Proceedings of DETC'04
ASME 2004 Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
September 28-October 2, 2004, Salt Lake City, Utah, USA

DETC2004-57529

A SKETCH-BASED INTERFACE FOR THE DESIGN AND ANALYSIS OF SIMPLE VIBRATORY MECHANICAL SYSTEMS

Levent Burak Kara
Mechanical Engineering Dept.
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: lkara@andrew.cmu.edu

Leslie Gennari
Mechanical Engineering Dept.
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: lgennari@andrew.cmu.edu

Thomas F. Stahovich
Mechanical Engineering Dept.
University of California, Riverside
Riverside, California 92521
Email: stahov@engr.ucr.edu

ABSTRACT

We describe a sketch-based interface designed to provide engineers with a computer environment similar to pen and paper. With our interface, users can construct functional engineering models simply by drawing sketches on a computer screen. Unlike paper sketches, however, our interface allows users to interact with their sketches in real time to modify existing objects and add new ones. To demonstrate the utility of our system, we have developed a sketch-based interface for designing and analyzing simple vibratory mechanical systems. The technical contributions of our work include: (1) a sketch parsing method for automatically locating the distinct graphical symbols in a sketch, (2) a general-purpose, trainable symbol recognizer, and (3) special purpose prerecognizers that consider shape information and make use of drawing conventions.

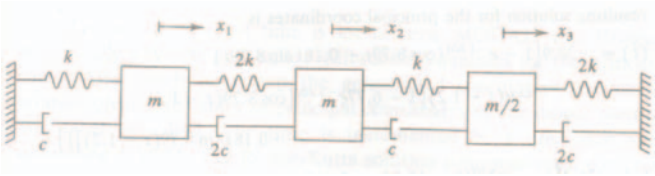
1 Introduction

Our work aims to create natural user interfaces that allow people to operate software using the same sorts of sketches that they would ordinarily use for problem solving and communicating with others. In many disciplines, sketches have great utility as a problem solving tool, as they provide a suitable medium for recording elusive thoughts, visualizing and testing emerging ideas, and for compactly and efficiently representing various types of information such as spatial, temporal and causal relationships. In the realm of engineering and architecture, sketches greatly facilitate conceptual design activities by freeing the designer from worrying about intricate details such as precise size, shape, location and color, and instead enabling him or her to focus on more critical issues that require creativity and abstraction [9]. Due to their minimalist nature, *i.e.*, articulating only what is necessary, they enhance collaboration and communication efficiency.

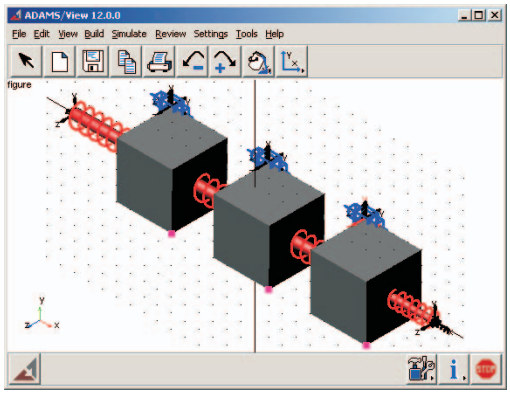
Despite the practical advantages of sketches, and the availability of pen-based hardware such as TabletPC's, electronic whiteboards and PDAs, most contemporary engineering tools cannot work from sketch input. We are working to change this by developing techniques that enable computer software to understand hand-drawn sketches. We are using these techniques to create easy-to-use, sketch-based software for engineering design and analysis. Our current focus is on the development of sketch-based software for engineering education. Students typically use only a subset of the capabilities of commercial engineering software, thus, sketch-based educational software is a readily achievable goal. Such software can be directly integrated into the classroom environment to better illustrate concepts that would ordinarily require "mental simulations." For instance, with a suitable sketch-based simulation tool, an instructor could sketch out a mechanical device on an electronic whiteboard, just as he or she normally would on an ordinary blackboard, and directly animate its behavior. A recent survey of the sophomore mechanical engineering students at Carnegie Mellon University revealed that using a commercial CAD system to visualize mechanical motions greatly helped in their understanding of the key concepts. Our goal is to make such tools readily available in the classroom, while the students are learning a new concept for the first time.

As an example, consider the schematic of a vibratory sys-

(a)



(b)



(c)

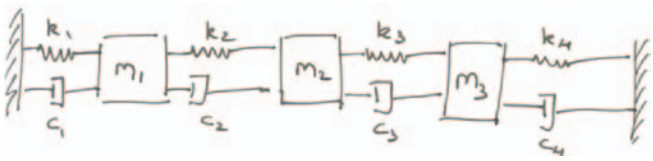


Figure 1. (a) Schematic of a vibratory system taken from [7]. (b) A computer model for the same system created using ADAMS, a commercial dynamic simulation package. (c) A hand-sketch of the same system drawn by a mechanical engineer.

tem shown in Figure 1a. While such diagrams serve as handy visualization tools in human-to-human communication, current computational tools are not designed to work from such representations. Instead, even a simple analysis of such a system typically requires the use of precise computer models such as the one shown in Figure 1b. The use of such software, however, requires a significant amount of training and experience. Part c of the figure, on the other hand, shows a sketch of the same system drawn by a mechanical engineer. From a user's perspective, this type of sketch embodies essentially the same information as in the computer model, yet requires only a fraction of the effort to create it. In this work, we are building techniques that allow dynamic simulators to work directly from such informal sketches.

