the new values. The new values are recognized using an image-based symbol recognizer we previously developed [32] and are then displayed in computer fonts. Similar dialog boxes exist for the other kinds of components. Changes made to the properties of an object are automatically transferred to MATLAB and a new simulation is performed.

The user has the option of viewing the model in its original sketchy form or in a cleaned up iconic form. The iconic form preserves the size of the original shapes. The user can toggle between these two views by tapping the "toggle view" button on the drawing surface. Users of our system have indicated that the informality of the sketchy view gives a sense of freedom and creativity, while the iconic view gives a sense of completeness and definiteness.

11 Evaluation and Discussion

We asked 13 subjects, most of them were graduate and undergraduate mechanical engineering students, to sketch two vibratory systems.[3] Each participant provided two sketches of each system for a total of four sketches. Figure 9 shows four typical sketches from the study and their corresponding interpretations. Participants had very little or no experience with the LCD tablet. Moreover, the test was conducted in a walk-up-and-draw fashion in which participants were nearly immediately asked to start drawing. Only a brief warm-up period of about 30 s was given to allow a participant to become familiar with the stylus and LCD tablet. No explanation was given about how the program performs its task. For example, participants were not told that the system begins by looking for closed loops and hatches to identify masses and ground symbols. Each session involved only data collection. The data were processed at a later time so as to prevent the participants from adjusting their drawing style based on our program's output.

The initial results indicate that we have a sound parsing and recognition approach. While our parsing algorithm worked quite successfully, when it did fail, it was due to the phenomenon illustrated in Fig. 7(b), in which symbols are too close to one another. For the sketches in which parsing was successful, we found our feature-based symbol recognizer to be highly accurate, despite the fact that none of the participants were involved in training the recognizer. The recognizer was previously trained by one of the authors using ten training examples for each symbol. Using such a small amount of training data, none of which was provided by the study participants, provides a challenging test of the recognizer's performance. We found that the rare misrecognitions were due to deficiencies in the segmentation process caused by participants drawing too quickly or too small.

Our mass recognizer worked correctly for 11 of the 13 participants. One participant sometimes drew a mass and spring together in a single pen stroke. Another drew small triangles for the arrowheads on the forces, which were then misrecognized as masses. We believe that this situation can be fixed relatively easily by filtering out masses that are significantly smaller than the other masses.

Our ground recognizer worked correctly for 9 of the 13 participants. One participant drew only three hatch strokes, while our program requires a minimum of four. A second participant drew ground symbols with three sets of hatches, each containing three

[3]To ensure consistency across participants, they were asked to sketch vibratory systems that were presented in a schematic drawing. While this study design does provide a meaningful evaluation of our software, it would also be useful to conduct additional studies in which subjects were asked to sketch devices of their choosing.

