Sim-U-Sketch: A Sketch-based interface for Simulink

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

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

ABSTRACT

SIM-U-SKETCH is an experimental sketch-based interface we developed for Matlab®'s Simulink® software package. With this tool, users can construct functional Simulink models simply by drawing sketches on a computer screen. To support iterative design, SIM-U-SKETCH allows users to interact with their sketches in real time to modify existing objects and add new ones. The system is equipped with a domain-independent, trainable symbol recognizer that can learn new symbols from single prototype examples. This makes our system easily extensible and customizable to new domains and unique drawing styles.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User Interfaces*; I.3.6 [**Computer Graphics**]: Methodology and Techniques [Interaction Techniques]

1. INTRODUCTION

Our work aims to create sketch understanding techniques to enable natural, sketch-based user interfaces. While hand-drawn sketches serve an important role as a problem solving tool in many disciplines, current computational tools are not designed to work from such representations. Our goal is to change this by combining the freedom of freeform sketching with the unique affordances of existing computer software.

To provide a test bed for our work, we have developed SIM-U-SKETCH, a sketch-based user interface for Matlab's Simulink software package. Simulink is an add on package for Matlab, and is used for analyzing feedback control systems and other similar dynamic systems. A typical session in Simulink involves a number of processes. Typically, the user navigates through a multi-level symbol palette to find, select and drag the components one at a time onto an empty canvas. Objects are connected to one another with arrows formed by extending directed lines from specific terminals on the source symbols, such as a Sine Wave, to specific terminals on the target symbols, such as a Scope block. Model parameters such as signal magnitudes, transfer function constants and the number of input/output terminals are specified by clicking on the objects and typing in the relevant values using the keyboard. Finally, simulations are performed using the Matlab engine running in the background.

SIM-U-SKETCH, on the other hand, allows the user to accomplish the same task in a more straightforward manner. We have designed our system so that the user can draw as he or she ordinarily would on paper, with minimal constraints imposed by our sketch understanding engine. For instance, rather than having to navigate a symbol palette to locate and choose an object, the user simply draws the symbols and combines them with hand-drawn arrows[1]. During this process, the system does not restrict the order in which the symbols must be drawn, and it does not require the user to indicate when one symbol ends and the next one begins. Also, users need not explicitly specify the number of input/output terminals for each symbol; the program automatically determines this by analyzing the spatial configurations of the arrows.

Once the user's sketch is interpreted by our system, it becomes a functional Simulink model that the user can interact with. For example, users can modify the parameters of various symbols and run a simulation of their models all in real-time. Another unique aspect of our interface is that users can customize the system to their own drawing styles by providing a single prototype example of each object they would like to use. This ability to quickly train the system frees the user from having to memorize the appearance of each Simulink symbol.

2. USER INTERACTION

SIM-U-SKETCH is deployed on a 9"x12" digitizing LCD display with stylus. Figure 1 shows a snapshot of our interface in use. As shown, our program demonstrates its understanding by placing a bounding box around each symbol, and indicating the symbol type with text. Once the sketch has been processed, the user can choose to view a cleaned up version of the sketch in which the symbols are replaced by beautified iconic images, and the arrows are straightened out into line segments.

The objects interpreted by SIM-U-SKETCH are live from the moment they are recognized, thus enabling the user to interact with them. For instance, the user can edit the properties of a Simulink object by clicking a button on the side of the stylus while it is pointing to the object. This brings

[1]Currently our program's library contains 16 Simulink objects.

AVI '04, May 25-28, 2004, Gallipoli (LE), Italy

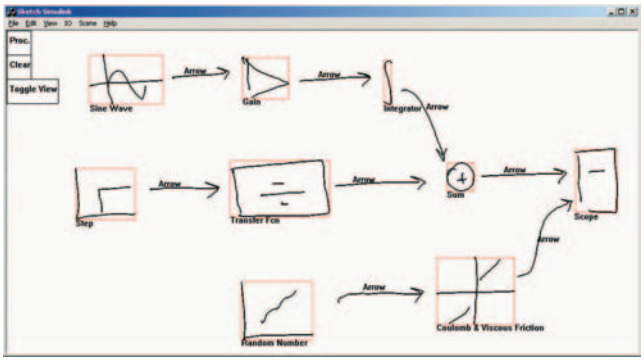

Figure 1: Our system interprets the sketch and displays its understanding by marking each of the Simulink objects and arrows.