2 Overview

To provide a test bed for our sketch understanding work, we have developed a sketch-based user interface for analyzing vibratory mechanical systems. We have designed our system so that the user can draw as he or she would on paper, with minimal constraints imposed by our sketch understanding engine. Unlike paper sketches, however, the sketches created with our system are "live." For example, users can interact with their sketches to edit various model parameters, or observe the physical response through live animations of their sketches.

In this work we address two principle challenges in sketch understanding. The first concerns *symbol recognition*, the task of recognizing individual hand-drawn figures 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. We have developed a multi-stroke, domain-independent, trainable symbol recognizer that can learn new symbols from a few prototype examples. Additionally, we have developed two special-purpose recognizers that use knowledge of the drawing conventions of this domain to recognize certain special symbols with high reliability. These special-purpose recognizers are used early in the processing of a sketch to help guide the remaining analysis.

The second issue we address is *ink parsing*, the task of automatically separating a stream of pen strokes into distinct symbols. Without parsing, one would need to dictate each symbol to the system one at a time, for example, by pausing between symbols or pressing a button, which would be a hindrance to the user. To facilitate parsing, we have developed a spatial clustering method that takes as input a cloud of unprocessed pen strokes, and groups them into distinct clusters, each corresponding to a mechanical object.

In the following sections we first describe the user interface of our system together, with its various capabilities. We then present the details behind our sketch understanding approach, and elaborate on our contributions outlined above. Next, we explain how we transform the interpreted sketch into a functional mathematical model. Finally, we present the results from our user studies and conclude with a summary and discussion of our work.

3 User Interaction

We have deployed our software on a 9 in x 12 in Wacom Cintiq digitizing tablet with a cordless stylus (Figure 2). This tablet is also an 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.

Figure 3 shows a snapshot of the user interface. Users can build 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. The user does not need to indicate when one symbol ends and another one begins (i.e, there is no need to pause, press a button, etc.). New components can be added to the sketch at any time.

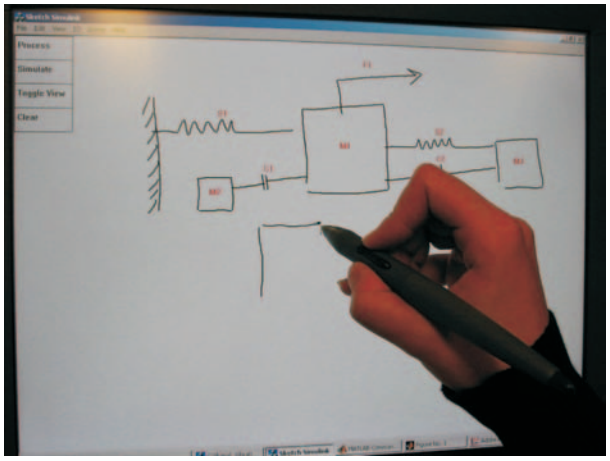


Figure 2. Our sketch-based interface is deployed on a Wacom Cintiq tablet with cordless stylus.

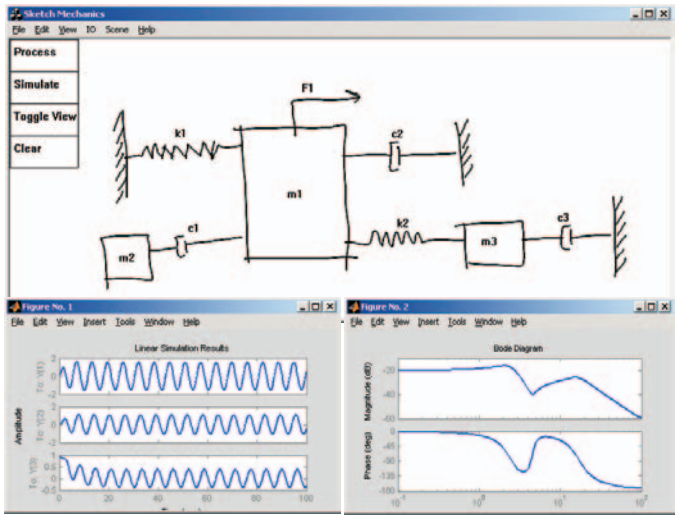


Figure 3. A typical vibratory system created with our system. The program interprets the sketch, performs a simulation of it, and displays the results in the form of live animations and graphical plots.

After the drawing is completed, the user instructs the program to interpret the scene by tapping the "Process" button located at the top left corner of the drawing surface. At this point, the program processes the collection of strokes and identifies the mechanical components present in the sketch. The program demonstrates its understanding by displaying unique text labels next to the identified components. These labels signify the type of the components, and the order in which they were drawn. 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 one 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 Ns/m. External forces are in the form of $F_o \cdot \cos(\omega \cdot t)$ with $F_o = 1$N and $\omega = 1$rad/s. Masses, are assigned mass values proportional to the size they were drawn. The geometrically largest mass block assigned a mass of 1kg, while the remaining ones receive proportionally smaller values. For example, a mass block half the size of the largest one is assigned a mass of 0.5kg.