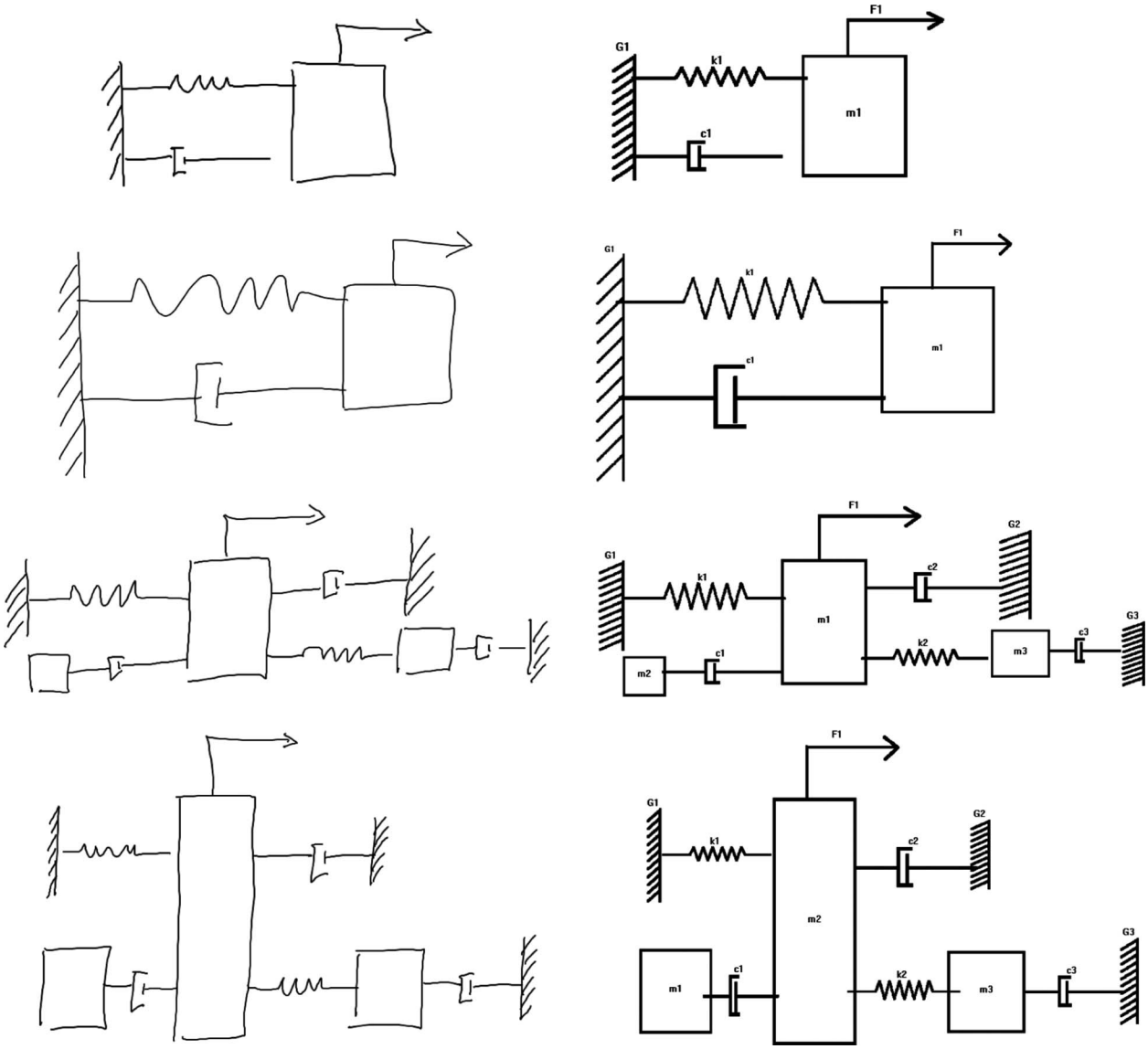

Fig. 9 Examples of sketches from the user study and their interpretations

strokes that were drawn far apart from each other. A third participant varied the directions of the hatch strokes, for example, with one pointing to the southwest, another pointing to the northeast, and so on. These three situations could be handled by a more general definition of a ground symbol. In particular, we expect that there are only a few common styles of hatches. We expect that a special-purpose recognizer could be built for each such style. A fourth participant sometimes used a single stroke to draw both a spring and a ground and rarely lifted the pen while drawing the hatches. These findings suggest that to accommodate a wider variety of users, it may be necessary to adjust some of the geometric criteria used in our special-purpose mass and ground recognizers.

Occasionally, the participants would try to improve the appearance of their sketch after it was nearly completed. For example, they might add a small bit of ink to try to close the boundary of a mass or they might try to extend a ground symbol by adding a few extra hatches. Our mass and ground recognizers require that strokes be drawn consecutively. Thus when new ink is added in this way, it is identified as a separate symbol. We are working to solve this problem by relaxing the requirement for temporal proximity when recognizing mass and ground symbols. Note that such added ink typically does not pose problems in the recognition of springs, dampers, and forces, as our clustering approach is not sensitive to the temporal order, and moreover our feature-based recognizer is robust to a few extra or missing strokes.

Our program did work as expected for the majority of the study participants. This is quite encouraging given that they had no experience with our system, and no information about how it worked, prior to the study. As described above, we are working to resolve the problems that some participants encountered. However, providing users with even minimal information about how the system works would also prevent errors and would still provide a natural drawing environment. To help reveal the complexity of sketches our system can undertake, Fig. 10 shows a sketch consisting of 92 strokes. The sketch is accurately recognized except for one spring symbol.

