EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2007)
M. van de Panne, E. Saund (Editors)

Kirchhoff's Pen: A Pen-based Circuit Analysis Tutor

Ruwanee de Silva,[1] David Tyler Bischel,[1] WeeSan Lee,[2] Eric J. Peterson,[1] Robert C. Calfee,[3] and Thomas F. Stahovich[1]

[1]Mechanical Engineering Department, University of California, Riverside
[2]Computer Science Department, University of California, Riverside
[3]Graduate School of Education, University of California, Riverside

Abstract

Kirchhoff's Pen is a pen-based tutoring system that teaches students to apply Kirchhoff's voltage law (KVL) and current law (KCL). To use the system, the student sketches a circuit schematic and annotates it to indicate component labels, mesh currents, and nodal voltages. The student then selects either mesh (KVL) or nodal (KCL) analysis, and writes the appropriate equations. The system interprets the equations, compares them to the correct equations (which are automatically derived from the circuit), and provides tutorial feedback about errors. Unlike traditional tutoring systems that work from input provided with a keyboard and mouse, our system works from ambiguous, hand-drawn input. The goal of our work is to create computational techniques to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil. Kirchhoff's Pen is an important first step toward this goal.

1. Introduction

Pen-based interaction is becoming increasingly important, due in part to the ready availability of pen-based hardware. Despite its potential, pen-based technology has not yet been widely applied to education. Perhaps the best current example of pen-based, educational technology is Classroom Presenter [AMS05], a classroom interaction system that allows students and instructors to communicate wirelessly in lecture environments using tablet computers. However, this system does not interpret what is written, nor is it intended to provide any instructional feedback.

Our work is focused on the use of pen-based technology for creating intelligent tutoring systems. In particular, our goal is to create computational techniques to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil. This goal is consistent with recent research comparing student performance across different user interfaces showing that "as the interfaces departed more from familiar work practice..., students would experience greater cognitive load such that performance would deteriorate in speed, attentional focus, meta-cognitive control, correctness of problem solutions, and memory" [OAC06]. While that work used systems that provided no problem-solving assistance (i.e., they were not tutoring systems), the findings provide compelling evidence of the potential benefits of well-designed, pen-based instructional tools.

As one step toward our goal, we have developed Kirchhoff's Pen, a pen-based tutoring system that teaches students to apply Kirchhoff's voltage and current laws. Kirchhoff's voltage law (KVL) states that the sum of the voltages around any closed loop, or "mesh," is zero. Kirchhoff's current law (KCL) states that the sum of the currents into an electrical node is zero. Our work to date has been primarily focused on the issues of interpreting ambiguous, hand-drawn input in an instructional setting.

To use Kirchhoff's Pen, the student sketches a circuit schematic and annotates it to indicate the component labels, mesh currents, and nodal voltages. A radio button at the top of the window allows the student to switch between circuit drawing and annotation mode. Next, the student selects either mesh analysis (KVL) or nodal analysis (KCL), and writes the appropriate equations in a window at the bottom of the screen. The system interprets the equations, compares them to the correct equations (which are automatically derived from the circuit), and provides feedback about errors. Figure 1 shows an example of the system being used for mesh analysis. The equation at the bottom of the screen

Sketch-Based Interfaces and Modeling 2007, Riverside, CA, August 02-03, 2007.
© 2007 ACM 978-1-59593-913-5/07/0008 $5.00

