ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Graphics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Combining geometry and domain knowledge to interpret hand-drawn diagrams

Leslie Gennari[a], Levent Burak Kara[a], Thomas F. Stahovich[b,*], Kenji Shimada[a]

[a]*Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[b]*Department of Mechanical Engineering, University of California, Riverside, CA 92521, USA*

Abstract

One main challenge in building interpreters for hand-drawn sketches is the task of parsing a sketch to locate the individual symbols. Many existing pen-based systems avoid this problem by requiring the user to explicitly indicate the partitioning of the sketch with button clicks or pauses in drawing. We have created a parser that automatically locates symbols by looking for areas of high ink density, and for points at which the characteristics of the pen strokes change. To demonstrate our techniques, we have developed AC-SPARC, a sketch-based interface for the SPICE electric circuit analysis program. An evaluation of our interface has indicated that, even for novice users, our system can successfully locate and identify most of the circuit components in hand-drawn circuit diagrams.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Pen-based computing; Sketch understanding; Sketch parsing; Symbol recognition; Ink density; Electric circuits

1. Introduction

Sketching with pencil and paper has always been an important means of communication and problem-solving for designers and engineers. There are a variety of reasons for this. For example, sketches are a convenient tool for examining geometric, temporal, and other similar relationships, which cannot be described easily in words. Similarly, the simplicity and ease of creating a sketch allows one to focus on problem solving rather than the communication medium. Yet, despite the importance of sketches in engineering practice, traditional engineering software can do little with them. Engineers often find themselves recreating their sketches on the computer in order to take advantage of such software. We are working to change this by creating sketch-understanding techniques that enable software to work directly from the kinds of sketches engineers ordinarily draw.

To be natural, a sketch-based interface must place few constraints on the way a user draws, allowing the same freedom provided by pencil and paper. For example, the user should be able to sketch continuously, without being interrupted by the system, and without having to alter his or her drawing style to fit the constraints of the system. We have developed a technique for automatically parsing sketches as one means of achieving this kind of natural interface. Parsing is the task of grouping a user's pen strokes into clusters representing the intended symbols, without explicit indications from the user about where one symbol ends and the next one begins. This is a difficult problem since the number of possible stroke groups increases exponentially with the number of strokes. Additionally, in many types of sketches, more than one symbol can be drawn in a single pen stroke, while conversely there are many other

*Corresponding author. Tel.: +1 951 827 7719; fax: +1 951 827 2899.
E-mail address: stahov@engr.ucr.edu (T.F. Stahovich).

0097-8493/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2005.05.007

common symbols that are typically drawn with multiple strokes.

To avoid the complications of parsing, many current sketch interpretation 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 [1–4]. Other systems require each object to be drawn in a single pen stroke [5,6]. Such constraints on the drawing process, however, often result in a less than natural drawing environment.

Our approach to parsing begins by segmenting the pen strokes into lines and arcs. A combination of geometric and domain-specific knowledge is then used to locate the symbols. One of our two symbol locators identifies candidate symbols by finding areas with high concentrations of pen strokes, which we refer to as areas of “high ink density”. The other locator identifies candidates by finding points in the temporal sequence of segments at which there are changes in the geometric characteristics of the segments. Once the candidates have been enumerated, domain-specific knowledge is used to prune out candidates that are unlikely to be symbols. Our parsing approach allows for multiple symbols to be drawn in the same stroke, and allows individual symbols to be drawn in multiple strokes.

Once the parser has located the symbols in a sketch, the next task is to identify them. We have developed a general purpose symbol recognizer for this task, but the details of this are beyond the scope of this article. After the sketch has been parsed and recognized, our sketch interpreter examines the internal context of the sketch to automatically correct typical processing errors. Domain knowledge is used to determine if the interpretations of the various pieces of the sketch are consistent with the interpretation of the sketch as a whole. If not, parsing and recognition are revisited so as to eliminate the inconsistencies.

Our system is designed to work for network-like diagrams containing isolated, non-overlapping symbols that are linked together. Examples include analog electric circuits, logic circuits, data flow diagrams, algorithmic flowcharts, and various other graphical models. As an illustration of our system’s capabilities and performance, we developed a sketch-based interface for SPICE, an electric circuit analysis program. Our system is called AC-SPARC for analog circuit sketch parsing, recognition, and error correction.

The electric circuit domain was chosen because of the challenges it presents. For example, in circuit sketches single pen strokes often contain multiple symbols. Additionally, when viewed in isolation, an individual pen stroke representing part of an electrical component is often difficult to distinguish from a stroke representing a wire.

The next section provides a review of related work. This is followed by an overview of the capabilities of our system and a discussion of how users interact with it. The details of the system’s operation are then presented, followed by the results of user studies evaluating the system. Finally, proposed future work is discussed, and conclusions are presented.

2. Related work

Some sketch-based systems facilitate parsing by requiring objects to be drawn with a predefined sequence of pen strokes [7,8]. While useful at reducing computational complexity, the strong temporal dependency in these methods forces the user to remember the correct order in which to draw the strokes. Our approach requires only that one symbol be completed before the next, and places no other constraints on the order in which the strokes are made.

Shilman et al. [9] present an approach to ink parsing that relies on a manually coded visual grammar. A large corpus of training examples is used to learn the statistical distributions of the geometric parameters used in the grammar, resulting in a statistical model. The grammar, and hence the statistical model, defines composite objects hierarchically in terms of lower level objects. The lowest level objects are single stroke symbols recognized with Rubine’s method [10]. Thus, their method requires that the lowest level objects—individual pen strokes—be recognizable in isolation, although ambiguity at this level can be handled naturally by their Bayesian approach. Our approach does not rely on low level objects that are recognizable in isolation. Additionally, the authors suggest that their approach may not scale well to large sketches, while our approach scales linearly with the size of the sketch. Finally, their approach assumes that shapes are drawn in certain preferred orientations. This is a property of both their grammar and Rubine’s method. Our approach is insensitive to orientation.

Alvarado and Davis [11] have developed a parsing approach based on dynamically constructed Bayesian networks. The approach is similar to that of Shilman et al. [9], but the lowest level objects are geometric primitives, such as lines and arcs, rather than symbols that must be recognizable in isolation. The approach uses both bottom-up and top-down reasoning, along with hypothesis pruning, to achieve efficiency and tolerance for low-level recognition errors. They used their method to implement a system for interpreting circuit diagrams. Comparing our user study results with theirs suggests that our approach may be moderately more effective for circuits than their approach. However, because the two user studies involve different test problems, precise comparison of the results is not possible.

Kara et al. [12,13] present a hierarchical parsing and recognition approach based on a "mark-group-recognize" architecture. First, a preliminary recognizer is used to identify "marker symbols", symbols that have unique geometric and kinematic properties that allow them to be easily extracted from a continuous stream of input. The marker symbols are then used to efficiently cluster the remaining strokes into distinct groups, each corresponding to an individual symbol. Finally, a symbol recognizer [14] is used to recognize the identified clusters. The technique may be applicable to electric circuits, but as yet marker symbols have not been identified for this domain.

Costagliola and Deufemia [15] present an approach based on LR parsing for the construction of visual language editors. Their method is intended for use with pre-recognized shapes (icons selected from a menu), and thus is not directly applicable to sketch understanding problems. Saund et al. [16] present 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, as their application does not require recognition of the graphical objects.

A few sketch-based interfaces have been developed for electric circuits. Narayanaswamy [4] developed a sketch-based interface for SPICE which uses hard-coded recognizers that assume a fixed drawing order. Also, the system avoids issues of parsing by requiring the user to pause between symbols. Hong and Landay [17] demonstrated the capabilities of their SATIN system by creating Sketchy SPICE, a circuit CAD tool capable of recognizing AND, OR, and NOT gates, and the wires connecting them. Gates must be drawn in either one or two strokes. Lee [18] describes a trainable recognizer for electric circuit symbols. It was developed for use with scanned bitmap images rather than sketches for which the pen trajectories are available as sequences of time-stamped coordinates. Lee's approach requires that a symbol be drawn using at most two strokes.