Our results suggest that the mark-group-recognize technique works well for the domain of vibratory mechanical systems. We believe that this technique also has direct application to other domains. For example, we have used it to interpret networklike diagrams consisting of symbols linked together with arrows [13]. We are continuing to explore ways to further generalize the mark-group-recognize technique. For example, in the work presented here, temporal proximity is used to locate the marker symbols and spatial proximity is used to locate the other symbols. It may be

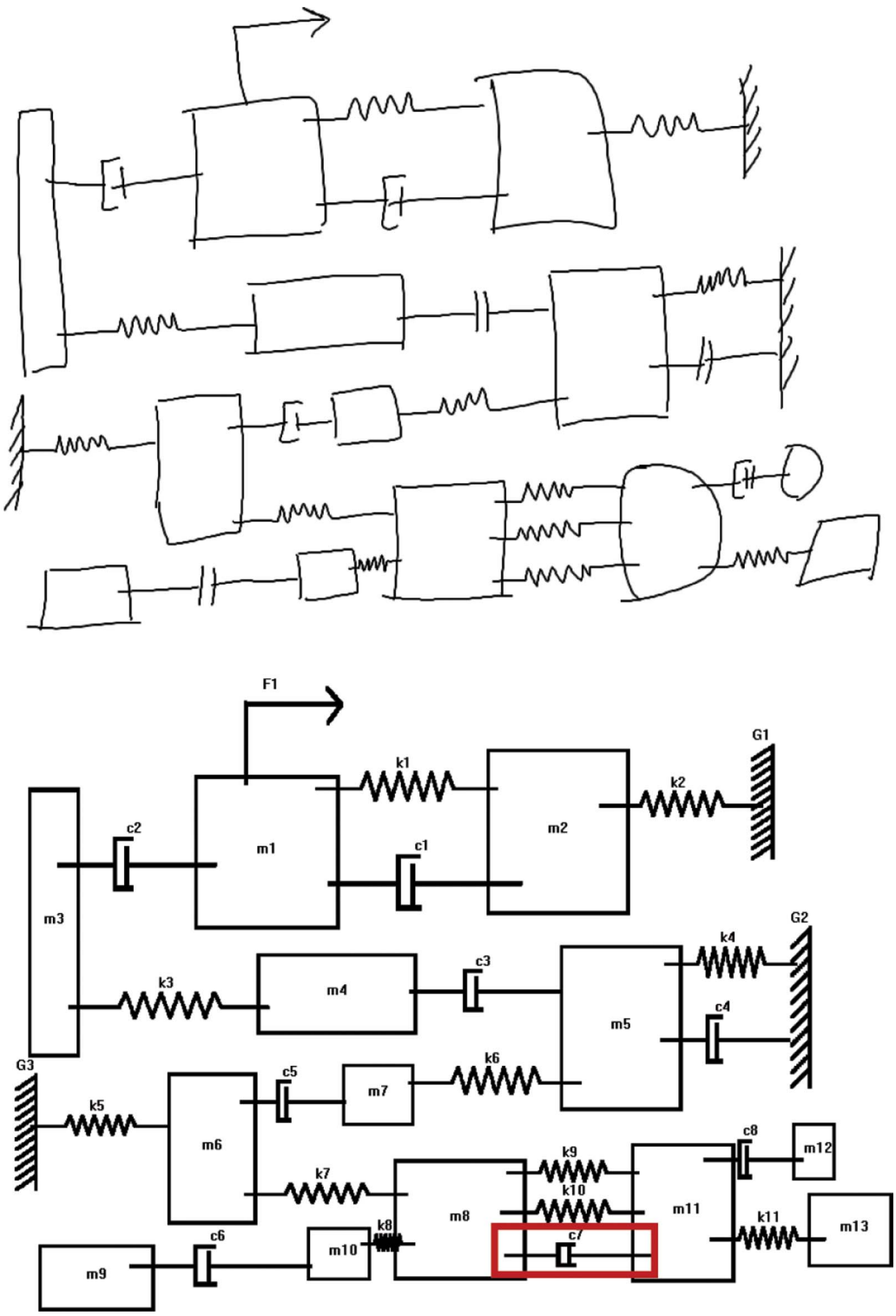

Fig. 10 An example sketch attempted by our system. The sketch is accurately recognized except for one spring symbol (enclosed in box at the bottom).

possible to create a more general technique in which spatial and temporal information are combined to locate individual symbols.