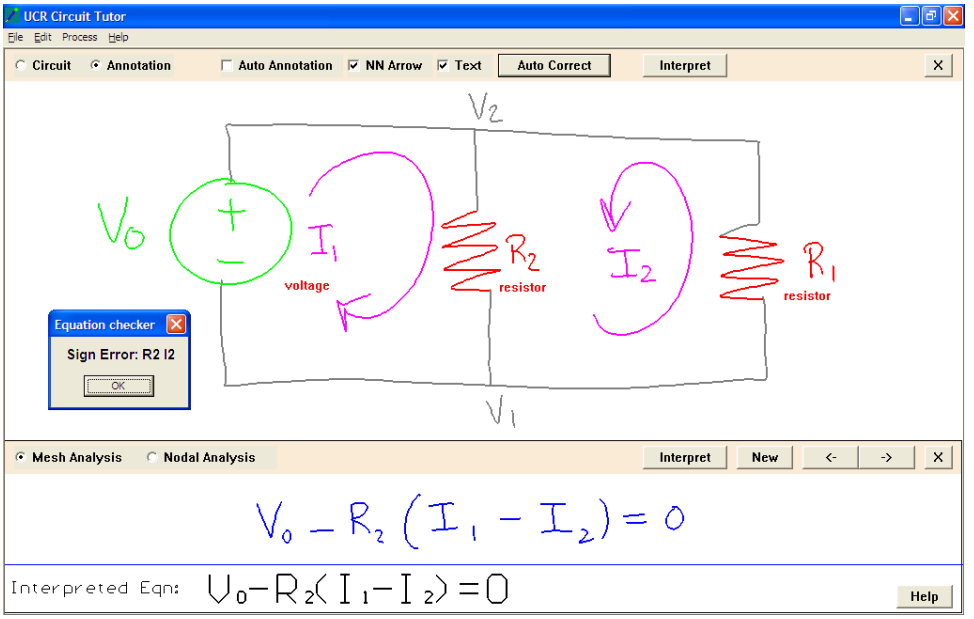

Figure 1: *Kirchhoff's Pen used for mesh analysis. The system informs the student of the sign error on the "R_2I_2" term.*

is intended to describe the mesh on the left side of the circuit. However, the student has made a sign error: the R_2I_2 term should be negative, but was written as positive. The system identifies the error and informs the student.

Kirchhoff's Pen is built on top of our AC-SPARC system [GKSS05], which can interpret a sketch of a circuit, and generate an input file for the SPICE circuit simulator [spi]. AC-SPARC is an analysis rather than instructional tool; it provides no tutoring capabilities. Additionally, AC-SPARC interprets only circuit sketches and cannot interpret circuit annotations or hand-written equations.

The next section presents a discussion of related work. This is followed by a brief discussion of AC-SPARC, and then the details of the Kirchhoff's Pen system. Finally, future work is discussed and conclusions are presented.

2. Related Work

Intelligent tutoring systems have been developed for a wide variety of domains, such as: medicine [SH04], law [Spa93], computer programming [FAR84], physics [VLS*05], and electric circuits [BRAH04, BDM06]. Nearly all of these systems are based on WIMP (windows, icons, mouse, and pointer) or keyboard interfaces. Our work, by contrast, is focused on building pedagogically-sound, pen-based interfaces for tutoring systems. While conventional tutoring systems work from unambiguous input provided with a keyboard and mouse, we focus on the challenges of working from ambiguous, hand-drawn input.

Research on pen-based interfaces is quite active at present. Examples of existing experimental applications include: a tool for simulating simple hand-drawn mechanical devices [AD01], a tool for sketching user interfaces [LM01], a UML diagram tool [HD02], a tool for interpreting hand-drawn equations [Mat99], and a tool for understanding military tactics [FFU01]. Likewise there has been significant progress in sketching 3D shapes [ZHH96].

While sketch understanding techniques have been used for a wide range of applications, the impact on tutoring systems has been limited. [AMS05] describes a classroom interaction system that allows students and instructors to communicate wirelessly in lecture environments using tablet computers. However, this system does not interpret what is written, nor is it intended to provide tutoring capabilities.

Researchers have recently begun to explore issues related to the development of pen-based instructional tools. For example, Oviatt [OAC06] compared student performance using paper-and-pencil, the Anoto digital pen system (a digital pen that digitizes ink but does not run applications) [ano], and the Tablet PC. Students used these platforms solely as a recording medium for problem solving; no tutoring capabilities were provided. This work demonstrated that "as the interfaces departed more from familiar work practice..., students would experience greater cognitive load such that performance would deteriorate in speed, attentional focus, meta-cognitive control, correctness of problem solutions, and memory." This work speaks to the importance of good user interface design in creating effective educational systems. [AYK05] describes a study of methods for entering mathematical equations into a computer. The goal was to explore interface issues for tutoring systems. Although the user input was not interpreted, the study suggests that pen-input of equations is substantially more efficient than keyboard entry, and is greatly preferred by users.

3. AC-SPARC Overview

Kirchhoff's Pen is built on top of our AC-SPARC system [GKSS05], which can interpret a sketch of a circuit and generate an input file for the SPICE circuit simulator [spi]. AC-SPARC is an analysis rather than instructional tool. It is concerned with interpreting circuit schematics; It does not interpret circuit annotations or hand-written equations, nor does it provide tutoring capabilities.

AC-SPARC was designed to provide a natural drawing experience, by placing minimal constraints on the way the user draws. The user can draw an electrical symbol with any number of pen 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 pen stroke, without lifting the stylus or pausing. The only constraint is that the user must finish drawing one symbol before starting another.