3. System overview

Our sketch-based interface is designed to put minimal constraints on the way the user sketches. The user can draw a symbol with any number of strokes, and each instance of a symbol can contain a different number of strokes. There are no requirements that the parts of a symbol be drawn in the same order in every instance. The user can also draw multiple symbols in the same stroke, without lifting the pen, as shown in Fig. 1. To the best of our knowledge, no other sketch interpretation system is capable of distinguishing between two symbols drawn in such a manner. The only constraint imposed by our system is that the user must finish drawing one symbol before starting the next. Thus, one cannot begin drawing one symbol, start on a second, and then return to the first. However, our observations of users indicate that people do not ordinarily draw diagrams in such an unfocused way.

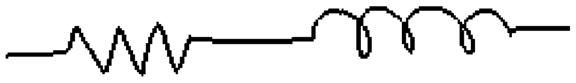


Fig. 1. Multiple symbols can be drawn in the same pen stroke.

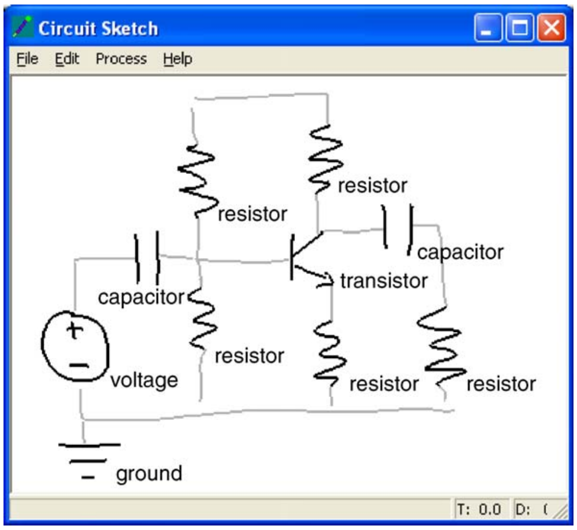


Fig. 2. Interpreted sketch with color coding and text labels to indicate the identified symbols.

Our interface was designed for use with a Wacom Cintiq LCD digitizing tablet and stylus, which enables users to draw directly on the computer display. The user can choose to view the sketch as raw pen strokes or as cleaned-up line and arc segments. Once the sketch is complete, the user selects a menu option with the stylus, thus causing the program to interpret the sketch. After the sketch has been processed, the identified symbols are indicated with color coding and text labels, as shown in Fig. 2.

Our system allows the user to easily correct common interpretation errors. If the program fails to locate a particular symbol, the user can explicitly mark it by holding down a button on the stylus and circling it. If any non-symbol ink is mistakenly identified as a symbol, it can be corrected by holding down a button on the stylus and drawing a diagonal line through it. If a symbol is misclassified, the user can hold down a button on the stylus and tap the point on the symbol. A dialog box will open containing a list of possible classifications for the symbol, and the user can use the stylus to select the appropriate one.

Our interface allows symbols to be easily added to or removed from the sketch as the design evolves. Users can erase ink by using the eraser end of the stylus, just as one would use a pencil eraser. Additional symbols can be added to the sketch at any time by simply drawing them. After modifying the sketch, the user must select a menu item to cause the program to reprocess the sketch and generate a new interpretation of it.

Rather than requiring all users to draw the same way, our system is designed to enable each user to quickly train it to match his or her style. The system typically requires only about five examples of a given symbol to recognize it reliably. The system is initially trained with one drawing style (that of one of the authors). The user can immediately choose to retrain the system by providing examples of each of the symbols. Alternatively, the user can use the system as is and let it adapt to his or her style over time. Each time the system processes a sketch, all of the recognized symbols serve as additional training examples. Thus, through ordinary use, the system begins to learn the user's style.

4. Technical details

Our approach to understanding a sketch is based on the architecture shown in Fig. 3. The first step involves decomposing the user's pen strokes into line and arc segments that closely match the original ink. This process, called ink segmentation, provides compact descriptions of the pen strokes that facilitate parsing and recognition. Next begins the first step of parsing in which geometric tests are used to locate candidate symbols. These are then classified using our symbol recognizer. This is followed by the second step of parsing in which knowledge about the particular domain of the sketch is used to prune the list of candidate symbols. Finally, domain knowledge is used to automatically correct errors made in the previous steps. This process results in a final interpretation of the sketch. The following sections describe each of these steps in detail.

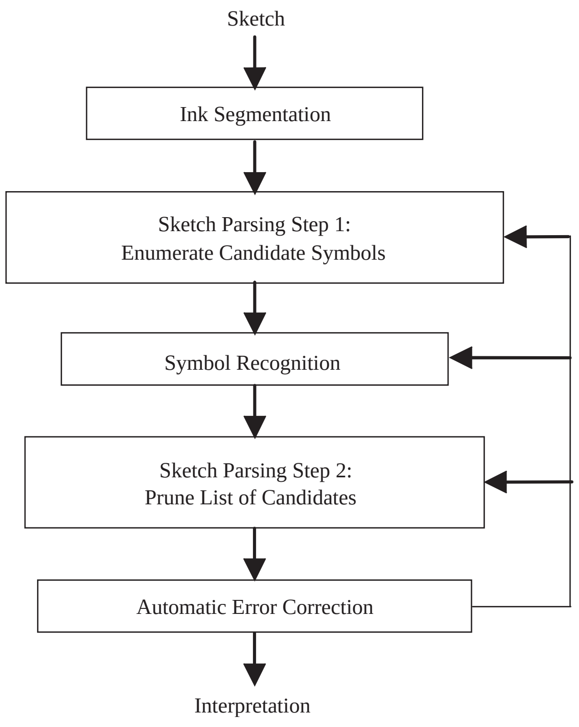


Fig. 3. Architecture of the sketch interpreter.

4.1. Segmentation

To facilitate parsing and recognition, the system segments the pen strokes into individual lines and arcs that closely match the original ink. This is done using the segmenter described by Stahovich [19]. Segmentation involves searching along each stroke for "segment points", points that divide the stroke into geometric primitives. These points are distinguished by both the motion of the stylus tip observed while the stroke was drawn, 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 changes. Additional segment points occur at the start and end of each pen stroke. Once the segment points have been identified, least squares analysis is used to fit lines and arcs between the segment points. Finally, a feedback process is employed in which segment points are added and removed as necessary to improve the segmentation.

4.2. Sketch parsing step 1: enumerating candidate symbols

The first step in our parsing approach is to enumerate candidate symbols. As mentioned previously, we require the user to finish drawing one symbol before drawing a connector or another symbol. Therefore, when locating candidate symbols, we need to consider only consecutively drawn segments. To further reduce the search space, we also establish limits for the number of segments that a symbol may contain. The lower limit is two, since it is uncommon that a symbol is represented by a single line or arc segment.[1] The upper limit depends on the particular user's drawing style. We determine this by examining the user's training data and finding the symbol drawn with the largest number of segments. In practice, this number is typically between 6 and 12. We

[1]Electric circuits do not contain symbols comprised of a single line or arc segment, and no domains we have considered contain such symbols.

use this value, plus a tolerance of 2, as the maximum number of segments to consider when enumerating candidate symbols.

