Text Segmentation and Grouping for Tactile Graphics

Matthew Renzelmann*

June 1, 2005

Abstract

The goal of our research is to develop computer software capable of automating the conversion of figures and diagrams in a book to a tactile format readily understandable by blind persons. Among the challenges this goal poses is the need to find and remove text from the figures and diagrams for eventual conversion to Braille. Since conventional optical character recognition (OCR) software is poorly suited to the task of finding short pieces of text embedded within images, we are exploring new methods based on machine learning. By exploiting the consistent style of figures and diagrams in a book, we can improve text recognition accuracy over that provided by OCR. The algorithm described herein is able to recognize a high percentage of the text once provided with a small training set from which to base its predictions. We discuss the specifics of the algorithm, its effectiveness, and its relationship with existing commercial OCR software.

*Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195. Email: mrenz@cs.washington.edu. This research is supported by the Mary Gates Endowment at the University of Washington, and by NSF Grant No. IIS-0415273.

Contents

List of Figures

List of Tables

1 Introduction

John is an engineering undergraduate at a large university. He attends class regularly, does his homework diligently, and meets with his friends in his spare time. It takes John significantly more effort to interpret all the schematics, figures, and diagrams in his textbooks than it does most students. Indeed, before examining them himself, he must take the most important ones to the university disabled student services office for translation to a tactile format.

You see, John is blind. In order for him to perceive an image, he must be able to feel it. Since literature in nearly every field is designed with the intention of being perceived by the eyes, it poses John and people like him with a significant challenge—a challenge we are working to overcome by combining new and existing technology.

Before delving into our solution to this problem, we will first briefly examine the state of the art: how people currently convert visual graphics into tactile graphics—graphics that can be felt.

1.1 Current Practices

At present, hundreds of people are employed across the country who, often on a part time basis, convert existing images into tactile graphics. Frequently, they work for school districts or universities to assist students like John.

The techniques employed vary widely. Some tactile graphics specialists use no modern technology whatsoever, employing instead materials such as swellpaper, aluminum foil, and thermoform [6]. Others use modern technology, like the Viewplus Tiger Embosser, which functions as a standard computer printer except it creates embossed bumps on the paper in place of dark colors. Stuart Olsen, an employee in the Access Technology Laboratory at the University of Washington, creates tactile graphics by tracing the existing image manually, inserting Braille in place of any text present, and printing the result on a Tiger Embosser. A typical image might take several hours to complete.

1.2 Future Practices

Because of the tremendous amount of manual work that goes into creating a tactile graphic using existing technology, few such images can be created. Consequently, blind people are often unable to access images used by sighted

people. We aim to improve this situation by automating the process of creating tactile graphics given existing digital images. Furthermore, instead of providing software to ease creation of a single image at a time, we intend to provide the means to convert an entire book to a tactile format. This approach represents a fundamental shift in the way tactile graphics are produced—a shift we feel is necessary for tactile graphic production to outgrow its current scope.

An outline of the proposed system is diagrammed in Figure 1. First, the tactile graphics specialist either scans a large set of scientific figures and diagrams into the computer, or retrieves digital versions of these images directly from the book's publisher. Then, an algorithm developed by Sangyun Hahn, a graduate student at the University of Washington, classifies these images according to their type (bar chart, line graph, etc). Next, using the algorithm developed in this paper, the text within these images is found and extracted. Once the text is extracted, optical character recognition software is used to convert the image of the text to an encoding such as Unicode. The computer then converts this Unicode into Braille. At the same time, the graphical parts of the image are simplified to facilitate easier tactual perception. After scaling up the original image to ensure the Braille fits, the Braille is then reintegrated with the simplified graphics to form the finished product. Additional manual work may take place should errors have occurred in this process.

1.3 Scope of This Work

Figures and diagrams found in high school and college textbooks are often highly abstract. They include schematics, graphs, diagrams, and charts—artificially generated images used to illustrate a point or clarify a concept. These kinds of images stand in contrast to photographs, for example, which consist of large numbers of continuously varying colors. In this work, we focus on images in the former category, as those in the latter require a very different approach to render successfully using technology like the Tiger Embosser.