The system employs a novel sketch segmentation technique. Segmentation is the process of decomposing a sketch into the constituent objects, in this case, electrical components and wires. AC-SPARC's segmenter locates circuit components by locating regions with high "ink density" – regions with a high concentration of pen strokes. It also locates circuit components by identifying changes in the character of the pen strokes, such as when a sequence of long line segments is followed by a sequence of short arc segments. Pen strokes are segmented into lines and arcs using the technique in [Sta04].

Once it has located the electrical components, AC-SPARC classifies them using a feature-based symbol recognizer. A symbol is characterized by the number of: pen strokes, line segments, arc segments, endpoint ("L") intersections, endpoint-to-midpoint ("T") intersections, midpoint ("X") intersections, pairs of parallel line segments, and pairs of perpendicular line segments. The final feature is the average distance between the endpoints of the segments, normalized by the maximum distance between any two endpoints. A symbol definition is learned from training examples by computing the mean and standard deviation of each feature from those examples. During classification, the features of the unknown symbol are compared to the distributions of the features in the definition, using a naive Bayesian approach.

AC-SPARC uses context to automatically correct interpretation errors. Domain knowledge is used to determine if the interpretation of the sketch is self-consistent. If it is not, segmentation and recognition are revisited so as to eliminate the inconsistencies. For example, if a component is recognized as a capacitor, but has only one wire attached to it, the program will revisit recognition to determine if another likely classification would make more sense. For instance, if the next highest ranked classification was that of a ground symbol, the program would reclassify the symbol as such, because that would match the number of attached wires.

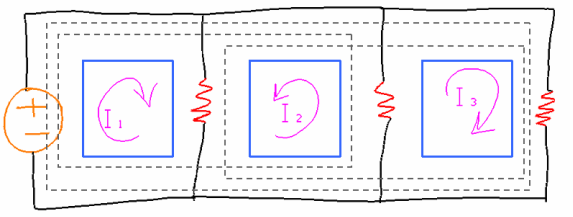

Figure 2: *Circuit with six meshes indicated by approximately inscribed rectangles (3 dashed, 3 solid). The three solid rectangles indicate "windows," meshes that are preferred for analysis.*

4. System Design

To interpret the student's work and provide tutorial feedback, Kirchhoff's Pen must perform a number of tasks. It must interpret the various circuit annotations, such as component labels and mesh-current arrows, and associate them with the objects they describe. It must derive the correct equations for the circuit. (Our underlying AC-SPARC system, described in Section 3, interprets the circuit.) Finally, it must interpret the student's hand-written equations, and compare them to those it derives. The following sections describe how the system performs these tasks.

4.1. Locating Meshes

To verify the student's mesh analysis, Kirchhoff's Pen must be able to compute the correct mesh equations for the circuit. To do this, the program must first identify all of the possible meshes in the circuit. A mesh is a closed, non-self-intersecting path through the circuit. The program locates meshes by searching a graph representation of the circuit produced by AC-SPARC. Each graph node represents a wire, or an electrical component, such as a resistor. The edges in the graph represent electrical connections between the wires and components. To begin, the program selects a graph node, and exhaustively searches for non-self-intersecting paths that return to that node. All such paths are meshes. Another graph node is then selected as the starting point for another search. First, however, the previous starting node is removed from the graph to prevent duplicate paths from being discovered. This process is repeated until all graph nodes have been considered, and thus all meshes located.

A typical circuit will have multiple meshes, but not all are needed to analyze the circuit. In general, the number of meshes required is given by:

$$M = C - N + 1 \tag{1}$$

where M is the number of meshes, N is the number of electrical nodes (wires with distinct voltages), and C is the number of components (resistors, voltage sources, and current sources) [AS04]. For example, the circuit in Figure 2 has

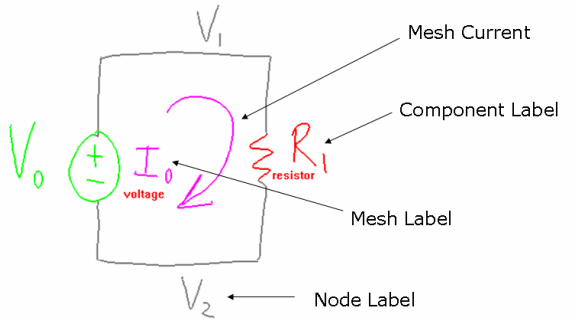

Figure 3: *Typical circuit annotations.*

six different meshes. However, because the circuit has four components and two electrical nodes (the top and bottom "rails"), only three meshes are required to analyze the circuit. Any three of the six meshes could be used, but students are initially taught to choose "windows," meshes that do not contain portions of other meshes. In Figure 2, the windows are indicated with solid rectangles. The program identifies the windows by sorting the meshes in increasing order of bounding box size, and selecting the M smallest meshes that do not subsume other meshes with smaller bounding boxes.