After identifying the components, our program performs a spatial analysis to determine how the components are connected to one another. It then constructs the set of differential equations that describe the dynamic behavior of the system. 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 understanding techniques, rather it avoids issues related to computing simulations, which are not the focus of this work. The equations are passed to, and solved by, the Matlab engine running in the background. Our program is responsible for initiating Matlab and linking it to our sketch interface. Our program retrieves the solution from Matlab, thus allowing the user to study the system behavior directly from our sketch interface. For example, the user can run a live animation by tapping the "Simulate" button in the interface. When the user does this, the sketch itself is animated: the masses move, the springs compress and stretch, and so on. The simulation results are also displayed in the form of graphical plots. As shown in Figure 3, the graphical output consists of position vs. time plots, and the frequency response of the system.

The objects interpreted by our system are live from the moment they are recognized, thus enabling the user to interact with them. For instance, the user can change the default parameters of an object by pointing to it with the stylus and clicking a button on the side of the stylus. Because the system has recognized the objects, doing this brings up the appropriate dialog box for editing the object's properties. Figure 4 shows an example. Here, the user has clicked on a mass object which brought up a dialog box specialized to masses. This dialog box contains fields for editing the mass value, the initial position and the initial velocity (which 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 program can recognize negative and/or decimal numbers. After updating the properties, the user taps "Process" to implement the changes. The new values are then recognized and displayed in computer fonts. The user closes the dialog box by tapping the "Done" button. Similar dialog boxes exist for the other kinds of components. Changes made to the properties an object are automatically transferred to Matlab and a new simulation is per-

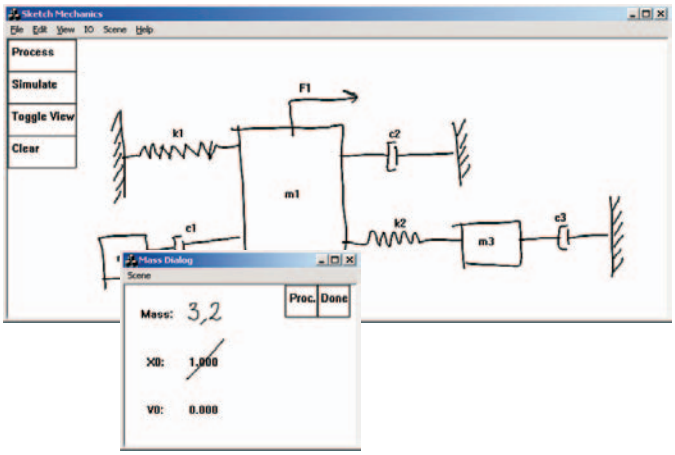


Figure 4. The user can interact with the system through sketch-based dialog boxes. In the instance shown, the user is editing a mass block that contains fields for controlling the magnitude, the initial position and the initial velocity of the mass.

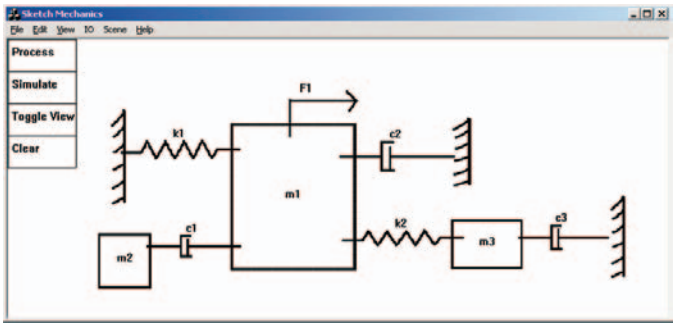


Figure 5. The user can view a beautified version of his model in which the original sketch is replaced by cleaned-up objects.

formed.

The user has the option of viewing the model in its original "sketchy" form or in a "cleaned up" iconic form. Figure 5 shows the cleaned up version of the sketch from Figure 3. Notice that the iconic forms preserve the size of the original shapes. The user can toggle between these two views by tapping the "Toggle View" button in the interface. We believe that the informality of the sketchy view gives a sense of freedom and creativity to the user. The cleaned up view, on the other hand, may give a sense of completeness and definiteness.

As new components are added to the sketch, the user may run out of drawing space. In such cases, the user can request more space by drawing a long line along the right border of the drawing surface. This gesture brings up a *sketchy* scrollbar that the user can instantly use to scroll down the page. The user can then continue drawing in the newly created white space (Figure 6). The scrollbar behaves just as traditional scrollbars do in that as the user drags the scroll thumb downward, the contents of the drawing surface move upward. (This feature is a favorite among those who have seen our system in use.)

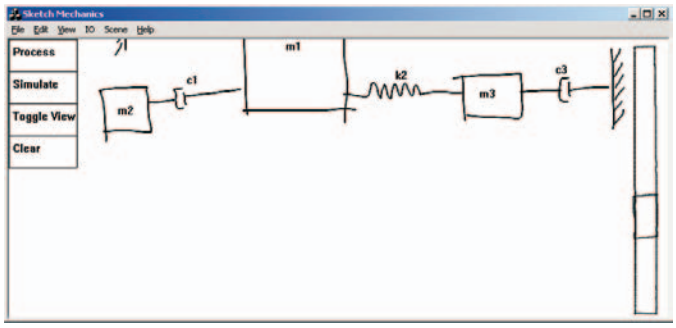


Figure 6. The user can create a scrollbar and instantly use it by drawing a line along the right border of the display.

4 Behind the Scenes

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.

This work addresses two key issues related to sketch understanding. The first concerns the *recognition* of the individual objects placed on the drawing surface. We refer to these objects as "symbols" and the corresponding recognition task as "symbol recognition" (or symbol classification). The springs, masses and dampers in the sketches discussed above are examples of symbols. The numerical digits in the dialog boxes are also symbols. The ability to distinguish between such symbols is the focus of symbol recognition.