Furthermore, because the general problem of converting complete science and engineering textbooks to a tactile format consists of so many components, this work will cover only the problem of separating text from graphics within scientific figures and diagrams. We will examine a technique based on connected components, its effectiveness, and some of its remaining problems. Because this problem is similar to that solved by existing optical character

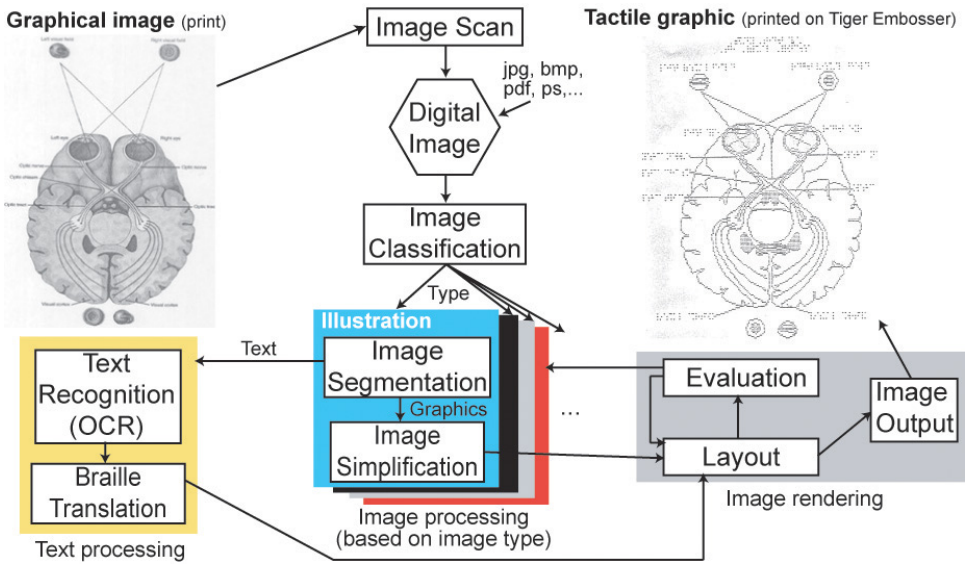

Figure 1: This diagram outlines the process we are using to automate the creation of tactile graphics [6].

recognition software, we will compare our technique with this technology.

2 Optical Character Recognition Software

With the rising popularity of document scanners, optical character recognition (OCR) technology has become increasingly pervasive. The goal of such software is to read in a scanned image which contains text and/or graphics, and convert the text to a form the computer can manipulate such as ASCII. After performing OCR on a scanned document, the user may directly edit the text within that document using a word processor.

Since our goal is to extract text from figures and diagrams and convert it to Braille, we first examine the effectiveness of OCR software at this task. Figure 2 shows an example bar chart from [3] that we would like to convert to a tactile format. To be suitable, existing software must be able to extract text from the figure without disrupting the rest of the graphic. Additionally, it must maintain the location of the extracted text to allow for appropriate reinsertion of the Braille.

Unfortunately, existing OCR software is unable to meet these demands.

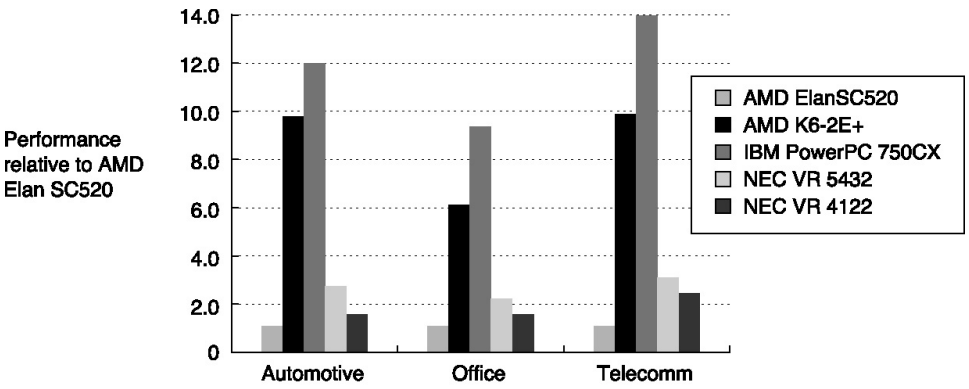

Figure 2: An example figure from a textbook.