4.2. Interpreting Circuit Annotations

After the student has drawn a circuit, he or she must annotate it to indicate the mesh currents, component labels, and nodal voltages. Figure 3 shows a typical set of annotations. To interpret these, the program must first segment them into distinct objects. The annotations are drawn in annotation mode, rather than circuit drawing mode, and thus are distinct from the circuit sketch. Consequently, the program needs only to distinguish the annotations from one another. It does this by first identifying the arrows representing mesh currents. All of the remaining annotations are text, which are grouped into individual labels using a clustering technique. The labels are then interpreted, and mapped to the corresponding circuit elements and mesh current arrows. The following sections describe these steps in more detail.

4.2.1. Recognizing Arrows

To recognize mesh-current arrows, we use a technique described in [Kar04]. Our implementation is suitable for single-stroke arrows drawn from tail to tip. The technique can also be used for two-stroke arrows, but we have not yet implemented this.

To begin, the pen stroke is resampled to produce 36 evenly spaced points. A line segment is then defined between each pair of consecutive points. Finally, the cosine of the angle between adjacent segments is computed, as shown in Figure 4. The cosine is inversely related to the curvature. For example, if two consecutive segments are nearly colinear, the cosine is close to 1.0. If there is a large discontinuity,

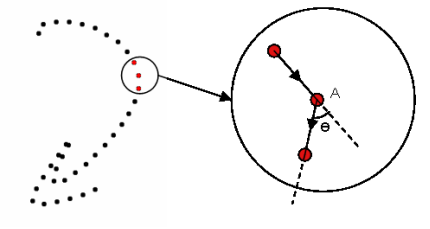

Figure 4: *Resampled arrow. Inverse curvature at point A is* $cos(\theta)$.

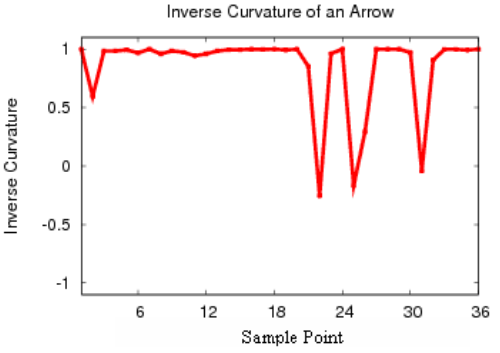

Figure 5: *Inverse curvature of the arrow from Figure 4.*

such as a 90^o bend, the cosine is close to 0.0. For this reason, the cosine of the angle between adjacent segments is called "inverse curvature." Figure 5 shows the inverse curvature representation of the arrow from Figure 4. Notice that the inverse curvature is approximately 1.0 for most points on the arrow, but is much smaller (in this case, less than 0.0) for the three discontinuities at the head of the arrow. It is these discontinuities that enable the technique to identify arrows. The shape of the arrow shaft is irrelevant. In fact, the first 18 sample points are actually discarded by the recognizer.

The arrow recognizer is a neural network comprised of an eighteen-node input layer, two five-node hidden layers, and a single-node output layer. The inputs to the network are the 18 inverse curvature values for the head end of the arrow. The output is the classification: arrow or non-arrow. We trained the network using 300 arrows and 1900 non-arrows provided by six different users. The non-arrows consisted of examples of all legal, non-arrow annotations, including "V," "I,", "*R*," and all single digit numbers. Using such a comprehensive set of negative training examples contributes to the robustness of the recognizer.

4.2.2. Clustering Text Labels

The arrow recognizer described in the previous section is applied to each annotation pen stroke. Any stroke that is classified as a non-arrow is considered text. Before the text can be recognized, however, the non-arrow pen strokes must be clustered into individual characters, which are then clustered

into distinct text labels. A character is a group of pen strokes that intersect, or whose separation from one another is less than a threshold. Similarly, a text label is a set of characters whose separation from one another is less than a threshold. The characters in a cluster are sorted from left to right and are recognized with the Tablet PC handwriting recognizer. Once the labels have been recognized, the hand-written characters are replaced with machine generated characters. The complete process is described in Section 4.3.

4.2.3. Associating Labels

After the text labels have been located and recognized, they must be associated with the objects – circuit components, electrical nodes (wires), and mesh-current arrows – they describe. We use geometric proximity to determine which object is associated with each label. However, this problem presents several challenges. First, it is common for a given label to be near several different objects, and furthermore, it is possible that the closest object is not the intended association. Label I_0 in Figure 3, for example, is intended to describe the mesh current, but is actually closer to the voltage source than it is to the arrow. Second, a label intended for a particular circuit component may be as close or closer to the wire attached to that component. Third, in most problems, some objects are not intended to have labels, even if there are nearby labels. For example, in mesh analysis the electrical nodes are not labeled.