12 Summary and Conclusions

We are working to develop sketch understanding techniques that will enable engineering software to work from the kinds of sketches engineers ordinarily draw when designing, communicating, and problem solving. There are many challenges to be faced in creating such techniques. This work focuses specifically on sketch parsing, the task of automatically separating a stream of pen strokes into distinct symbols. To this end, we have presented a novel mark-group-recognize approach to sketch parsing. Our parser helps to provide a natural drawing environment by allowing the user to draw continuously, without needing to indicate when one symbol ends and the next one begins. In the first step of parsing, easy-to-recognize "marker symbols" are extracted from

the input stream, thus helping to separate the remaining symbols. A clustering algorithm is then used to group the remaining pen strokes into distinct clusters, each representing a unique symbol. Finally, the clusters are recognized with a trainable multistroke symbol recognizer. This recognizer uses a feature-based representation and relies on a statistical model, thus making it robust to drawing variations, such as variations in drawing order and shape.

We have used these techniques to build a sketch-based tool for designing and analyzing vibratory mechanical systems. This tool allows users to construct a computational model of a vibratory system by simply sketching the system on an LCD tablet. Our tool interprets the sketch, analyzes the vibratory behavior, and produces an animation of that behavior by directly animating the user's sketch. Our user studies have shown that our parsing and recognition algorithms work well even for novice users. While our software is not a comprehensive vibration analysis tool, it is one step toward our vision of enabling natural sketch-based interfaces to support engineering design and analysis.

References

[1] Shpitalni, M., and Lipson, H., 1995, "Classification of Sketch Strokes and Corner Detection Using Conic Sections and Adaptive Clustering," ASME J. Mech. Des., **119**, pp. 131–135.

[2] Ullman, D. G., Wood, S., and Craig, D., 1990, "The Importance of Drawing in the Mechanical Design Process," Comput. Graph., **14**(2), pp. 263–274.

[3] Schutze, M., Sachse, P., and Romer, A., 2003, "Support Value of Sketching in the Design Process," Res. Eng. Des., **14**, pp. 89–97.

[4] Yang, M. C., 2003, "Concept Generation and Sketching: Correlations With Design Outcome," ASME Design Engineering Technical Conferences and Design Theory and Methodology Conference, Chicago, IL, September 2–6.

[5] Song, S., and Agogino, A. M., 2004, "Insights on Designers' Sketching Activities in New Product Design Teams," ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Salt Lake City, UT, September 28–October 2.

[6] Yang, M. C., and Cham, J. G., 2007, "An Analysis of Sketching Skill and its Role in Early Stage Engineering Design," ASME J. Mech. Des., **129**(5), pp. 476–482.

[7] Chusilp, P., and Jin, Y., 2006, "Impact of Mental Iteration on Concept Generation," ASME J. Mech. Des., **128**(1), pp. 14–25.

[8] de Silva, R., Bischel, D. T., Lee, W., Peterson, E. J., Calfee, R. C., and Stahovich, T. F., 2007, "Kirchhoff's Pen: A Pen-Based Circuit Analysis Tutor," Eurographics Workshop on Sketch-Based Interfaces and Modeling, June 11–13, Annecy, France.

[9] Fonseca, M. J., Pimentel, C., and Jorge, J. A., 2002, "Cali-an Online Scribble Recognizer for Calligraphic Interfaces," AAAI Spring Symposium on Sketch Understanding, March 25–27, Palo Alto, CA, pp. 51–58.

[10] Narayanaswamy, S., 1996, "Pen and Speech Recognition in the User Interface for Mobile Multimedia Terminals," Ph.D. thesis, University of California at Berkeley, Berkeley, CA.

[11] Landay, J. A., and Myers, B. A., 2001, "Sketching Interfaces: Toward More Human Interface Design," IEEE Computer, **34**(3), pp. 56–64.

[12] Gennari, L., Kara, L. B., and Stahovich, T. F., 2004, "Combining Geometry and Domain Knowledge to Interpret Hand-Drawn Diagrams," AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural, October 21–24, Washington, D.C.

[13] Kara, L. B., and Stahovich, T. F., 2004, "Hierarchical Parsing and Recognition of Hand-Sketched Diagrams," User Interface Software Technology (UIST), October 24–27, Santa Fe, NM.

[14] Saund, E., Mahoney, J., Fleet, D., Larner, D., and Lank, E., 2002, "Perceptual Organisation as a Foundation for Intelligent Sketch Editing," AAAI Spring Symposium on Sketch Understanding, March 25–27, Palo Alto, CA, pp. 118–125.

[15] Jacobs, D. W., 1988, "The Use of Grouping in Visual Object Recognition," MIT AI Lab, Technical Report No. 1023.