The second issue has to do with *ink parsing*, which refers to the task of grouping a user's pen strokes into clusters of intended symbols without requiring the user to indicate when one symbol ends and the next one begins. However, 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 alleviate this difficulty, many of the current systems require the user to explicitly indicate the intended partitioning of the ink. This is often done by pressing a button on the stylus or by pausing between symbols [5, 11]. Alternatively some systems require each object to be drawn in a single pen stroke [9]. However, such constraints

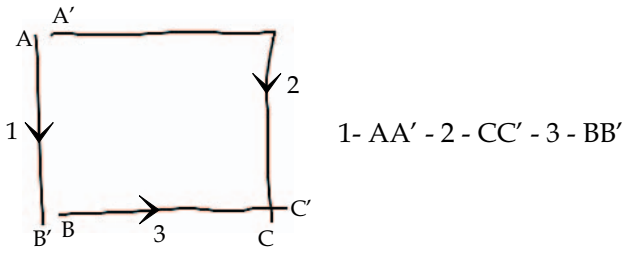


Figure 7. A stroke chain involves the original pen strokes and the hypothetical linkages between them. The stroke-linkage sequence on the right shows the resulting stroke chain. The numbers and arrows indicate the order and directions in which the strokes were drawn. The stroke chain does not assume a particular drawing order or direction.

usually result in a less than natural drawing environment.

Our approach is based on a hierarchical mark-group-recognize architecture. The first step involves a preliminary recognition procedure that examines the stream of pen strokes to identify "markers," symbols that are easily and reliably extracted from a continuous stream of input. Once the marker symbols are identified, the remaining strokes are partitioned into distinct clusters each representing a single symbol. Next, the identified stroke clusters are recognized using a symbol recognizer to determine which components they represent. In last step of our analysis we determine how the recognized components are connected to one another and construct the mathematical equations describing the system behavior. The following sections describe each of these steps in detail.

4.1 Preliminary Recognition

In the domain of mechanical systems, we have found mass and ground symbols to be good marker symbols as they possess a number of unique geometric characteristics that facilitate their recognition. For example masses invariably consist of closed loops. Similarly, 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 a mass object involves finding a set of consecutively drawn strokes that connect end to end forming a closed loop. To determine if a set of strokes forms a closed loop, our program constructs a fully connected *stroke chain* that consists of the original strokes and a set of hypothetical linkages between them. The linkages are formed by joining the strokes to one another based on the minimum endpoint distance. For example in Figure 7, the beginning point of stroke-1 is connected to the beginning point of stroke-2 (with the hypothetical linkage AA') because these two ends are closer to each other than any other pair involving them. In a perfect closed loop, all strokes would be connected precisely at their endpoints and

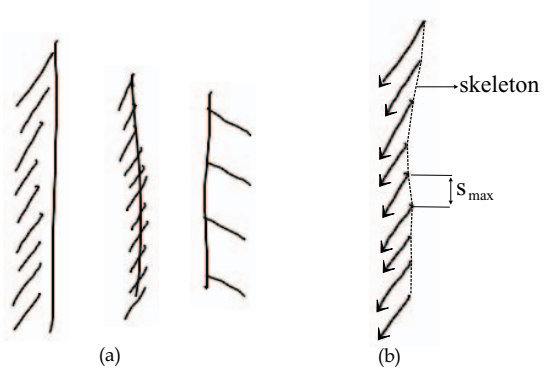


Figure 8. (a) Examples of correctly recognized ground symbols. (b) For recognition, our program considers various features such as the length of the skeleton, the separation between hatches and the orientation of hatches.

therefore the total linkage length would be zero. 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. For strokes that do not form a closed loop, this ratio is typically much higher. For example, it is 100% for a straight line, and can even be even higher than this for arbitrary stroke sets.

Using this algorithm, our program identifies closed loops composed of up to five consecutively drawn strokes[1] including single-stroke loops. Note that this method allows the constituent strokes to be drawn in any arbitrary order and direction. Also, the patterns identified in this way need not form a particular geometric shape such as a square or rectangle, but can be of any arbitrary shape. After identifying the closed loops, our program instantiates the mass objects 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. The distinguishing characteristic of a ground symbol is a set of short, parallel line segments (*i.e.*, the hatches) that are aligned approximately along a straight line (Figure 8). Moreover, these segments are almost always drawn consecutively. Our program thus searches for such patterns in the raw strokes to locate the ground symbols. To prevent arbitrary parallel strokes from being recognized as grounds, our program requires a minimum of four strokes in the hatch area before a ground symbol can be conjectured. To test whether a group of strokes constitutes a ground symbol, our program determines if (1) they are roughly uniformly separated, (2) they are more or less parallel, and (3)

[1] We have found that in the domain of vibratory systems, people usually draw masses in two or three strokes. Hence, the upper limit of five strokes has been sufficient for our purposes.