To address these issues, we define a cost function, based on the sum of the distances from the labels to their associated objects:

$$C(a) = \sum_{i=1}^{N} d(L_i, a(L_i)) + P(a(L_i)) \qquad (2)$$

Here L_i is the i^{th} label, $a(L_i)$ is the object associated with L_i, $d(L,O)$ is the Euclidean distance from label L to object O, and N is the total number of labels. $P(O)$ is a penalty function that causes the system to preferentially apply labels to electrical components and mesh-current arrows, rather than wires. $P(O)$ is 200 pixels (about 20% of the horizontal resolution of the tablet) if object O is a wire, and is zero otherwise.

The best set of label associations is the one that minimizes the cost in Equation 2:

$$BestAssociations = \underset{a \in A}{argmin}\, C(a) \qquad (3)$$

where A is the set of all mappings of labels to objects. We find this best set using a simulated annealing approach. Initially, a greedy approach is used to assign each label to the nearest un-assigned object. Then, in each iteration of simulated annealing, two labels are randomly selected, and their associations swapped. The cost of the new assignments is computed using Equation 2. If the new cost is less than the previous value, the swap is kept. If the cost increases, there is still a probability that the swap will be kept. The probability is inversely proportional to the amount of increase of the cost, and the number of iterations that have already occurred. The process terminates after 100 successive iterations with no improvement, or a total of 1000 iterations. The final associations are revealed via color coding as shown in Figure 1.

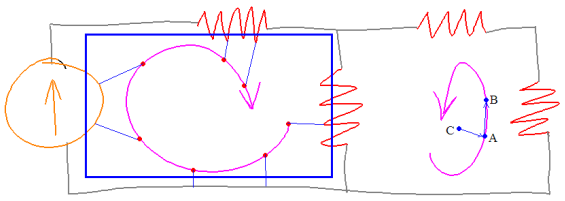

Figure 6: *Left: arrow mapped to the mesh indicated by the rectangle. Right: vectors used for computing orientation of mesh-current arrow.*

In our studies, we have not required users to provide training data for the circuit component classifier, but have instead relied on default training data. This has caused some users to experience some classification errors. As a remedy, Kirchhoff's Pen has a mode in which the labels are used to automatically correct errors in the classification of the components. By convention, resistor labels begin with "R," voltage source labels begin with "V", and current source labels begin with "I". If the classification of a component is inconsistent with its label, the program changes the classification accordingly. For example, if label "I_1" is associated with a voltage source, it is reclassified as a current source.

4.2.4. Associating Arrows

Once the mesh-current arrows have been identified, they must be associated with the appropriate meshes. Here, again we make use of geometric proximity. The arrows are resampled by keeping only every fourth data point. The distance from arrow W to mesh M is computed by finding, for each sample point w in W, the closest point m on an electrical component or wire in M:

$$D(W,M) = \sum_{w \in W} \underset{m \in M}{min} \|w - m\| \qquad (4)$$

Figure 6 shows an example in which each sample point on the arrow is connected to its nearest point on the components and wires of the mesh on the left side of the circuit. To find the mesh associated with a particular arrow, Equation 4 is used to compute the distance from that arrow to each mesh. The arrow is associated with the mesh that produces the smallest distance.

The final step in associating an arrow with a mesh is determining the orientation – clockwise or counterclockwise – of the arrow. The orientation, which is needed to determine the sign of each term in a mesh equation, is computed using a cross product. Consider the arrow on the right side of Figure 6. Two nearby points, A and B, are selected on the shaft of the arrow such that A is closer to the tail than B. A

point C is then selected at the center of the bounding box of the arrow. If $(\overrightarrow{CA} \times \overrightarrow{AB}) \cdot \hat{k} > 0.0$, the arrow is counter-clockwise, otherwise it is clockwise. ($\hat{k}$ is a unit vector in the z-direction, i.e., out of the screen.)

4.3. Interpreting Equations and Labels

Kirchhoff's Pen uses the Tablet PC handwriting recognizer to interpret the characters in text labels and equations. This recognizer is intended primarily for cursive writing rather than block characters. Furthermore, the recognizer uses a dictionary to improve recognition accuracy for words. This actually hinders accuracy in our application, because our labels and equations are not contained in the dictionary. As a remedy, our system segments the text into individual characters, and passes them to the handwriting recognizer one at a time. To improve accuracy, our system then uses a set of domain-specific debugging rules to correct common recognition errors. Details of these steps are described in the sections that follow.