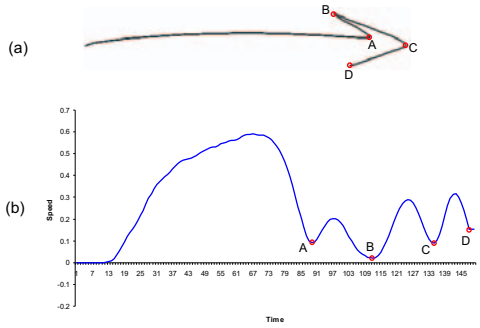

Figure 2: Arrow recognition. (a) A one-stroke arrow with the key points labeled. (b) Speed profile. Key points are speed minima.

up a sketch-based dialog box that allows the user to change numerical values by simply crossing out the old ones and writing in new ones; the new values are recognized using a digit recognizer we developed. After completing the sketch, the user can run a simulation of the system and view the results directly through our sketch-based interface. At any time, the user can sketch new objects into the model, and a new simulation is automatically performed.

3. BEHIND THE SCENES

To create our sketch understanding system we had to address two key technical challenges. The first is *symbol recognition*, the recognition of the individual objects placed on the drawing surface. The ability to distinguish between, say a Sine Wave and a Transfer Function, is the focus of symbol recognition. The second issue has to do with *ink parsing*, the task of grouping a user's pen strokes into clusters representing 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 [4, 7]. Alternatively some systems require each object to be drawn in a single pen stroke [6]. However, such constraints usually result in a less than natural drawing environment.

Our approach is based on a hierarchical mark - group - recognize architecture. The first step is to examine the stream of pen strokes and identify "markers," symbols that are easily and reliably extracted from a continuous stream of input. These markers are then used to efficiently cluster the remaining strokes into distinct groups corresponding to individual symbols. The key here is that stroke clustering is driven exclusively by the marker symbols identified in the first step, without need for search. In the last step, the identified stroke groups are evaluated using a symbol recognizer we developed. Although our working example is Simulink, these techniques are equally applicable to other types of data flow diagrams such as organizational charts and algorithmic flowcharts, and various graphical representations including finite state machines, Markov models and Petri nets.

3.1 Preliminary Recognition

One key to successful sketch understanding lies in the ability to establish the ground truths about the sketch early on, before costly mistakes take place. 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. To be useful, such markers must serve as natural delimiters between the other symbols. In the domain of data flow diagrams such as Simulink, we have found arrows to be this kind of pattern. Our program thus begins by using a prerecognizer to find the arrows in the sketch.

Our user studies have indicated that users typically draw arrows with either a single pen stroke or two consecutive strokes. Either way, they are invariably drawn from tail to head. Thus, our arrow prerecognizer examines the end of each pen stroke to determine if it is an arrowhead. This determination is made on the basis of pen speed information. As shown in Figure 2, the characteristic corner points of an arrowhead typically occur when the pen speed is at a minimum. Once these points are determined from the pen speed profile, a series of geometric tests is performed to determine whether or not the stroke really is an arrow. For example, we require angles $\widehat{ABC}$ and $\widehat{BCD}$ to both be less than 90°, and the lines defined by BC and DC to be short compared to the total length of the pen stroke. Because the approach need not consider the shape of the arrow shaft, it can recognize a wide variety of arrows.

3.2 Stroke Clustering

Once the arrows have been recognized, the next step is to group the remaining strokes into clusters representing individual symbols. In data flow diagrams, each arrow connects a source object at its tail to a target object at its head. Hence, different clusters can be identified by grouping together all of the strokes that are near the end of a given arrow.

Stroke clustering begins by assigning each non-arrow stroke to the nearest arrow end (Figure 3a). Strokes assigned to the same arrow end are grouped to form a cluster (Figure 3b). Clusters with partially or fully overlapping bounding boxes are then merged (Figure 3c). Finally, each arrow end that has no cluster is linked to the nearest stroke cluster (Figure 3d). This helps to ensure the intended connectivity of the diagram by ensuring that each arrow has a cluster at its tail and head.