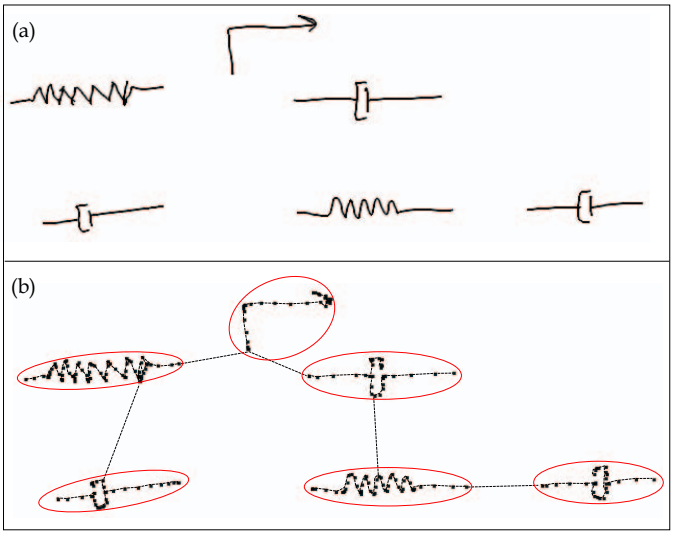


Figure 9. (a) The remaining objects that need to be identified once the masses and the grounds in Figure 3 have been recognized. (b) The hierarchical clustering algorithm separates the scene into distinct clusters. In the configuration shown, the algorithm has been run until a single cluster was obtained. The marked clusters are later determined by analyzing the distance between the merged clusters at every iteration.

the line formed by connecting their starting points (which we call the skeleton) is close to 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 the first points of the strokes to their last points all point to the same quadrant, for example south-west in Figure 8b. The last requirement is satisfied if the skeleton length is within 5% of that of the line extending from the first to the last stroke. Once a core sequence of four strokes that satisfy the above requirements is found, our program determines the extent of the pattern by appending the subsequent strokes one at a time until the pattern is disrupted. Finally, the long stroke that appears next to the hatches is found and added to the pattern. The same procedure is applied to find other ground symbols.

4.2 Clustering

The previous step identifies the masses and grounds but leaves the rest of the sketch uninterpreted. When the masses and grounds are removed from the sketch, one is left with the springs, dampers and forces. For example, Figure 9a shows what is left after the masses and grounds are removed from Figure 3. We split the task of identifying these components into two sub problems. The first is *stroke clustering* in which the strokes are grouped into clusters corresponding to distinct objects. Once the stroke clusters are identified, the next step is to *recognize* each stroke group using the symbol recognizer described in Section 4.3. This section concerns the first of these tasks.

The clustering problem can be formally defined as finding the best grouping of the strokes such that each group embodies all the strokes belonging to a single object while excluding those coming from other objects. For example in Figure 9 this means identifying the six clusters corresponding to the two springs, three dampers and the force. There are four key issues that complicate the problem. The first is that the clusters can occur arbitrarily close to or far from one another. Hence we cannot set a fixed threshold distance below which two strokes would be considered in the same cluster. Second, the clusters can have arbitrary sizes and shapes. Third, each cluster may contain an arbitrary number of strokes. Fourth, and most importantly, one does not know a-priori the number of clusters to be determined.

Our clustering approach relies on the observation that the clusters in our domain typically occur at spatially distinct regions without overlapping. In fact, the purpose of excluding masses and grounds through a preliminary recognition process is to accentuate the separation between clusters. Also, although the distance between two clusters is arbitrary, it is usually greater than the distance between the strokes within the clusters. Hence, different clusters can be identified by grouping together the strokes that reside close to each other and separating those that are not. To implement this idea, we have adopted the agglomerative hierarchical clustering algorithm described in [4].

The clustering procedure is facilitated if the scene is viewed as a collection of data points rather than pen strokes. In this representation, each data point initially forms a distinct seed cluster. The algorithm takes as input these seed clusters and recursively merges them until a single, all-encompassing cluster is obtained. At each step, the two nearest clusters are merged resulting in a bigger cluster that contains the combined set of data points. At each iteration, the number of clusters thus decreases by one.

To find the two nearest clusters at a given step, we must define a distance metric. In our approach, the distance between two clusters A and B is given by:

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

where $\|a-b\|$ represents the Euclidian distance between points a and b. In this formulation $d(A,B)$ corresponds to the distance between the two closest points in A and B, and is known as the *nearest-neighbor* distance. At each step, the program computes this distance for all cluster pairs and merges the two having the minimum of these distances. Although other metrics could be used to determine cluster distances, 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' [4]. Due to their typical appearances, the springs, dampers and forces in our domain often benefit from this effect.

While using the sampled data points as the initial seed clus-

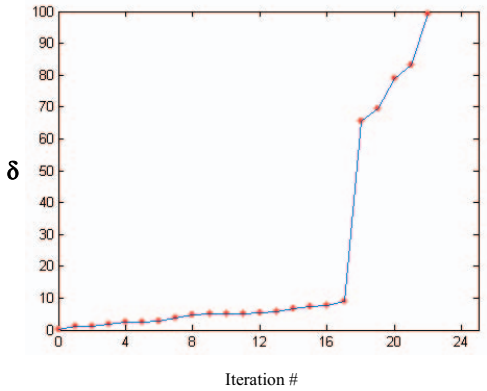


Figure 10. The dissimilarity score δ increases monotonically with the number of iterations. Sharp leaps, such as the one at iteration 17, usually correspond to forced mergers and thus can be used to determine the number of natural clusters.

ters facilitates clustering, it also results in superfluous computations in the early stages of the algorithm. Unless very unusual drawing styles are used, each pen stroke typically belongs to only one symbol. Because of this we initially group all of the data points coming from a single stroke into a single cluster. We have found this to greatly reduce the amount of computation needed to perform clustering.