Figures 3 and 4 show how poorly two leading OCR packages extract text from the sample image. In the first figure, the bars on the right have been truncated, and the text labels at the bottom of the image have been misaligned. The software failed to recognize the labels along the y-axis as text, thereby rendering it useless to a blind student examining a tactile version of this image in which the characters must be converted to Braille. In the second example (Figure 4), the OCR software recognized all the text in the image, but its location was incorrectly recorded. Consequently, the labels have been placed inappropriately. Furthermore, the bars have been distorted or removed, rendering this graph completely useless to both sighted and blind people. Clearly, commercially available software is insufficient for the task of extracting short text labels from scientific figures and diagrams.

Despite these shortcomings, OCR software remains highly effective at recognizing large blocks of text. Consider the example image in Figure 5. Both of the commercial OCR packages used in the previous examples are able to recognize the text within this image with perfect accuracy. The algorithm we present below takes advantage of this strength to improve text recognition accuracy in scientific figures and diagrams significantly. Instead of providing the OCR software with the original image, our algorithm finds the text, provides the OCR software with an image like that in Figure 5, and lets the OCR software determine the actual characters.

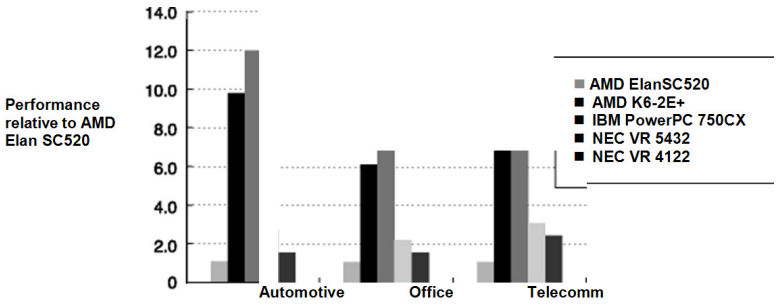

Figure 3: This image demonstrates the ineffectiveness of existing OCR software at recognizing text in images like bar charts. The text along the y-axis was not recognized, the labels along the x-axis were misaligned, and the bars themselves were inexplicably truncated.

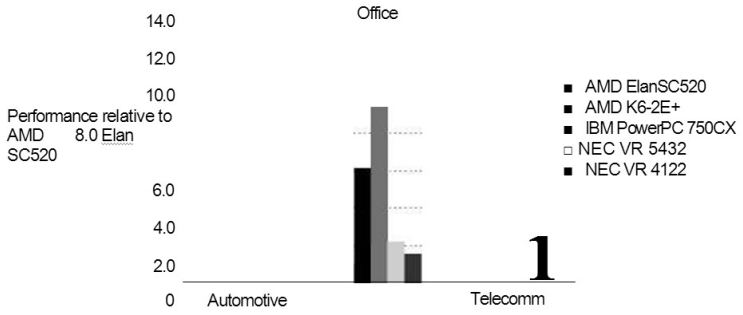

Figure 4: This image also shows how poorly modern OCR software recognizes text in figures like this bar chart. The "Office" label was placed incorrectly, many of the bars were either deleted or mysteriously converted to characters, and the labels along the y-axis were misplaced.