These two constraints significantly reduce the size of the search space. If instead we considered every possible combination of segments for a sketch containing N segments, the number of combinations would be of order 2^N. However, by considering only combinations containing between 2 and k segments, the number of combinations is reduced to order N^k. Using only consecutively drawn segments reduces this even further to order kN. As computational cost is linear in the size of the sketch, our approach scales well to large sketches. Note that the key to the scalability of our approach is the assumption that the user always completes drawing one symbol before beginning the next. In the experiments we have performed, this assumption has been justified. Further investigation is necessary to explore how well this assumption holds for more complicated sketches.

Candidate symbols are enumerated using two types of geometric tests to identify possible starts and ends of the symbols. The first test looks for regions in which there is a high concentration of ink. The second looks for changes in the characteristics of the segments, such as when a long segment is followed by a much shorter segment, or when a line segment is followed by an arc. These tests are described in detail below.

4.2.1. *Using ink density to locate symbols*

Symbols usually consist of a high concentration of ink, while the ink of connectors is often more spread out. Our ink density approach identifies candidate symbols by searching for these regions of high ink density. More specifically, we search for sequences of segments having the property that the addition of another segment to either end of the sequence causes a decrease in density, as this is an indication of adding a connector segment. We define *ink density* as the ratio of the square of the ink length to the area of the oriented bounding box of the ink:

$$density = \frac{ink\_length^2}{oriented\_bounding\_box\_area}. \qquad (1)$$

Here, in addition to the actual ink shown on the screen, the ink length also includes the ‘‘hidden ink’’, which we define as the ink that would occur if the user did not pick up the stylus while drawing. Hidden ink is assumed to consist of straight line segments between pen up and pen down locations. For example, the hidden ink of a voltage source is shown by the dotted lines in Fig. 4. Including the hidden ink accentuates the density of symbols drawn with multiple strokes, thus making them easier to locate. We square the ink length so that it scales the same way as bounding box area, thus making the density parameter insensitive to uniform scaling. The oriented bounding box of the ink is the smallest rectangle, not necessarily aligned with the coordinate axes, that contains all of the segments comprising the ink.

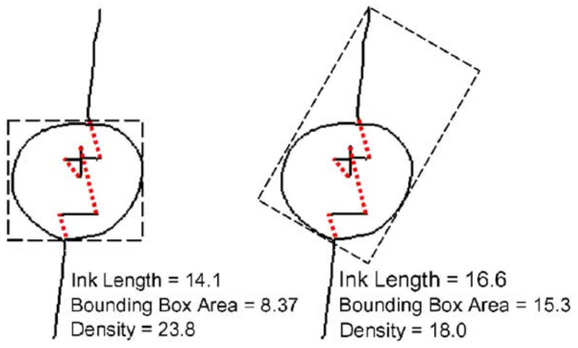


Fig. 4. Density decreases when a segment is added to the end of the voltage source symbol. Hidden ink is shown by dotted lines. Bounding boxes are shown by dashed rectangles.

A symbol consists of a sequence of line and arc segments, beginning with a start segment and ending with an end segment. The ink density analysis uses a forward–backward algorithm to find the start and end segments of candidate symbols. The forward step is used to find the end segment of a symbol by finding segments whose addition to the sequence significantly decreases the sequence’s density. In the backward step, the best start segments are located for each end segment, again by looking for decreases in density.

In the forward step, the approach starts with a given segment and considers increasingly long sequences of consecutively drawn segments. Each time a segment is added to the sequence, the density is computed. If there is a decrease in density of 20% or more,[2] it is quite possible that a connector segment was added to the sequence. In this case, the previous segment is deemed a possible end of a symbol. This may not be the best end, however, and thus additional segments continue to be added to the sequence until the user-specific maximum number of segments is reached. Each time there is a decrease in density of 20%, another candidate end segment is identified. Thus, starting from a given start segment, multiple sequences, each with a different end segment, may be found. This process is repeated, starting from each segment in the sketch, and a list of possible symbol end segments is created.

Consider applying this method to the sketch in Fig. 5. Table 1 shows the sequence that starts with Segment 5.

[2]This tolerance was determined empirically by collecting sample circuit sketches from several users and experimenting with different thresholds until the best parsing rate was obtained.

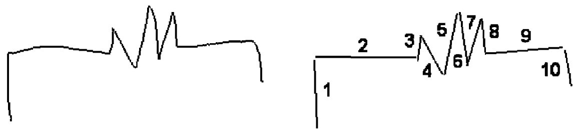
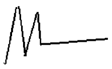
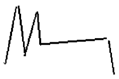


Fig. 5. Sketch used to illustrate the density method for locating symbols.

Table 1
Forward step: finding a possible end segment for a symbol

Ink	Start segment	End segment	Density	Density change (%)
	5	6	9.5	
	5	7	17.8	87.4
	5	8	16.9	−5.1
	5	9	10.9	−35.5
	5	10	12.0	10.1

The initial sequence consists of Segments 5 and 6. Segment 7 is then added to this sequence, and the change in density is calculated. The remaining segments are added one at a time, and the change in density is calculated at each step, until the user-specific maximum number of segments is reached. The decrease in density resulting from the addition of Segment 9 to this sequence is 35.5%, indicating that Segment 9 is a potential connector segment. As a result, Segment 8 is considered a possible end segment.

Note that there need not be a connector drawn after each symbol. Should the user desire, he or she can first draw all the symbols in the sketch, and then draw the connectors. The ink density analysis will still locate the symbols, since the gap that typically occurs between symbols will decrease the density in the same manner that a connector segment would.

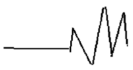
In Fig. 5, we now know that Segment 8 is a possible end segment, but we have no verification that Segment 5 is a good choice for the start segment. This segment could be part of a connector, or it could be somewhere in the middle of a symbol, as in fact it is. Thus, the task of the backward step is to find the start segment of each symbol.

In the backward step, ink density analysis is once again applied. Starting with each possible end segment, prior segments are added one at a time until the user-specific maximum number of segments is reached, and the changes in density are monitored with each addition. The segment drawn after the segment whose addition causes the largest decrease in density is selected as the start segment for the given end segment. The sequence is now considered a candidate symbol. If there is no segment whose addition to the beginning of the sequence causes a decrease in density, then the sequence is discarded. In this case, the sequence likely consists only of connector segments.

Table 2 illustrates the process of finding a start segment for end Segment 8. The initial sequence consists of Segments 8 and 7. A segment is added to the start of this sequence, and the change in density is calculated. This is repeated until the user specific maximum number of segments is reached. Since the addition of Segment 2 to the sequence causes the density to decrease by 23.0%, a bigger decrease than for the addition of any other segment, Segment 3 is considered the best start segment for the sequence. Therefore, the resulting candidate symbol consists of Segments 3–8, which corresponds to the resistor in Fig. 5. Note that in this example, the sequence of segments from 4 to 8 actually had higher density than the sequence of segments from 3 to 8. However, since the density of the former would not have decreased significantly with the addition of another segment to its start, it is not considered a candidate symbol.

When this forward-backward process is applied to the entire sketch, a list of candidate symbols is obtained. All sequences that survive this analysis have the property that adding a segment at either end of the sequence results in a density decrease, and thus all such sequences

Table 2
Backward step: finding the best start segment for a given end segment

Ink	Start segment	End segment	Density	Density change (%)
	7	8	9.8	
	6	8	14.0	42.9
	5	8	16.9	20.7
	4	8	19.1	13.0
	3	8	18.7	−2.1
	2	8	14.4	−23.0

are considered candidate symbols. There is no guarantee, however, that all such sequences have been found. We have discovered that the addition of another segment at one end of a symbol usually results in a large decrease in density, but this is not always true for both ends. In the forward step, we commit to finding symbols for which adding a segment to the end would decrease the density by at least 20%. The only requirement in the backward step is that there is some decrease in density caused by adding a segment to the start of the symbol. However, some symbols have the property that adding a segment to the start causes a large decrease in density, while adding a segment to the end causes only a small decrease. Thus, to find the additional candidate symbols, we repeat the analysis with time, in effect, reversed. A backward step is performed first, and possible choices for start segments are located by searching for decreases in density greater than 20%. A forward step is then applied to find the best end segment for each start.