As mentioned, not knowing the number of symbols to be identified a-priori presents a challenge to our analysis. If this number was known, the clustering algorithm could be terminated when the desired number of clusters was achieved. In our case, however, this number must be determined automatically. The hierarchical clustering algorithm provides a means to accomplish this. At each level of the algorithm, the distance between the clusters merged at that level is stored as a dissimilarity score δ. Because the algorithm merges the nearest clusters at each iteration, δ monotonically increases with the number of iterations. The key, however, is that a large increase in δ usually signals a 'forced merge' [4] - a merge that combines two distant clusters - and thus can be used as a stopping criterion.

We exploit this observation to find the number of clusters. Consider Figure 10 that shows the dissimilarity score versus the iteration number obtained from Figure 9. The large jump from iteration 17 to 18 corresponds to the merging of the force symbol with the damper at its lower right. The subsequent iterations further combine the remaining clusters until a single cluster is obtained. Clearly the intended clusters are those obtained at the end of iteration 17. By finding the sharp leaps in δ, we can thus determine the best stopping iteration. However the challenge is to reliably determine such leaps, which in general may not be as distinct. In our implementation we define the best stopping iteration i^* as the one that maximizes the leap from the preceding iteration to the next, while taking into account the absolute

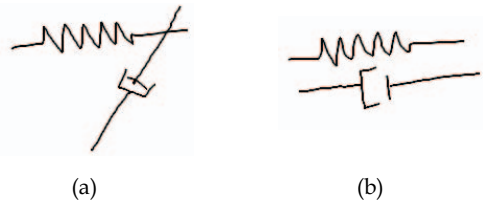


Figure 11. The clustering algorithm falls short when symbols overlap or when intra-symbol distances are comparable to inter-symbol distances.

magnitude of the leap. That is,

$$i^* = \underset{i}{argmax}[\frac{\delta_{i+1} - \delta_i}{\delta_i - \delta_{i-1}} \cdot (\delta_{i+1} - \delta_i)]$$

The first term in the above expression (the ratio) measures the change in the increase of δ between consecutive iterations. This is useful for detecting sharp leaps in δ such as the one that occurs at iteration 17 in Figure 10. However, because the ratio measures only the relative increase, if the increase in δ in the previous iteration was minute, even a small increment in δ on the current 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 in an arbitrary number of strokes, and is not sensitive to the angular orientation of a symbol or the angular orientation of one symbol relative to another. However, it is not well suited when different symbols overlap (Figure 11a), or when an internal gap in a symbol is comparable in size to the distance to a neighboring symbol (Figure 11b). 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 intra-cluster merges to inter-cluster merges will not be distinct as in Figure 10. Although the first of these issues is highly 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 requiring the user to keep the gaps in the dampers to a minimum.

4.3 Symbol Recognition

Once the symbol clusters have been identified, the next step is to actually recognize each cluster. We have developed a trainable symbol recognizer for this purpose. The recognizer takes as input the raw strokes in a cluster and outputs the domain object that best matches the given strokes. Because masses and grounds are already identified in the preliminary recognition step, the

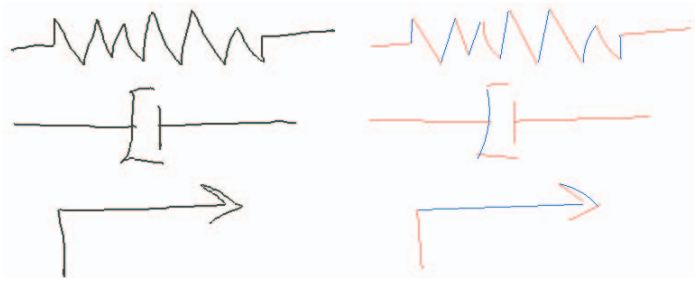


Figure 12. The original (left) and segmented (right) versions of a spring, damper and a force symbol.

symbol recognizer presented here is used for distinguishing between springs, dampers and forces only. The relatively small number of patterns to consider in our working example, however, should not obscure the utility of our symbol recognizer. To date, we have successfully used this recognizer in several other domains with significantly larger symbol libraries. The following steps describe the details of our recognizer.

Segmentation: Our recognizer first decomposes the raw strokes into line and arc segments that closely 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 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 at a minimum, the ink exhibits high curvature, or the sign of the curvature of the ink changes (the details can be found in [2]). 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. Examples of segmented spring and damper symbols are shown in Figure 12.

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 non-uniform scaling.

A set of nine geometric features are extracted from the segmented version of each training example. These 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, the average distance between endpoints of the segments, 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 it is 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 only assume discrete values, the average distance between endpoints is continuously valued.

Once these nine features have been computed for each of the training examples of a symbol, a statistical definition model is constructed for the symbol. 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 range 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. At the end, S is classified by the definition D^* that maximizes the probability of match. That is:

$$D^* = \underset{i}{argmax}\, P(D_i|S)$$

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 tells us that the definition which best classifies the symbol is the one that maximizes the likelihood of observing the symbol's individual features.

$$D^* = \underset{i}{argmax} \prod P(x_j|D_i)$$

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

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

Since we are assuming that the features are 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 [3] show 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.

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 since a small number of training examples are often used, and since eight of the features used for classification can only assume 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 method significantly increases recognition rates, especially when only a few examples have been used for training.