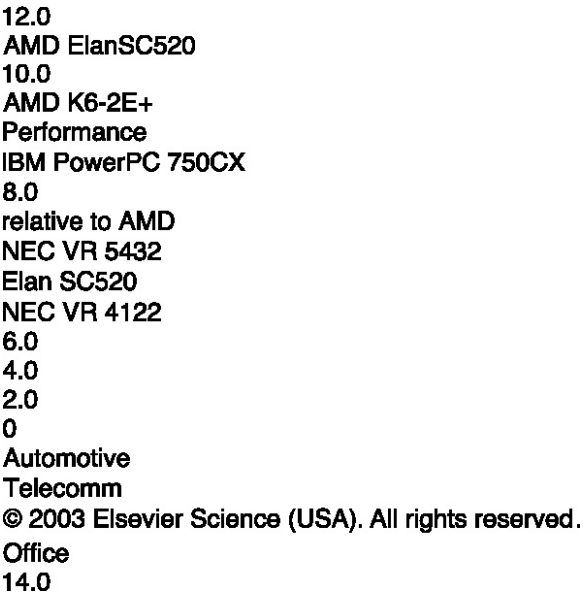

Figure 5: This figure is an image derived from the example bar graph. The text includes all the labels from the bar graph arranged in such a way as to facilitate high accuracy OCR using existing software. The algorithm presented in this paper extracts text from such figures as the bar chart and puts it in this format.

3 Finding Characters of Text

In this section, we will examine the details of the character-finding algorithm we use to find text in figures and diagrams like those seen above. The technique is based on ideas first presented in [2] and [5].

3.1 Character Training

The first step in finding text within an image is finding the individual characters. Accomplishing this task requires some understanding of the appearance of the text. Consequently, the character-finding algorithm begins with a training phase in which the user supplies examples of characters. With a font database populated with details on the training characters, the character-finding algorithm should be able to find characters in images that it has not previously encountered with high accuracy. This assertion rests on the assumption that the characters in one image are similar to those in another. Fortunately, since the editors of textbooks often go to great lengths to ensure the figures and diagrams in their textbooks are consistently formatted, this assumption is probably safe.

The character-finding algorithm treats each image as an undirected graph. Pixels are nodes, and undirected edges exist between pixels only if they are adjacent horizontally, vertically, or diagonally, and are the same color. Figure 6 illustrates this idea. Therefore, most scientific figures and diagrams consist of a large set of connected components, such as lines, x- and y-axes, and individual characters.

Once the user has specified a set of training characters, the font database is updated. This font database consists of details of all the selected connected components.

3.2 Observation

Before discussing the technique used to find the characters given a training set, we must consider an observation of the appearance of text like English. Figure 7 is an example of such text. Note the relatively uniform color of the text. The constituent characters all appear approximately the same; none of the characters stand out or are otherwise distinguished. This appearance is no accident: the designers of most typefaces endeavor to design their fonts to achieve this appearance. The character-finding algorithm presented

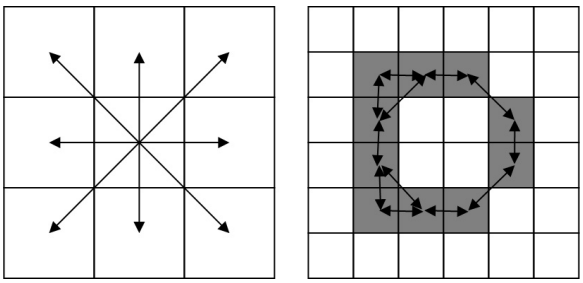

Figure 6: Definition of "connected."

in the following section exploits this uniform appearance to find characters accurately.

3.3 Character-Finding Algorithm

After the user trains the character-finding algorithm by providing it with examples of characters, it predicts which connected components in subsequent images are characters. The principle idea is to compare each "candidate" connected component in the new image with each connected component in the database. If an individual pair is sufficiently similar, then the candidate is marked as a character. If the pair is too different, the candidate may or may not be a character. Only after finding that the candidate is unlike every connected component in the database does the character-finding algorithm conclude it is not a character.

We now examine the details of this algorithm. For each connected component in the database and in the image being analyzed, the following pieces of information are maintained:

- Width
- Height
- Bounding box area
- Connected component color
- Pixel colors

Figure 8 shows an example connected component with labels indicating the width and height. The bounding box area is the connected component's