[16] LaViola J., and Zeleznik, R., 2004, "Mathpad2: A System for the Creation and Exploration of Mathematical Sketches," ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004), August 8–12, Los Angeles, CA, Vol. 23, pp. 432–440.

[17] Notowidigdo, M., and Miller, R. C., 2004, "Off-Line Sketch Interpretation," AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural, October 21–24, Washington, D.C.

[18] Costagliola, G., and Deufemia, V., 2003, "Visual Language Editors Based on lr Parsing Techniques," Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT'03), Nancy, France.

[19] Sezgin, T. M., and Davis, R., 2005, Hmm-Based Efficient Sketch Recognition, International Conference on Intelligent User Interfaces (IUI'05), New York, January 9–12.

[20] Yasuda, H., Takahashi, K., and Matsumoto, T., 2000, "A Discrete Hmm for Online Handwriting Recognition," Int. J. Pattern Recognit. Artif. Intell., **14**(5), pp. 675–688.

[21] Grimson W. E. L., 1991, "The Combinatorics of Heuristic Search Termination for Object Recognition in Cluttered Environments," IEEE Trans. Pattern Anal. Mach. Intell., **13**(9), pp. 920–935.

[22] Shilman, M., Pasula, H., Russell, S., and Newton, R. 2002, "Statistical Visual Language Models for Ink Parsing," AAAI Spring Symposium on Sketch Understanding, March 25–27, Palo Alto, CA, pp. 126–132.

[23] Rubine, D., 1991, "Specifying Gestures by Example," Comput. Graph., **25**, pp. 329–337.

[24] Shilman, M., and Viola, P., 2004, "Spatial Recognition and Grouping of Text and Graphics," EU-ROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling, August 30–31, Grenoble, France.

[25] Alvarado, C., and Davis, R., 2005, "Dynamically Constructed Bayes Nets for Multi-Domain Sketch Understanding," International Joint Conference on Artificial Intelligence, July 30–August 5, Edinburgh, Scotland, UK.

[26] Kimura, T. D., Apte, A., and Sengupta, S., 1994, "A Graphic Diagram Editor for Pen Computers," *Software Concepts and Tools*, pp. 82–95.

[27] Ozer, O. F., Ozun, O., Tuzel, C. O., Atalay, V., and Cetin, A. E., 2001, "Vision-Based Single-Stroke Character Recognition for Wearable Computing," IEEE Intell. Syst., **16**(3), pp. 33–37.

[28] Apte, A., Vo, V., and Kimura, T. D., 1993, "Recognizing Multistroke Geometric Shapes: An Experimental Evaluation," Proceedings of the UIST 93, November 3–5, Atlanta, GA, pp. 121–128.

[29] Fonseca, M. J., and Jorge, J. A., 2000, "Using Fuzzy Logic to Recognize Geometric Shapes Interactively," Proceedings of the Ninth International Conference on Fuzzy Systems (FUZZ-IEEE 2000), San Antonio, TX, May 2000.

[30] Matsakis, N. E., 1999, "Recognition of Handwritten Mathematical Expressions," MS thesis, MIT, Cambridge.

[31] Hse, H., and Newton, A. R., 2004, "Sketched Symbol Recognition Using Zernike Moments," *17th International Conference on Pattern Recognition*, Cambridge, UK, Vol. 1, pp. 367–370.

[32] Kara, L. B., and Stahovich, T. F., 2004, "An Image-Based Trainable Symbol Recognizer for Sketch-Based Interfaces," AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural, October 21–24, Washington, D.C.

[33] Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Raj Reddy, D., 1980, "The Hearsay-II Speech Understanding System: Integrating Knowldge to Resolve Uncertainty," ACM Comput. Surv., **12**(2), pp. 213–253.

[34] Duda, R. O., Hart, P. E., and Stork, D. G., 2001, *Pattern Classification*, 2nd ed., Wiley, New York.

[35] Kara, L. B., 2004, "Automatic Parsing and Recognition of Hand-Drawn Sketches for Pen-Based Computer Interfaces," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

[36] Stahovich, T. F., 2004, "Segmentation of Pen Strokes Using Pen Speed," AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural, October 21–24, Washington, D.C.

[37] Domingos, P., and Pazzani, M. J., 1997, "Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier," Mach. Learn., **29**, pp. 103–130.