4.2.2. *Using differences in segment characteristics to locate symbols*

There are usually large differences between a connector and the first segment of a symbol, and between the last segment of a symbol and the subsequent connector. Our second symbol locator finds symbols by identifying those differences. For each segment in the sketch, we calculate four characteristics and compare them to those of the segment before it. These characteristics include: (1) Segment type: line vs. arc; (2) Segment length: the lengths of two segments are considered different if one is more than 40% longer than the other; (3) Segment orientation: if the acute angle between two segments differs by 65° or more, they are considered different in this characteristic; (4) Intersection type: classified as none, endpoint-to-endpoint (“L”), endpoint-to-midpoint (“T”), or midpoint-to-midpoint (“X”).

We define a good candidate symbol to be a group of segments that are similar in these four characteristics but which differ from the other segments touching the group. For example, the inductor in Fig. 6 is easily distinguishable as a series of short arc segments, with longer line segments, representing wires, on either side. As this example illustrates, transitions between connectors and symbols often occur between segments having significant differences in the four characteristics. In practice, we consider any pair of consecutively drawn segments that differs in two or more characteristics to be a possible transition between a symbol and a connector. The point between such a pair of segments is referred to as a “segment difference point”. Fig. 6 shows all such points for a typical circuit sketch.

Candidate symbols are defined to be sequences of segments, bounded by two segment difference points,

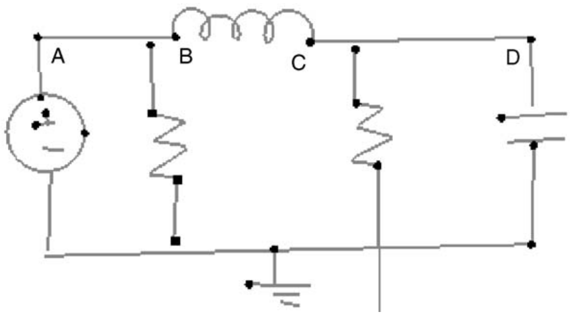


Fig. 6. Example of segment difference analysis. Possible transitions between symbols and connectors are show as dots.

containing between two and the user-specific maximum number of segments. Note that the points bounding candidate symbols need not be consecutive, and thus candidates can overlap. For instance, candidate symbols for Fig. 6 include the sequences of segments between points A and C, A and D, B and C, B and D, and so on. While this approach finds many valid symbols, it also locates many non-symbols. The latter are pruned using domain knowledge, as described in Section 4.4.

4.3. *Symbol recognition*

The task of the symbol recognizer is to classify each candidate symbol. The recognizer takes as input the segments comprising a candidate symbol and returns the definition model that best matches those segments. Our recognizer uses training examples to construct a probabilistic definition model of each symbol, based on geometric features of the segments. A statistical approach is used to match an unknown symbol to one of these definitions. This statistical approach naturally accounts for the variations inherent in hand-drawn sketches and allows symbols to be drawn using any number of strokes drawn in any order. The recognizer is insensitive to size and orientation, and it is robust to moderate non-uniform scaling.

To train the recognizer, the user draws several examples of a symbol. The examples are segmented (Section 4.1), and a set of nine geometric features are extracted from each example. These features include the number of: pen strokes, line segments, arc segments, endpoint (“L”) intersections, endpoint-to-midpoint (“T”) intersections, midpoint (“X”) intersections, pairs of parallel lines, and pairs of perpendicular lines. The final feature is the average distance between the endpoints of the segments. This average is normalized by the maximum distance between any two endpoints, thus accounting for uniform scaling. This feature helps the system differentiate between objects that contain non-uniformly scaled versions of the same segments. For example, the average distance between endpoints of a square is larger than that of a rectangle.

Once the values of the nine features have been determined for each of the training examples of a symbol, a statistical definition model is constructed. We assume a Gaussian distribution for the feature values, and the mean and standard deviation are calculated for each feature. A Gaussian model naturally accounts for variations in the training examples. However, because eight of the features assume only discrete values, and since we aim to use only a handful of training examples which may happen to show little difference in some features, 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 kinds of symbols considered.

During recognition, we use a statistical classifier to determine which definition is the best match for an unknown symbol. 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 model, D_i. The symbol is classified by the definition D^* that maximizes the probability of match:

$$D^* = \underset{i}{\operatorname{argmax}}\, P(D_i|S). \tag{2}$$

We assume that all definitions are equally likely to occur, and 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. Bayes' Rule tells us that the definition which best classifies the symbol is therefore the one that maximizes the likelihood of observing the symbol's individual features:

$$D^* = \underset{i}{\operatorname{argmax}} \prod_j P(x_j|D_i). \tag{3}$$

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 $\sigma_{i,j}$:

$$P(x_j|D_i) = \frac{1}{\sigma_{i,j}\sqrt{2\pi}} \exp\left[\frac{-(x_j - \mu_{i,j})^2}{2\sigma_{i,j}^2}\right]. \tag{4}$$

Because we are assuming that the features are independent, this is referred to as a naïve 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 inter-related. For example, the number of intersections in a symbol frequently increases with the number of line and arc segments. Domingos and Pazzani [20] have suggested that the naïve Bayesian classifier does not require independence of the features to produce useful results: While the actual values of the probabilities of match may not be accurate, the rankings of the definitions are often 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 often only a small number of training examples are used, 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 training examples have been used.

Once the recognition process is complete, the symbol is added to the set of training data for the identified definition, and the mean and standard deviation of each feature are recalculated. Thus, each time a symbol is recognized, the definition of that symbol becomes more accurate and better tuned to the particular user's drawing style.

4.4. *Sketch parsing step 2: pruning list of candidates based on domain knowledge*

The final step in the parsing process is to use information specific to the particular domain of the sketch to prune out the candidates that do not represent symbols. For this step, the system uses several heuristics to help determine if a group of segments really is a symbol. The basic approach is to collect information supporting and refuting the fact that a group of segments is a symbol. Each candidate is initially assigned a score of zero. Points are then added to the score for each positive indication that is observed, and subtracted for each negative indication that is observed. Below is a description of the heuristics used to assign points for our electric circuit application.

Positive evidence that a group of segments is an electric circuit symbol includes the following:

- The density of the candidate symbol is higher than that of any overlapping candidates (+1).
- The probability of match between the candidate symbol and its definition is greater than 0.5%[3] (+1). This is equivalent to each of the nine features of the

[3]This tolerance, as well as the others used for parsing and error correction, was determined empirically by collecting sample circuit sketches from several users, and comparing groups of segments that represented circuit components with those that did not.

symbol having a 56% probability of match with the corresponding features of the definition. An additional point is given if the probability of match is greater than 10%, and is higher than that of all overlapping candidates (+1).