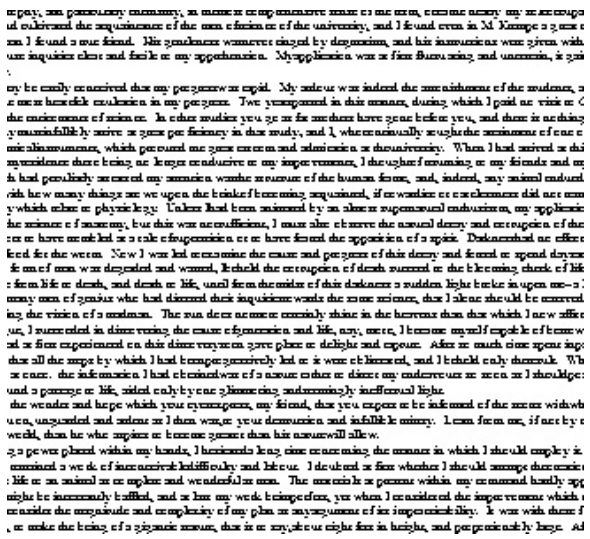

Figure 7: This image demonstrates the uniform appearance of text. None of the individual characters are significantly different from any of the others.

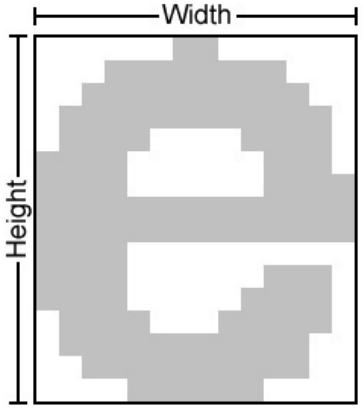

Figure 8: Width and height are two characters used to determine whether a given connected component is a character.

width multiplied by its height, and the color is simply "gray." The pixel colors being maintained will be discussed at greater length in an upcoming section. First, we will discuss how the character-finding algorithm uses this information to find characters.

The comparison between a candidate connected component and one in the font database consists of five steps. At each step, the character-finding algorithm compares a single feature of the two connected components. If the features are different enough, the test halts, and the algorithm begins comparing the candidate with the next member of the database. If the candidate passes all five of these tests, it is marked as a valid character since it is relatively similar to a member of the font database. Otherwise, once the algorithm has compared the candidate with every connected component in the font database, the candidate is rejected and not marked.

The following list describes these five tests. The three parameters T_W, T_H, and T_A are "magic numbers" used to tune performance of the character-finding algorithm. A smaller value leads to more false negatives, in which a character is incorrectly identified as a graphical element, while a larger value leads to more false positives, where a graphical element is classified as a character. W, H, A, and C represent the width, height, area, and color of the candidate connected component, while W', H', A', and C' represent the same quantities of the connected component stored in the font database.

1. If $|W - W'|/\min(W, W') > T_W$, then reject.
2. If $|H - H'|/\min(H, H') > T_H$, then reject.
3. If $|A - A'|/\min(A, A') > T_A$, then reject.
4. If $C \neq C'$, then reject.
5. If the pixel color distribution of the two connected components is sufficiently different, then reject. Since this test is more complex, it will be discussed separately.

Therefore, if the candidate passes all five of these tests, it is probably a character and is marked accordingly. Similarly, if a candidate is unable to pass all five tests when compared with any element of the database, the candidate is probably not a character and is ignored.

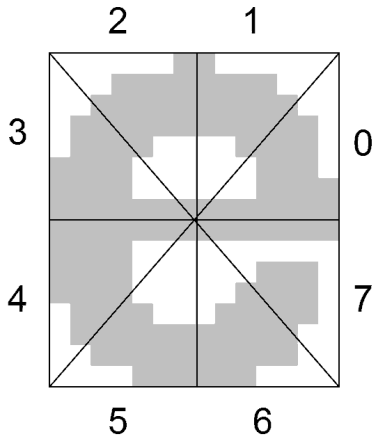

Figure 9: By dividing each connected component into several slices, the character-finding algorithm better captures its uniform appearance.