4.3.1. Clustering Characters

The program uses geometric proximity to cluster pen strokes into individual characters. Characters are located by grouping pen strokes that intersect, or whose separation from one another is less than a threshold. Because the two lines in an equal sign are separated by a distance comparable to the typical inter-character spacing, equal signs are handled as a special case: Two horizontal lines located one above the other, and separated by a distance that is a fraction of their length, are clustered into a single character. The clustered characters are sorted from left to right. This enables users to write the terms of an equation in any order. Once the characters are sorted, subscripts are identified as characters that are less than two-thirds the height of the previous character.

4.3.2. Recognizing Characters

Most characters are recognized using the Tablet PC handwriting recognizer. However, we developed special-case recognizers for the equal sign, plus sign, capital "I" with bars on the top and bottom, and the forward slash ("/"), because these are frequently misclassified. If a character is recognized by one of these special-purpose recognizers, it is not sent to the Tablet PC handwriting recognizer.

The Tablet PC handwriting recognizer produces a ranked set of alternative interpretations for each character. We select the highest-ranked choice that is consistent with our problem domain. Specifically, for both text labels and equations, the allowed characters consist of "V", "I", and "R" and the digits "0" – "9." Equations can also contain: "(", ")", "[", "]", "+", "−", "/", and "=".

4.3.3. Debugging Rules

By using our special-purpose character recognizers, and biasing the handwriting recognizer to the legal characters, our program does avoid many recognition errors. However, there are still several kinds of common misclassifications. For example, it is common for "I", "1", and "/" to be confused with one another. We correct these sorts of errors by examining the local context of each character using the rules shown in Table 1. For example, subscripts must be digits. Thus, if a subscript is interpreted as an "I", it is changed to a "1". Conversely, if a "1" has a subscript, the "1" is changed to "I". This set of rules has proven to substantially improve recognition accuracy.

Equation Type	Rule	Example
All	$\square_I \longrightarrow \square_1$	$R_I \longrightarrow R_1$
	$1\square \longrightarrow I\square$	$1_3 \longrightarrow I_3$
	$\square_- \longrightarrow \square-$	$)_- \longrightarrow)-$
	$\square_\square 4 \longrightarrow \square_\square +$	$R_2 4 \longrightarrow R_2 +$
	$12 \longrightarrow R$	$12 \longrightarrow R$
Mesh	$/\square \longrightarrow I\square$	$/_1 \longrightarrow I_1$
	$\square_/ \longrightarrow \square_1$	$I_/ \longrightarrow I_1$
Nodal	$1V \longrightarrow (V$	$1V \longrightarrow (V$
	$IV \longrightarrow (V$	$IV \longrightarrow (V$
	$)1R \longrightarrow)/R$	$)1R \longrightarrow)/R$
	$)IR \longrightarrow)/R$	$)IR \longrightarrow)/R$

Table 1: *Debugging rules for improving recognition accuracy for text. Rules for "All" apply to text labels, and both mesh and nodal equations. "□" represents any character.*

4.4. Critiquing Equations

To critique the student's equations, Kirchhoff's Pen first derives the correct equations for the circuit. The mesh equations are derived from the meshes identified by the methods described in Section 4.1. The nodal equations are derived from a graph representation of the circuit produced by the underlying AC-SPARC system.

To facilitate comparing the student's equations to the correct ones, we represent each equation with a matrix containing the coefficients of the various terms in the equation. Here, we will describe how the process works for mesh equations; nodal equations are handled in an analogous fashion.

The terms in a valid mesh equation consist of either a single variable, or a product of two variables. In particular, a valid mesh equation will contain terms representing voltage sources, such as "V_0," and products of currents and resistances, such as "$R_1 I_1$." An equation can be represented by a matrix in which there is a row and a column corresponding to each variable. So that a single variable can be treated as product of two variables, an extra row and column, corresponding to a value of "1", are added to the matrix. "V_0" is treated as the product of "1" and "V_0," for example. The values stored in the matrix represent the coefficients of each possible term, i.e., each possible product of two variables. A "1" in the matrix indicates that the term exists in the equation. A "-1" indicates that the term exists, and has a negative sign. A "0" indicates that the term does not exist in the equation.