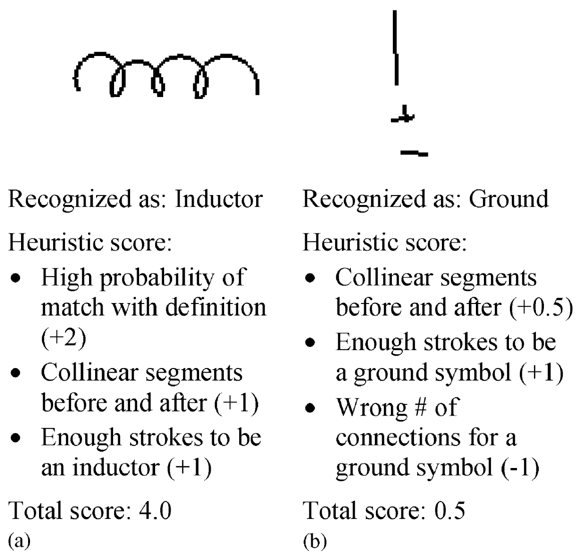
- The candidate symbol contains enough strokes to be the symbol it has been classified as (+1). For example, a current source requires at least two strokes, one for the circle and one for the arrow. Note that the recognizer does not automatically exclude definitions on the basis of stroke count so as to allow for missing strokes and situations in which the user did not lift the pen between strokes.
- The candidate contains nearly a full circle, i.e., an arc of at least 300° (+1). This is a good indication of a symbol since wires typically are not drawn as circles.
- There are collinear segments attached to either side of the candidate (+1). This is a good indication of a symbol because circuit components often have a wire on either side. So as not to penalize ground symbols which should have a wire on only one side, ground symbols are automatically given half a point ($+\frac{1}{2}$).

Indications that a group of segments is not an electric circuit symbol include the following:

- The bounding box of the candidate symbol is very thin, indicating it likely contains only wires (−1). A bounding box is considered too thin if its length-to-width ratio is greater than 8.
- The bounding box of the candidate is large in comparison to the bounding boxes of the other candidates. Typically, the symbols in a circuit are of similar size. If the bounding box area of a candidate is larger than the mean bounding box area plus one standard deviation, one point is subtracted (−1). An additional point is subtracted if the bounding box area is larger than the mean plus two standard deviations (−1).
- The average length of the segments in a candidate symbol is very long in comparison to the average length of the segments in the other candidates. A large average length indicates the candidate likely contains wires. If the average length of a candidate is larger than the average of the other average lengths plus one standard deviation, one point is subtracted (−1). A second point is subtracted if the average of the average lengths is exceeded by two standard deviations (−1).
- The candidate symbol contains three or fewer segments and they are all connected by endpoint-to-endpoint ("L") intersections (−1). This pattern is typical of a sequence of wires.
- The density of the candidate symbol is less than 10 (−1). An additional point is subtracted if the density is less than 7 (−1).
- The candidate symbol has the wrong number of electrical connections. A point is subtracted for each additional or missing connection (−1 times the difference between the actual and the correct number of connections). E.g., for a resistor with only one connection, one point is subtracted.

For a group of segments to be considered a symbol, its score must be greater than a threshold. We use a threshold of one point for our electric circuit application, but this may vary between applications. Since two symbols cannot share a segment, the final pruning step is to remove candidates that overlap other candidates having higher scores. Any segment not identified as part of a symbol is considered a connector.

Fig. 7 shows the scores for two of the candidate symbols located in the circuit sketch shown in Fig. 6. The arc segments shown in Fig. 7(a) were correctly identified as an inductor. This symbol had a high probability of match with the inductor definition, had collinear segments touching each side of it, and was drawn with the typical number of strokes for an inductor. This symbol was not pruned because its heuristic score of 4.0 was higher than that of any overlapping candidates. The group of line segments shown in Fig. 7(b) was best classified as a ground symbol. However, the bottom three lines in this candidate are actually part of a voltage source, and the top line is actually a wire. This symbol was given half



(a)

Recognized as: Inductor

Heuristic score:

- High probability of match with definition (+2)
- Collinear segments before and after (+1)
- Enough strokes to be an inductor (+1)

Total score: 4.0

(b)

Recognized as: Ground

Heuristic score:

- Collinear segments before and after (+0.5)
- Enough strokes to be a ground symbol (+1)
- Wrong # of connections for a ground symbol (-1)

Total score: 0.5

Fig. 7. Heuristic scores for two candidate symbols from Fig. 6. (a) Four arc segments making up an inductor. (b) Four line segments that do not represent a symbol: the bottom three lines are actually part of a voltage source and the top line is actually a wire.

a point because, as a ground symbol, it would not be required to have collinear lines touching it. It was given another point because it was made up of four strokes, which is more than the minimum required to draw a ground symbol. However, a point was subtracted from this symbol's heuristic score because it had two connections, whereas a ground symbol should have only one. This symbol was pruned because its heuristic score of 0.5 was below the threshold used for our application.

The heuristics used to prune the list of candidates are sensitive to the accuracy of the symbol recognizer. For example, if a symbol has been classified incorrectly, the heuristics may expect the wrong number of connections, thus unfairly penalizing the candidate. However, domain knowledge can be used to correct these kinds of recognition and parsing errors as described in the next section.

To illustrate how the parsing steps work together, consider the circuit diagram shown in Fig. 6, which contains 36 segments. The two geometry-based symbol locators found a total of 78 candidate symbols. Of these candidates, 12 remained after pruning the candidates with low heuristic scores. After removing candidates that overlapped higher scoring candidates, six symbols remained. These six were the intended voltage source, resistors, inductor, capacitor, and ground symbol.

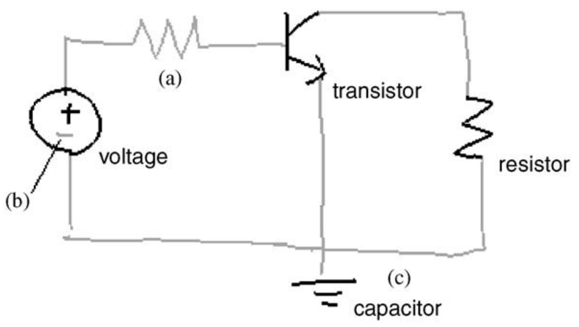


Fig. 8. Examples of mistakes that the system can find and fix. (a) Missed symbol (resistor) that was diagnosed as a wire with too many consecutive segments. (b) Parsing error diagnosed as a dangling wire. (c) Recognition error identified by considering the number of connections: capacitors should have two connections, but this symbol has only one.

4.5. Automatic error correction

Once the parsing and recognition steps are complete, the system knows the locations of the symbols, and the connections between them. At this point, the system can use domain specific knowledge to correct parsing and recognition errors. Here we summarize our approach for circuits. While this approach is specific to circuits, other domains will likely have similar sorts of debugging rules.

One indication that there may be a parsing problem in an electric circuit is that a large number of consecutively drawn segments have been identified as wires (Fig. 8a). It is uncommon for a user to draw wires this way, thus suggesting that a component has been missed. In such situations, the system first tries to find the missed component by lowering the threshold for heuristic pruning. If a component is still not found, a misclassification may have caused the parser to err. In this case, the system considers lower ranked classifications for any candidate components that contain the wire segments in question. If the score of one of those candidates is now above the heuristic threshold, the system keeps that candidate and its new classification.

Because wires are used to connect components, it is unlikely that a wire will have fewer than two connections (Fig. 8b). (Connections are identified as intersections between segments. A tolerance proportional to the length of the segments in question is used when identifying intersections.) If a wire is found to have too few connections, the system first checks to see if it should belong to a nearby component. If the segment drawn before or after the wire has been recognized as part of a component, the wire is added to that component, so long as its heuristic score would not decrease by doing so. If adding the wire does decrease the heuristic score, the wire's length is extended by up to 30% to see if any more connections are found. If a connection is found, we assume that it was intended in the original sketch, and the wire is kept at the new length. Since a segment initially recognized as a wire may simply be an extra segment caused by an unintended pen stroke, the system leaves the segment as a disconnected wire if these methods fail.

Another indication of a problem is that a component has the wrong number of connections (Fig. 8c). This is often a result of incorrect classification by the symbol recognizer. This problem is sometimes fixed by replacing the classification of the symbol with the second or third choice of the recognizer, as long as the heuristic score of the symbol does not decrease. Otherwise, the system assumes that the problem is due to the sketchiness of the drawing: two segments that were intended to intersect did not, or two segments that were not supposed to intersect did. Thus, the segments comprising the symbol are extended or shortened by up to 30%. If the correct number of connections is found, the segment responsible for the repair is kept at the new length, while all other segments are returned to their original lengths.