4.4 Connectivity Analysis

The final step in our analysis involves finding how the recognized components are connected to one another so that we can construct the equations of motions. This is accomplished in a straightforward way by connecting the components that are spatially nearest to each other. For example in Figure 3, the right end of spring k1 is connected to mass m1 because among all grounds and masses, m1 is the nearest component to the right end of k1. Our measure of proximity between a mass and a spring is the Euclidian distance between the bounding box center of the mass and the end of the spring[2]. Similar measures are used for determining the connectivity between springs and grounds, dampers and grounds, dampers and masses, and forces and masses. Note that in the models we consider, we require each end of a spring or damper to be connected to precisely one mass or one ground symbol, whichever is closer. 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 structural analysis described above circumvents the pitfalls that can occur due to a literal interpretation of the sketch. Our goal is to infer the *intended* rather than the *apparent* structure. For instance, in Figure 3, although c1 and m1 are not actually attached, our program decides, just as anybody seeing the sketch would, that the two are connected. A literal interpretation, on the other hand, would consider the two components disconnected.

After determining the connectivity between components, we construct the equations of motions. 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 initial velocity vectors. All of these can be straightforwardly written once the connectivity of the components has been determined. Finally, these system matrices and vectors are passed to, and solved by, the Matlab engine running in the background. The solution is a displacement vector, whose elements are the displacements of each of the masses as a function of time. These results are displayed to the user in the form of conventional Matlab plots and as an animation of the user's sketch where masses translate, and dampers and springs stretch and compress (Figure 3).

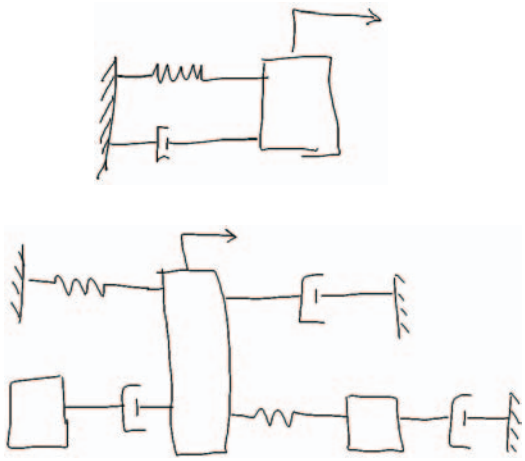


Figure 13. Two successfully recognized sketches employed in our user study.

5 Evaluation and Discussion

We are currently in the process of conducting formal user studies to test and improve our system. As a first step, we chose to evaluate the parsing and recognition accuracy of our system. Usability studies considering such things as the editing and viewing capabilities of the user interface described in Section 3 will be conducted in the future.

We asked 13 subjects, most of whom were graduate and undergraduate mechanical engineering students, to sketch the two types of vibratory systems shown in Figure 13. Each subject provided four sketches, two of each type. Subjects had very little or no experience with the LCD tablet. Moreover, the test was conducted in a walk-up-and-draw fashion in which subjects were nearly immediately asked to start drawing. Only a brief warm-up period of about 30 seconds was given to allow the subject to become familiar with the stylus and LCD tablet. No explanation

[2]The bounding box of a symbol is the smallest sized rectangle, aligned with the coordinate axes, that fully encloses the symbol.

was given about how the program performs its task. For example, subjects were not told that the system begins by looking for closed loops to identify masses, and hatches to identify ground symbols. Each session involved only data collection; the data was processed at a later time. This approach was chosen to prevent the participants from adjusting their style from one sketch to the next based on our program's output.

These initial results indicate that we have a sound parsing and recognition approach. However, to accommodate a wider variety of users, it may be necessary to adjust some of our assumptions about drawing styles. In general, our parsing algorithm worked quite successfully. However when it did fail, it was due to the phenomenon illustrated in Figure 11b, 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, even though none of the participants were involved in the training of the recognizer.[3] We found that the rare misrecognitions were due to deficiencies in the segmentation process caused by subjects drawing too quickly or too small.

Our mass recognizer worked correctly for 11 of the 13 subjects. One subject 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 this situation can be fixed relatively easily by filtering out masses that are geometrically small compared to the rest of the masses.

Our ground recognizer worked correctly for 9 of the 13 subjects. One subject drew only three strokes for the hatch, while our program requires four. A second subject drew ground symbols in which the hatch consisted of three sets of hatches, each containing three strokes, that were drawn far apart from each other. A third subject varied the directions of the strokes in the hatch, for example, with one pointing to the south-west, another pointing to the north-east, and so on. These three situations might be handled by a more general definition of a ground symbol. A fourth subject sometimes used a single stroke to draw both a spring and a ground, and rarely lifted the pen while drawing the hatches.

Occasionally, the test subjects 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 special-purpose mass and ground recognizers require that strokes be drawn consecutively. Thus when new ink is added in this way, it will be identified as a separate symbol. We are currently 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 parsing 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 test subjects. This is quite encouraging given that they had no experience with our system, and no information about how it worked, prior to the test. As described above, we are working to resolve the problems that some test subjects 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.

6 Related Work

Alvarado [1] describes a system that can interpret and simulate a variety of simple hand-drawn mechanical systems. While we share similar end goals as theirs, our approach differs in the way it interprets sketches. Their system uses a number of heuristics to construct a recognition graph containing the likely interpretations of the sketch. The best interpretation is chosen using a scoring scheme that uses both contextual information and user feedback. With their system, each time a new stroke is entered, the entire recognition tree is updated. Our parsing approach is intended to minimize computation by avoiding the need to search the entire sketch with a recognizer. We propose the mark-group-recognize approach where potentially expensive recognition is deferred until the intended stroke groups have been identified. Moreover, our feature-based symbol recognizer has been designed to be trainable, thus allowing for greater adaptability and customization.