To illustrate the approach, consider the mesh equation in

Equation 5, which is represented by the matrix in Table 2. The term "V_0" is represented by the "1" in row 5, column 7 (and row 7, column 5). The term "$-I_0R_1$" is represented by the "-1" in row 1, column 4 (and row 4, column 1). The term "$-I_0R_0$" is represented by the "-1" in row 1, column 3 (and row 3, column 1). As these are the only three terms in the equation, all other matrix entries are "0".

$$V_0 - I_0R_1 - I_0R_0 = 0 \tag{5}$$

	I_0	I_1	R_0	R_1	V_0	V_1	1
I_0	0	0	-1	-1	0	0	0
I_1	0	0	0	0	0	0	0
R_0	-1	0	0	0	0	0	0
R_1	-1	0	0	0	0	0	0
V_0	0	0	0	0	0	0	1
V_1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0

Table 2: *Matrix representation of Equation 5*

To simplify the implementation, we actually use a 31×31 matrix, with the column and row labels being $I_0, I_1, ..., I_9, R_0, R_1, ..., R_9, V_0, V_1, ..., V_9$, and 1. Because subscripts are limited to a single digit, this will represent all possible, legal equations. The matrix is symmetric, so only the upper triangle (or lower triangle) of the matrix need be considered.

Kirchhoff's Pen uses a simple numerical technique to derive the matrix representation for an equation. To determine the coefficient for a term representing the product of two particular variables, the program sets those variables equal to one, sets all other variables equal to zero, and numerically evaluates the left side of the equation. For example, to determine the coefficient of the term "I_0R_1" in Equation 5, the program sets $I_0 = R_1 = 1$ and $V_0 = R_0 = 0$ as shown in Equation 6. This correctly identifies the coefficient as negative one:

$$0 - 1 * 1 - 1 * 0 \tag{6}$$

The student is not required to indicate which mesh a particular equation is intended to represent. We assume the equation was intended to represent whichever mesh it describes best. More precisely, the intended mesh is identified by comparing the matrix form of the student's equation to the matrix form of each of the correct equations. The mismatch between two matrices is defined as the number of non-zero elements unique to one or the other of the matrices. Once the intended equation has been identified, the matrices for the student's equation and the correct equation are compared term by term. Errors are reported if the student's equation is missing terms, has extra terms, or has terms with the wrong sign.

4.5. Editing Gestures

Kirchhoff's Pen provides several editing gestures to allow the user to change the input or correct interpretation errors. Ink can be erased with the eraser end of the stylus. The circuit can be extended at any time by simply drawing new wires and components. Recognition errors can be corrected by tapping the stylus on the circuit symbol or text in question, and selecting the correct interpretation from a pop-up list. If the circuit segmenter fails to locate a circuit component, it can be manually located by holding the button on the stylus and circling it. If the segmenter incorrectly marks ink as a circuit component, this can be corrected by holding the button on the stylus and making a slashing gesture through the ink.

5. Discussion and Future Work

We have not yet conducted a formal user study of Kirchhoff's Pen. We have, however, obtained informal feedback from a number of subjects who have tried the system. The feedback has been generally favorable, but has revealed opportunities for improving the system. In particular, we plan to increase the naturalness of interaction by eliminating some of the assumptions about how objects are drawn. For example, we currently assume that arrows are drawn with single-strokes, but some users prefer to draw them with two-strokes. Fortunately, implementing a two-stroke arrow recognizer is a straightforward extension of our current system. Also, our text clustering technique requires the user to maintain a minimum inter-character spacing. Users can adapt to this relatively easily, but we plan to develop a more robust approach.

Currently, the circuit is drawn in circuit mode, and is annotated in annotation mode. We used this approach for our initial prototype system, because it allowed us to efficiently build tutoring capabilities on top of our existing AC-SPARC system. In future work, we will eliminate the modes, and develop techniques to enable the system to distinguish annotations from the circuit. This will be feasible, as the circuit is comprised, for the most part, of connected pen strokes.

The system currently teaches students to apply Kirchhoff's current and voltage laws. We plan to extend the system to other topics, such as simplifying circuits by identifying parallel and series components, and transforming sources, i.e., computing Norton and Thevenin equivalents. Additionally, we plan to greatly expand the kind of tutorial feedback the system provides. For example, if the "I_2R_2" term is missing from the equation in Figure 1, the program simply reports that the term is absent. A better explanation would also indicate the likely source of the error, drawing the student's attention to the fact that resistor R_2 is part of two meshes. We will provide this sort of explanation using the "buggy rules" approach to intelligent tutoring [FAR84].

Once we address the remaining user interface issues and complete the tutorial capabilities, we plan to deploy Kirchhoff's Pen in an introductory electric circuits course at the University of California, Riverside. We plan to conduct a formal assessment of the usability of the system, and its value as an instructional tool.