Occasionally, critical problems still remain after the error correction step is complete. For example, a component may still have the wrong number of connections. If such critical problems remain, the user is presented with a message describing the problem and is asked to fix it. At this point, the user can add or remove symbols from the sketch. Additionally, using the editing techniques described in Section 3, the user can manually correct parsing and recognition errors.

5. Evaluation

To test the performance of AC-SPARC, we asked 10 subjects to train the system and draw a series of electric circuits. The subjects were all engineering students, and each had previously taken at least one class that required them to draw and analyze electric circuits. Only one subject (Subject 8) had prior experience with a digitizing tablet, although several subjects had used a PDA.

The test was performed using a Wacom Cintiq LCD tablet and stylus, which enabled the subjects to draw directly on the computer display. The subjects were first asked to train the system by providing six examples of each circuit component including: resistors, capacitors, inductors, transistors, grounds, voltage sources, and current sources. Next, the subjects were then presented with precise drawings of 8 circuits, which they were then asked to draw. These circuits contained between 6 and 16 components, with an average of 9.25 components per circuit. Appendix A shows sketches of the circuits drawn by the test subjects.

The subjects sketched in the raw ink view, and thus did not see how their circuits were segmented. The subjects were given no information about how the system works, and they were told only that they should finish drawing one symbol before drawing a wire or starting another symbol. Subjects were also instructed in the use of the three editing gestures described in Section 3: circling symbols that the system did not locate, drawing a line through any ink that the system incorrectly identified as a symbol, and tapping on an incorrectly classified symbol to change its classification. After a circuit was parsed and each symbol was recognized, the subjects were asked to use these gestures to correct any mistakes made by the system. After a subject finished editing a sketch, the symbols were automatically added to the training data for the symbol recognizer.

As shown in Table 3, the system performed reasonably well for most subjects. For the typical subject, the program correctly located and identified between 74% and 90% of the symbols, prior to any editing to correct mistakes. The average across all users was 77%. When errors occurred, on average, only 2.64 editing gestures per sketch were applied to correct them (excluding Subject 6, who is discussed below). After editing, on average, 95% of the symbols were correctly located and recognized. Ideally, this would be 100%, however, sometimes segmentation problems prevented complete error correction. We do have gestures that allow users to correct segmentation mistakes, but we did not include this capability in these studies.

Table 3
Results of the AC-SPARC user study

Subject	Symbols correctly located and recognized (before edits) (%)	Average no. of extra symbols per circuit	Average no. of edits per circuit
1	74.3	0.000	3.75
2	90.4	0.375	1.38
3	75.3	0.000	3.50
4	76.7	0.000	4.88
5	83.6	0.000	1.75
6	37.0	0.500	n/a
7	83.8	0.500	2.38
8	83.8	0.000	2.13
9	82.4	0.125	2.00
10	82.4	0.375	2.00
Average	77.0	0.188	2.64

Table 4
The sources of AC-SPARC's errors

Processing step	Errors attributed to step as % of total no. of errors (%)
Segmentation	35.3
Parsing	50.0
Recognition	14.7

Table 4 shows a breakdown of the sources of errors. Note that this is only an estimate. It is difficult to determine precisely which step caused each mistake because the steps are so interrelated. We used the following assumptions when attributing the errors: If the segmentation of a symbol's ink was incorrect, and this led to the symbol being parsed or recognized incorrectly, an error was assigned to the segmentation step. If the ink was segmented correctly but too few or too many segments were grouped to form a symbol, an error was assigned to the parsing step. If a symbol's ink was segmented and parsed correctly, but the symbol was incorrectly classified, an error was assigned to the recognition step.

As Table 4 shows, the majority of the errors occurred in the parsing step. However, recognition errors could have contributed to some of the parsing errors, as the parser is dependent on the recognizer. Overall, 91% of the symbols that subjects drew were segmented correctly. Of the correctly segmented symbols, 86.7% were parsed correctly, and 95.4% of the correctly parsed symbols were recognized correctly.

During the user study, the pen stroke data for each circuit was saved. We were able to use this data to determine how well the system would have performed if there were no segmentation errors. Using editing gestures not available to the test subjects, we corrected segmentation errors and reprocessed the sketches. As shown in Table 5, the average number of symbols correctly located and recognized by the system improved by 7.1 percentage points to 84.1%. Note that this result does not include any other editing. Furthermore, this result is consistent with our estimated rate of errors due to segmentation shown in Table 4.

We also used the saved data to test the learning capabilities, automatic error correction, and user dependence of the system. These results are shown in Table 5. In the original test, each time the subject finished editing a sketch, the symbols of that sketch were added to the training data. In this new experiment, the symbols were not added, and the training data remained the same for each circuit. From this experiment, it is apparent that the learning capabilities of the system allowed its performance to increase by 5.5 percentage points, on average. Similarly, to examine the performance of the automatic error correction step, we reprocessed the sketches with this step disabled. The performance decreased by 10.9 percentage points, thus indicating that automatic error correction has a significant effect on the system's performance.

Finally, we examined how well the system would have performed if the subjects had not initially trained the system themselves, and had instead used someone else's training data. In this experiment, one of the authors trained the system by providing 10 examples of each symbol. The stored sketches from a particular user were then replayed, one at a time. After a sketch was replayed, processing errors were tabulated. Then, editing gestures were applied to correct parsing and recognition errors, and the symbols were added to the training data. Such editing was necessary to provide accurate training data for the recognizer. However, the system performance reported in Table 5 does not include the direct effects of editing, because errors were tabulated prior to editing. This process was repeated using the data from all ten test subjects and the results were averaged. On average, the results obtained using the author's initial training data were only two percentage points worse than those obtained using the subjects' own initial training data.

Table 5
Results that would have been obtained if the segmentation of the circuits had been perfect, if the system had no learning capabilities or error correction, or if the user did not train the system

Test	Symbols correctly located and recognized (before editing) (%)
Original test	77.0
Segmentation manually corrected	84.1
Recognized symbols not added to training data	71.5
No automatic error correction	66.1
Using default training data rather than the user's	75.0

As shown in Table 3, particularly poor results were obtained for Subject 6. This subject drew the circuits very quickly and more sloppily than the other subjects. The system often failed at the segmentation step. Another subject, Subject 4, did not follow the usual conventions for drawing circuits. This subject began by sketching the general framework of the circuit, as a set of wires, and then sketched the symbols on top of it. Thus, each symbol had only one connection to a wire. Since our parser makes use of the number of connections to a symbol, our parser made frequent errors in this case.

If the results of Subjects 4 and 6 are excluded from the data, the average percentage of symbols correctly located and recognized by the system was 82.0%. Furthermore, with these two subjects excluded, 89.6% of the symbols were correctly located and recognized when the segmentation of each circuit was fixed, and no other editing was performed.

Most feedback from the subjects was positive. Several subjects cited the freedom to draw the circuits in any order and at different angles as useful features. A couple of the subjects who had previously used a SPICE graphical user interface (GUI) stated that they felt our system was much more intuitive and faster to use than the traditional buttons and pull-down menus of a GUI. Most subjects agreed that our system did a reasonable job of locating and identifying symbols, and one subject explicitly stated that he would not expect such a system to perform flawlessly. The most common complaint was that it was sometimes difficult to use our pen gestures to correct the mistakes of the parser and recognizer.

The results we have obtained indicate that there is no individual step of the interpretation process that is the source of all errors. In fact, the segmenter, parser, and recognizer all achieved accuracy rates of over 86%. The principal challenge comes from the fact that mistakes made in one step carry over to the next.