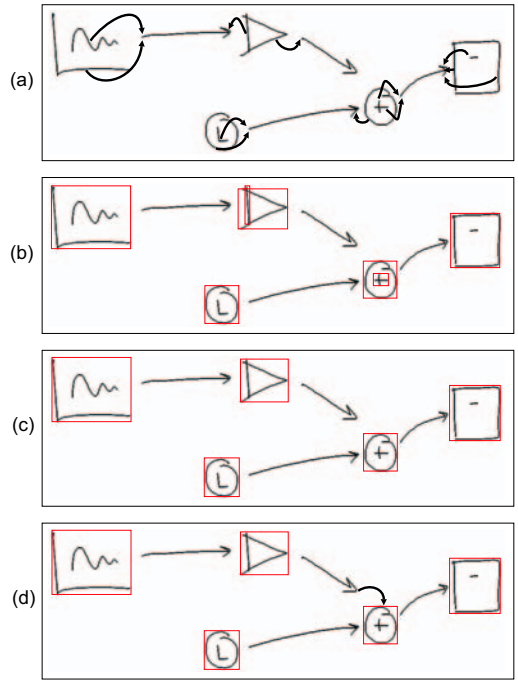

Figure 3: Illustration of the cluster analysis.

3.3 Generating Symbol Candidates

After identifying the stroke clusters, the next step is to recognize the symbols suggested by each of the clusters. Sim-U-Sketch combines contextual knowledge with shape recognition to achieve accuracy and efficiency. One form of contextual knowledge in the Simulink domain comes from the examination of the number of input and output terminals associated with each symbol. Certain objects can have only output terminals and therefore will have only outgoing arrows. For example source signals such as the Sine Wave, Chirp Signal, Random Number Generator and Step Function are of this type. Likewise, some symbols can have only input terminals, such as the Scope block, or may have an arbitrary number of input and output terminals such as the Sum block.

By examining the number of input and output arrows for a given cluster, the program is able to narrow down the set of possible interpretations of a symbol. This reduces the amount of work the recognizer must do. It also helps increase accuracy by reducing the possibilities for confusion. For example, while the Sum block and the Clock look quite similar (the two circular symbols in Figure 3), context dictates that a Sum block must have at least two incoming arrows while the Clock must have none. With this additional knowledge, the recognizer would never consider the Sum block and the Clock as two competing candidates during shape recognition.

3.4 Symbol Recognition

Sim-U-Sketch employs a novel image-based recognizer to find the best interpretations of the stroke clusters. Input symbols are internally described as 24x24 quantized bitmap images which we call "templates". This representation has a number of desirable characteristics. First, segmentation – the process of decomposing the sketch into constituent primitives such as lines and curves – is eliminated entirely. Second, the representation is well suited for recognizing "sketchy" symbols such as those with heavy overtracing, missing or extra segments and different line styles (solid, dashed, *etc.*). Lastly, this recognizer puts no restrictions on the number of strokes used for a symbol, or the order in which the strokes are drawn.

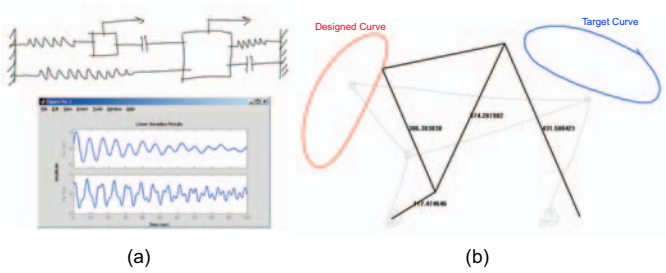

Figure 4: (a) A sketch-based interface for analyzing vibratory mechanical systems.(b) A sketch-based 4-bar linkage design tool.

Our recognizer uses an ensemble of four different classifiers to evaluate the match between an unknown symbol and a candidate definition symbol. These classifiers are extensions of the following methods: (1) Hausdorff distance [9], (2) Modified Hausdorff distance [2], (3) Tanimoto coefficient [3] and (4) Yule coefficient [10]. Each classifier provides a similarity score between two symbols by superimposing their template representations and measuring the match between their black pixels. The classifiers differ in the way they measure this similarity. During recognition, the classifiers evaluate the unknown against each of the symbol definitions, and each outputs a list of definitions ranked according to their similarity to the unknown. Results of the individual classifiers are then synthesized, and the candidate definition with the best combined score is assigned to the symbol.