6. Conclusion

We have presented Kirchhoff's Pen, a pen-based tutoring system that teaches students to apply Kirchhoff's voltage law (KVL) and current law (KCL). To use the system, the student sketches a circuit schematic and annotates it to indicate component labels, mesh currents, and nodal voltages. The student then selects either mesh analysis (KVL) or nodal analysis (Kirchhoff's KCL), and writes the appropriate equations. The system interprets the equations, compares them to the correct equations (which are automatically derived from the circuit), and provides tutorial feedback about errors.

Intelligent tutoring systems have been widely studied, and applied to a variety of subjects. However, most current systems work from unambiguous input provided with a keyboard and mouse. Our work has addressed some of the challenges of working from ambiguous, hand-drawn input.

Our work is motivated by research suggesting that skill transfer is higher when training and testing environments are similar. Our work is also supported by recent research suggesting the potential benefits of well-designed, pen-based instructional tools. For these reasons, our goal is the creation of computational techniques to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil. Kirchhoff's Pen is a prototype system, and there is clearly much more work to be done. Nevertheless, this system is an important first step toward our goal.

7. Acknowledgments

The authors are grateful to Microsoft Research for their support for this work.

References

[AD01] ALVARADO C., DAVIS R.: Resolving ambiguities to create a natural sketch based interface. In *IJCAI'01* (2001), pp. 1365–1371.

[AMS05] ANDERSON R., MCDOWELL L., SIMON B.: Use of classroom presenter in engineering courses. In *FIE'05* (2005), pp. T2G–13–18.

[ano] Anoto Group AB. http://www.anoto.com/.

[AS04] ALEXANDER C., SADIKU M.: *Fundamentals of Electric Circuits*. McGraw-Hill Science/Engineering/Math, 2004.

[AYK05] ANTHONY L., YANG J., KOEDINGER K. R.: Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI '05* (2005).

[BDM06] BUTZ B. P., DUARTE M., MILLER S. M.: An intelligent tutoring system for circuit analysis. In *IEEE Transactions on Education* (2006), vol. 49, pp. 216–223.

[BRAH04] BILLINGSLEY W., ROBINSON P., ASHDOWN M., HANSON C.: Intelligent tutoring & supervised problem solving in the browser. In *ICWI* (2004), pp. 806–810.

[FAR84] FARRELL R. G., ANDERSON J. R., REISER B. J.: An interactive computer-based tutor for LISP. In *AAAI'04* (1984), pp. 106–109.

[FFU01] FORBUS K. D., FERGUSON R. W., USHER J. M.: Towards a computational model of sketching. In *6th International Conference on Intelligent User Interfaces* (2001), pp. 77–83.

[GKSS05] GENNARI L., KARA L. B., STAHOVICH T. F., SHIMADA K.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics 29*, 4 (2005), 547–562.

[HD02] HAMMOND T., DAVIS R.: Tahuti: A geometrical sketch recognition system for UML class diagrams. In *AAAI Spring Symposium on Sketch Understanding* (2002), pp. 59–68.

[Kar04] KARA L. B.: *Automatic Parsing and Recognition of Hand-Drawn Sketches for Pen-Based Computer Interfaces*. PhD thesis, Department of Mechanical Engineering, Carnegie Mellon University, September 2004.

[LM01] LANDAY J. A., MYERS B. A.: Sketching interfaces: Toward more human interface design. *IEEE Computer 34*, 3 (2001).

[Mat99] MATSAKIS N.: *Recognition of Handwritten Mathematical Expressions*. Master's thesis, MIT, Cambridge, MA, 1999.

[OAC06] OVIATT S., ARTHUR A., COHEN J.: Quiet interfaces that help students think. In *UIST '06* (2006), pp. 191–200.

[SH04] SUEBNUKARN S., HADDAWY P.: A collaborative intelligent tutoring system for medical problem-based learning. In *IUI '04* (2004), pp. 14–21.

[Spa93] SPAN G.: LITES, an intelligent tutoring system for legal problem solving in the domain of dutch civil law. In *Proc. 4th International Conference on AI and Law* (1993), pp. 76–81.

[spi] The spice page. http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

[Sta04] STAHOVICH T. F.: Segmentation of pen strokes using pen speed. In *AAAI Symposium, Making Pen-Based Interaction Intelligent and Natural* (2004).

[VLS*05] VANLEHN K., LYNCH C., SCHULZE K., SHAPIRO J. A., SHELBY R., TAYLOR L., TREACY D., WEINSTEIN A., WINTERSGILL M.: The Andes physics tutoring system: Lessons learned. *International Journal of AI in Education 15*, 3 (2005).

[ZHH96] ZELEZNIK R., HERNDON K., HUGHES J.: SKETCH: An interface for sketching 3D scenes. In *SIGGRAPH '96* (1996), pp. 163–170.