This user study is perhaps a harsh evaluation of our system because the subjects had no previous experience with our system, and because only one of the subjects had used a digitizing tablet before. Our system performed better for the one subject who did have experience with a tablet (Subject 8) than for most of the other users. We suspect that as a user becomes more familiar the tablet and with the interface, the system will begin to perform better. Additionally, allowing the user to view the segmented ink would provide feedback that would also increase performance.

6. Future work

We have found that our segmenter and parser worked well for many users, but produced poorer performance for some. We believe that the performance of these two systems could be improved by enabling the user to train them (the recognizer is already trainable). For instance, the user could provide several standard training examples, and the tolerances and thresholds of the segmenter and parser could be automatically tuned to improve the performance on those examples. This would likely be done using an optimization technique.

AC-SPARC was intended as a test bed for our parsing and recognition techniques. To make a more useful engineering tool, it is necessary to improve some of the basic user interface features. For example, the system offers a cut (erase) function, but needs copy and paste functions. Likewise, the editing gestures were designed for ease of programming, and need to be made more flexible and robust. Finally, the system should provide a method for setting the parameter values for each of the electrical components. Currently, the system assigns a default value of one (1 ohm, 1 V, etc.) to each circuit component. Ideally, the user should be able to write the parameter values with the stylus, either with a dialog box interface, or as a direct annotation on the sketch.

We have demonstrated our techniques in the domain of electric circuits. In the future, we plan to explore the application of these techniques to other network-like domains such as flow charts and UML diagrams.

7. Conclusions

We have developed a sketch parsing technique that can extract hand-drawn symbols from a continuous stream of pen strokes. Our technique does not require the user to provide explicit indications of where symbols start and end, however, it does require that the user complete one symbol before starting the next. Our technique is suitable for network-like diagrams containing isolated, non-overlapping symbols that are linked together. Our parser begins by using geometric reasoning to locate candidate symbols. We employ two location techniques. The first identifies candidate symbols by looking for regions of high ink density. The other identifies candidates by finding points in the temporal sequence of line and arc segments at which there are changes in the geometric characteristics of the segments. Once the candidates have been enumerated, domain knowledge is used to prune away the unlikely candidates.

With our parser, the parts of a symbol can be drawn in any order, symbols can be drawn in multiple pen strokes, and a single pen stroke can contain multiple symbols. The computational cost of our parsing technique is linear in the number of line and arc segments, thus the technique is suitable for interactive sketch understanding systems. Finally, in user studies in the domain of electric circuits, our parser has proven to be reliable at extracting symbols.

Due to the variations, inconsistencies, and ambiguities inherent in hand-drawn sketches, it is difficult to achieve perfect accuracy when parsing and recognizing a sketch. To address this issue, we developed an approach to automatically correct common parsing and recognition errors. Once a sketch has been interpreted, domain knowledge is used to determine if the interpretation of the various pieces of the sketch is consistent with the interpretation of the sketch as a whole. If not, parsing and recognition are revisited so as to eliminate the inconsistencies. The specific techniques we implemented are for the domain of electric circuit sketches, but many of the concepts should generalize to other network-like diagrams.

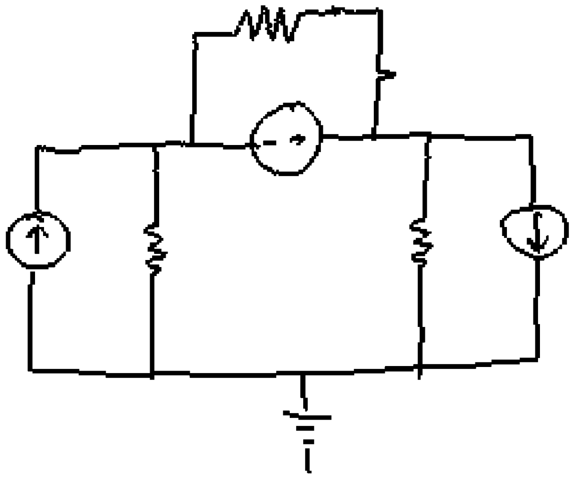
We have used this set of techniques to build AC-SPARC, a sketch-based user interface for the SPICE circuit analysis program. We conducted a user study to evaluate the performance of AC-SPARC, and the results were promising. The system performed with reasonable accuracy, and user response to the system was very favorable. While AC-SPARC can be used to solve real problems, it will require refinement before it can serve as a production engineering tool. Nevertheless, our system has demonstrated that it is possible to create an interface that combines the ease and freedom of pencil and paper sketching with the power of traditional computer software.

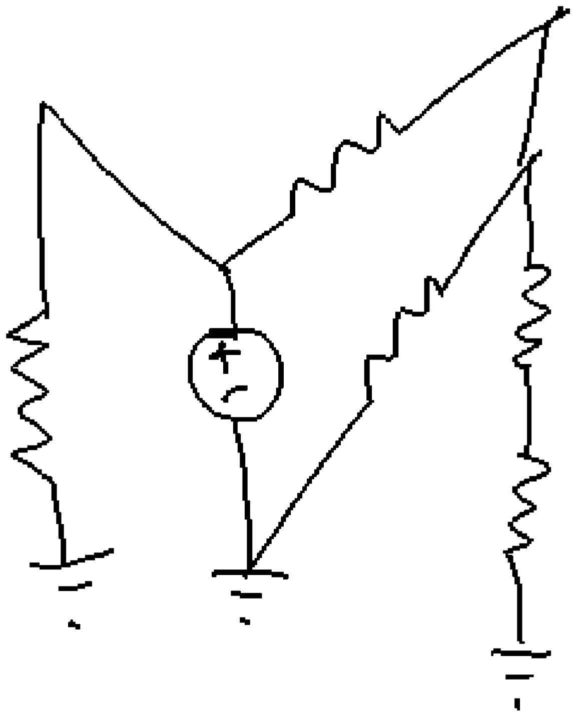
Acknowledgments

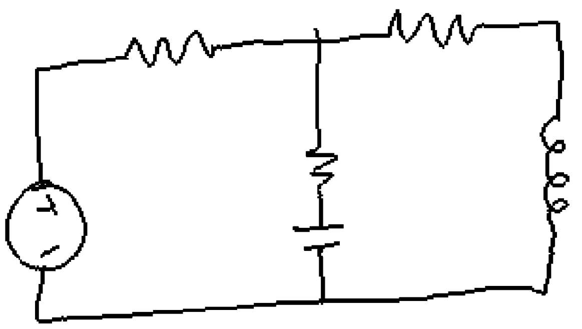
This work was supported in part by the National Science Foundation under grant DMI-0200262.

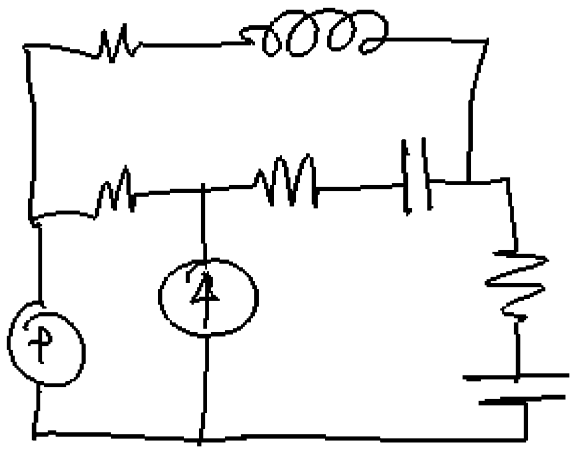
Appendix A. Example circuits from AC-SPARC user study

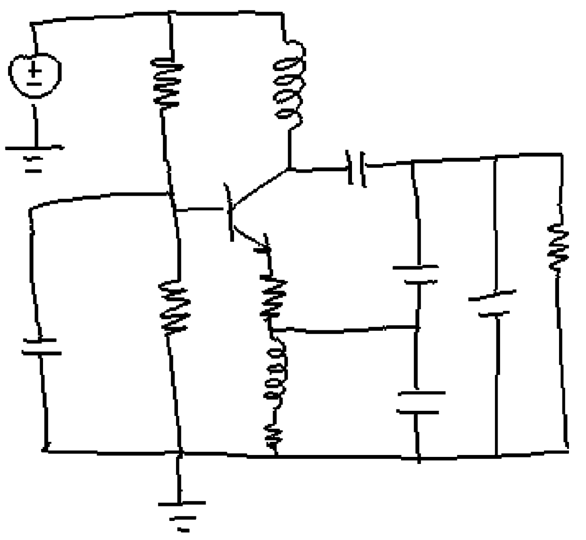
Each circuit was drawn by a different subject from the user study.