Rubine [12] describes a trainable gesture recognizer for direct manipulation interfaces. A gesture is characterized by a set of 11 geometric and 2 dynamic attributes. Based on these attributes, a linear discriminant classifier is constructed whose weights are learned from the set of training examples. Because this method was developed exclusively for gesture-based interfaces, it is only applicable to single-stroke sketches and is sensitive to the drawing direction.

Kurtoglu and Stahovich [8] describe a program that augments sketch-understanding with qualitative physical reasoning to understand schematic sketches of physical devices. Harnessed with the shape recognizer described in [2] the program first identifies the geometric interpretation of an input shape and then uses constraint satisfaction techniques to efficiently construct physically consistent interpretations of the identified components. It then uses qualitative simulation to select the interpretation that produces an intended behavior. One key feature of their system is that it allows users to incorporate shapes from several different domains instead of limiting them to one particular domain.

Landay and Myers [9] present an interactive sketching tool called SILK that allows designers to quickly sketch out a user interface and transform it into a fully operational system. As

[3] The recognizer was previously trained by one of the authors using 10 training samples for each symbol.

the designer sketches, SILK's recognizer (adapted from Rubine's method) matches the pen strokes to symbols representing various user interface components, and returns the most likely interpretation. Their recognizer is limited to single-stroke shapes drawn in certain preferred orientations. Our method handles multi-stroke shapes drawn in any orientation.

Hong and Landay [6] describe a program called SATIN designed to support the creation of pen-based applications. SATIN consists of a set of mechanisms for manipulating, handling, interpreting and viewing strokes; a set of policies to distinguish between the *type* (gesture vs. symbol) of the input stroke;[4] and a number of beautification techniques to organize and clean up sketches. Their system employs Rubine's algorithm as the primary recognition engine and hence is limited to single stroke objects.

Mankoff *et.al.* [10] have explored methods for modeling and resolving ambiguity in recognition based interfaces. Drawn from a survey on existing recognizers, they present a set of ambiguity resolution strategies, called mediation techniques, and demonstrate their ideas in a program called Burlap. Their resolution strategies are concerned with how ambiguity should be presented to the user and how the user should indicate his or her intention to the software. This work highlights a number of critical considerations that demand consideration for a better interaction between the end user and the software.

7 Summary and Conclusions

We are working to develop sketch understanding techniques that will enable software to operate from the kinds of sketches people ordinarily draw when communicating and problem solving. This work addresses two key technical challenges. The first is "parsing," the process of extracting distinct symbols from a continuous stream of pen strokes. For this, we developed a mark-group-recognize scheme. Easily recognizable "marker symbols" are first 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 symbols, which are passed to the recognizer for identification. This 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. The second challenge addressed in this work is "recognition," the process of identifying a graphical symbol composed of a set of pen strokes. For this, we developed a multi-stroke, trainable symbol recognizer that uses statistical, feature-based pattern recognition techniques. One of the strengths of this recognizer is that it can learn new symbol definitions from only a handful of prototype examples, thus allowing the system to be readily customizable to individual users. Also, this recognizer is robust to variations in drawing order and segmentation.

Although this work is at an early stage and there is clearly much more to be done, it suggests that it may in fact be possible to build sketching systems that are "better than paper." For example, our system requires no more effort than is required to make a sketch on paper, yet it provides functionality not offered by paper, such as the ability to directly animate a sketch of a mechanical device.

[4] They rely on buttons located on the mouse or the stylus to distinguish the type of the stroke.

REFERENCES

[1] Christine Alvarado. *A Natural Sketching Environment: Bringing the Computer into Early Stages of Mechanical Design*. Master thesis, Massachusetts Institute of Technology, 2000.

[2] Chris Calhoun, Thomas F Stahovich, Tolga Kurtoglu, and Levent Burak Kara. Recognizing multi-stroke symbols. In *AAAI Spring Symposium on Sketch Understanding*, AAAI Technical Report SS-02-08, pages 15–23, 2002.

[3] Pedro Domingos and Michael J Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. *Machine Learning*, 29:103–130, 1997.

[4] Richard O. Duda, Peter E. Hart, and GDavid G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., 2nd edition edition, 2001.

[5] Manueal J Fonseca, Cesar Pimentel, and Jaoquim A Jorge. Cali-an online scribble recognizer for calligraphic interfaces. In *AAAI Spring Symposium on Sketch Understanding*, AAAI Technical Report SS-02-08, pages 51–58, 2002.

[6] Jason I Hong and James A Landay. Satin: A toolkit for informal ink-based applications. In *ACM UIST 2000 User Interfaces and Software Technology*, pages 63–72, San Diego, CA, 2000.

[7] S. Graham Kelly. *Fundamentals of Mechanical Engineering*. Mc Graw-Hill, 1993.

[8] Tolga Kurtoglu and Thomas F Stahovich. Interpreting schematic sketches using physical reasoning. In *AAAI Spring Symposium on Sketch Understanding*, AAAI Technical Report SS-02-08, pages 78–85, 2002.

[9] James A Landay and Brad A Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3):56–64, 2001.

[10] Jennifer Mankoff, Gregory D. Abowd, and Scott E Hudson. Oops: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. *Computers and Graphics*, 24(6):819–834, 2000.

[11] Shankar Narayanaswamy. *Pen and Speech Recognition in the User Interface for Mobile Multimedia Terminals*. Ph.d. thesis, University of California at Berkeley, 1996.

[12] Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25:329–337, 1991.