The recognizer is versatile in that we use it both for graphical symbol recognition and digit recognition. One advantage of this recognizer over traditional ones is that it can learn new definitions from *single* prototype examples. For training, the user creates a new symbol definition by simply presenting an example of it to the system. With this approach, users can seamlessly train new symbols, and remove or overwrite existing ones on the fly, without having to depart the main application.

4. OTHER EXAMPLES

To further explore the use of our techniques, we have developed other sketch-based interfaces for analyzing vibratory mechanical systems (Figure 4a) and 4-bar linkages (Figure 4b). The first tool allows users to sketch out a mechanical system and study its dynamic behavior. Once the user's sketch is interpreted, our program can directly animate the user's hand-drawn sketch. During the animation, the masses move, the springs compress and stretch, *etc.* The simulation results are also displayed in the form of graphical plots including a position vs. time plot for each of the masses in the system. The second tool allows users to sketch out a 4-bar linkage mechanism and see its animation. This visual feedback is useful in that users can immediately determine the type of their mechanism, such as a crank-and-rocker,

double-crank, or double rocker.[2] During the animation, one can also see the trajectory followed by the coupler point (the apex of the triangle in Figure 4b), whose motion is usually the main design objective. Users can also design new mechanisms by simply sketching a target trajectory to be traversed; using an optimization algorithm, our program produces a new mechanism that minimizes the difference between the target curve and the curve produced by the 4-bar linkage the user sketched.

5. RELATED WORK

Recent years have seen the development of experimental sketch-based interfaces for a number of different disciplines including engineering design, user interface design and architecture. Alvarado and Davis [1] describe a system that can interpret and simulate a variety of simple, hand-drawn mechanical systems. The 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. In their approach, each time a new stroke is entered, the entire recognition tree is updated. By contrast, we defer analysis until the user finishes sketching. Also, their shape recognizers are sensitive to the results of segmentation (*i.e.*, fitting line and arc segments to the raw ink), forcing the user to be cautious during sketching. Our approach does not rely on segmentation, thus allowing for more casual drawing styles.

Rubine [8] 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.

Landay and Myers [6] 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 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 [5] 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;[3] 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.

6. CONCLUSION

We described an experimental system called SIM-U-SKETCH designed to enable sketch-based interaction with the Simulink software package. This work addresses two key technical challenges in sketch understanding. The first is the recognition of the objects implied by a user's pen strokes. For this, we developed a domain-independent, multi-stroke, trainable symbol recognizer. One advantage of this recognizer over traditional ones is that it can learn new definitions from single prototype examples. The second challenge was parsing - identifying distinct symbols from a continuous stream of pen strokes. We developed a multi-level parsing scheme that allows users to continuously sketch, without needing to indicate when one symbol ends and a new one begins. The parser uses contextual knowledge to both improve accuracy and reduce recognition times.

We are currently in the process of conducting formal user studies to test and improve our system. Several users tested our interface and all had highly favorable opinions of it. Most users found it easy and straightforward to use, although some had difficulty using the LCD tablet (they feared that resting their hand on the display would break it). The preliminary results have indicated that we have a sound parsing and recognition algorithm but that the arrow recognition could be improved to accommodate a wider variety of users. Nevertheless, we have found the latter to be only a minor issue as even first-time users quickly become adept at using our system.

[2]Although this distinction is central to the design process, it is usually not discernable from the static image of the mechanism.

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

7. REFERENCES

[1] C. Alvarado and R. Davis. Resolving ambiguities to create a natural sketch based interface. In *IJCAI-2001*, 2001.

[2] M.-P. Dubuisson and A. K. Jain. A modified hausdorff distance for object matching. In *12th International Conference on Pattern Recognition*, pages 566–568, Jerusalem, Israel, 1994.

[3] M. Fligner, J. Verducci, J. Bjoraker, and P. Blower. A new association coefficient for molecular dissimilarity. In *The Second Joint Sheffield Conference on Chemoinformatics*, Sheffield, England, 2001.

[4] M. J. Fonseca, C. Pimentel, and J. 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.

[5] J. I. Hong and J. 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.

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

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

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

[9] W. J. Rucklidge. *Efficient Visual Recognition Using the Hausdorff Distance*. Number 1173 Lecture Notes in computer Science,. Springer-Verlag, Berlin, 1996.

[10] J. D. Tubbs. A note on binary template matching. *Pattern Recognition*, 22(4):359–365, 1989.