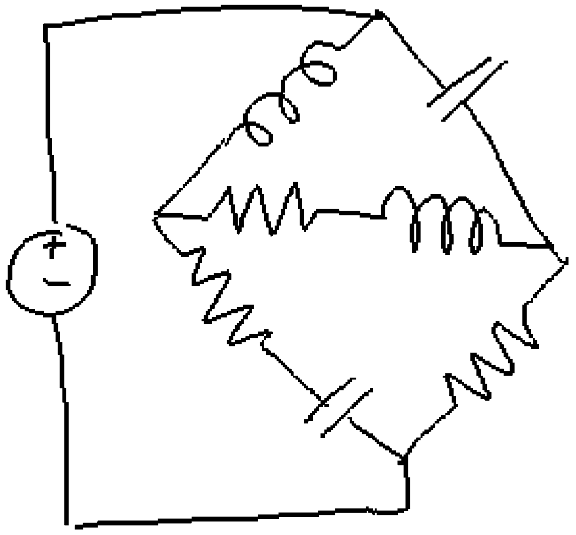


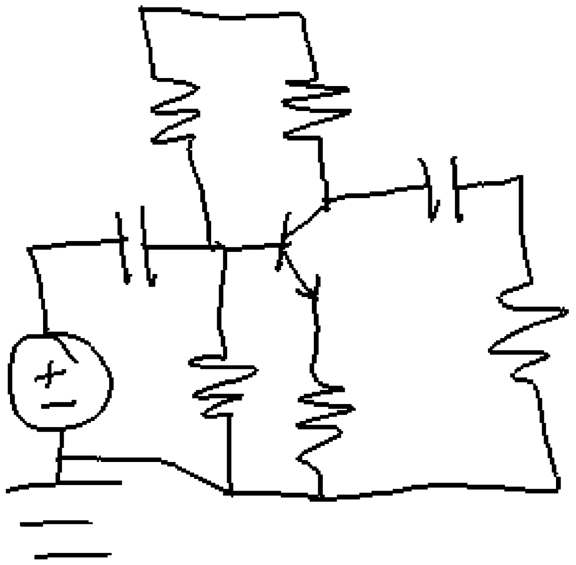


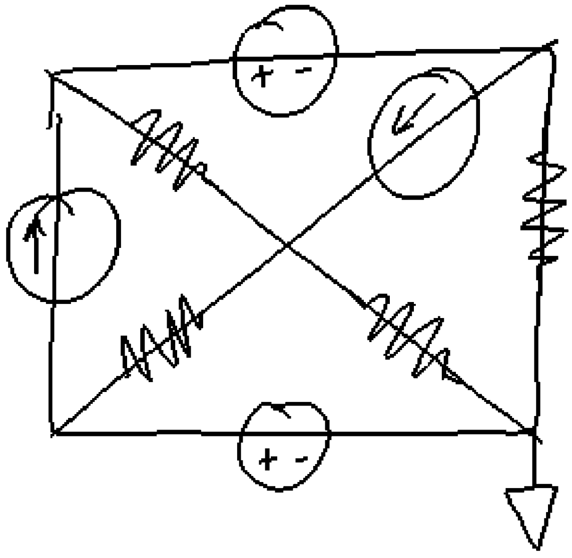












References

[1] Apte A, Vo V, Kimura TD. Recognizing multistroke geometric shapes: An experimental Evaluation. Proceedings of 16th annual ACM symposium on user interface software and technology (UIST'93) 1993:121–8.

[2] Fonseca M, Pimentel C, Jorge J. CALI: an online scribble recognizer for calligraphic interfaces. Proceedings of 2002 AAAI spring symposium on sketch understanding 2002:51–8.

[3] Gross MD. Recognizing and interpreting diagrams in design. Proceedings of the workshop on advanced visual interfaces (AVI'94) 1994:88–94.

[4] Narayanaswamy S. Pen and speech recognition in the user interface for mobile multimedia terminals. Ph.D. thesis, University of California, Berkeley; 1996.

[5] Kimura TD, Apte A, Sengupta S. A graphic diagram editor for pen computers. Software Concepts and Tools 1994;15(2):82–95.

[6] Landay JA, Myers BA. Sketching interfaces: toward more human interface design. IEEE Computer 2001;34(3): 56–64.

[7] Sezgin TM, Davis R. HMM-based efficient sketch recognition. International Conference on Intelligent User Interfaces (IUI'05) 2005.

[8] Yasuda H, Takahashi K, Matsumoto T. A discrete HMM for online handwriting recognition. International Journal of Pattern Recognition and Artificial Intelligence 2000;14(5):675–88.

[9] Shilman M, Pasula H, Russell S, Newton R. Statistical visual language models for ink parsing. Proceedings of AAAI Spring Symposium on Sketch Understanding 2002:126–32.

[10] Rubine D. Specifying gestures by example. Computer Graphics 1991;25(4):329–37.

[11] Alvarado C, Davis R. SketchREAD: a multidomain sketch recognition engine. Proceeding of the 17th annual ACM symposium on user interface software and technology (UIST'04) 2004:23–32.

[12] Kara LB, Stahovich TF. Hierarchical parsing and recognition of hand-sketched diagrams. Proceedings of 17th annual ACM symposium on user interface software and technology (UIST'04) 2004:13–22.

[13] Kara LB, Gennari L, Stahovich TF. A sketch-based interface for the design and analysis of simple vibratory mechanical systems. Proceedings of 30th ASME design automation conference (DAC'04) 2004.

[14] Kara LB, Stahovich TF. An image-based, trainable symbol recognizer for hand-drawn sketches. Computer Graphics, in press, doi:10.1016/j.cag.2005.05.004.

[15] Costagliola G, Deufemia V. Visual language editors based on LR parsing techniques. Proceedings of 8th international workshop on parsing technologies 2003:78–90.

[16] Saund E, Mahoney J, Fleet D, Larner D, Lank E. Perceptual organization as a foundation for intelligent sketch editing. Proceedings of 2002 AAAI spring symposium on sketch understanding 2002:118–25.

[17] Hong JI, Landay JA. SATIN: a toolkit for informal ink-based applications. Proceedings of the 13th ACM symposium on user interfaces software and technology (UIST'00) 2000:63–72.

[18] Lee S. Recognizing hand-drawn electrical circuit symbols with attributed graph matching. In: Baird HS, Bunke H, Yamamoto K, editors. Structured document image analysis. New York: Springer; 1992. p. 340–58.

[19] Stahovich TF. Segmentation of pen strokes using pen speed. Proceedings of 2004 AAAI fall symposium on making pen-based interaction intelligent and natural 2004:152–8.

[20] Domingos P, Pazzani M. Beyond independence: conditions for the optimality of the simple Bayesian classifier. Proceedings of 13th international conference on machine learning 1996:105–12.