3.3.1 The Pixel Color Test

We now discuss the fifth and final comparison made to determine if a candidate connected component is sufficiently similar to one in the database. First, each of the two connected components is divided into several "pie slices," as shown in Figure 9. The point of intersection between these slices is the centroid of the connected component, calculated by averaging the coordinates of all the constituent pixels. Then, the character-finding algorithm counts the number of pixels of each color in each slice. In the example figure, some of the pixels are gray, and others are white. Once each connected component is divided in this way, the algorithm executes the following test on each pair of corresponding slices:

If $\left| \frac{C}{\left(\frac{A}{n}\right)} - \frac{C'}{\left(\frac{A'}{n}\right)} \right| > T_C$, then reject. C is the count of pixels in a particular slice with the color of the connected component, A is the area of the connected component, and n is the number of slices. C' and A' are the respective values for the connected component in the font database. In Figure 9, the "e" is divided into eight slices. By dividing the connected components into slices, the character-finding algorithm captures the uniform appearance of each character.

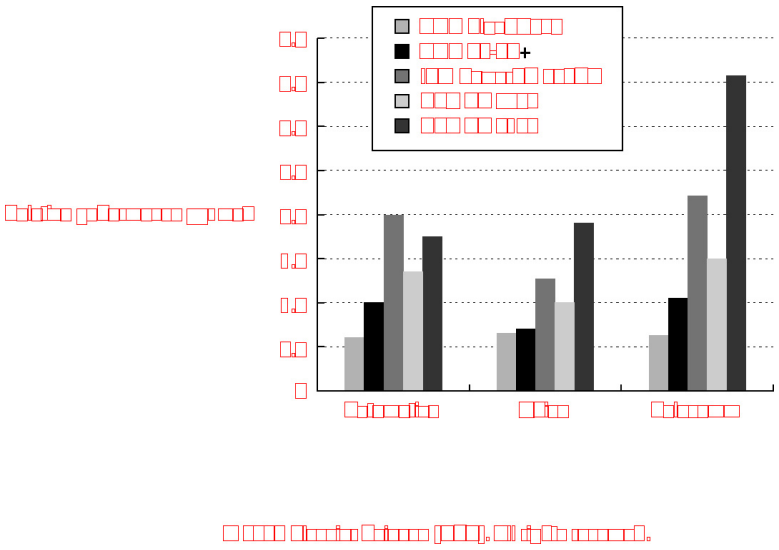

Figure 10: Character-finding algorithm example.

3.3.2 Example

Figure 10 shows an example run of the character-finding algorithm. By comparing each of the connected components in the image with those stored in the font database, the character-finding algorithm achieves a high degree of accuracy, as this example shows. The red boxes in this image denote connected components that the character-finding algorithm classified as characters. All other connected components in this image were classified as graphical elements. The font database for this example was constructed using another, similar image. Note that the plus sign ('+') was incorrectly classified as a graphical element; the small training set used is the most likely explanation for this error.

3.4 Problems With This Approach

Unfortunately, the simplicity of this character-finding algorithm renders it vulnerable to several common and unavoidable problems like that seen in the previous example. In this section, we examine these problems in detail. First, we discuss the problem of noise in an image: what happens when a user wishes to find text in a scanned image? Second, we examine the effect of individual characters joining with each other or other graphical elements

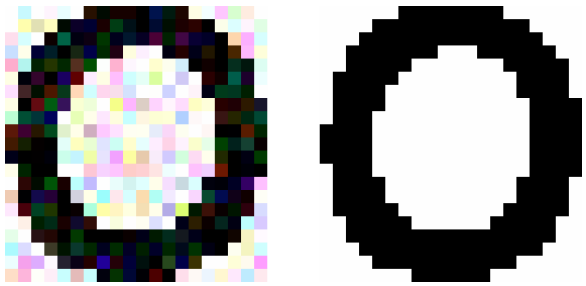

Figure 11: Noise can be a major problem with the character-finding algorithm. Fortunately, a variety of straightforward solutions exist as workarounds.

as a result of noise or ink spread. Finally, we consider cases in which certain connected components are ambiguous and cannot be classified as either a character or a graphical element.

3.4.1 Noise

When an image is scanned into a computer, noise is inevitably introduced. Figure 11 provides an example of this effect. As a result, our original definition of "connected component" breaks down: every pixel becomes disconnected from its neighbors because their colors are all slightly different. Running the character-finding algorithm on such an image is ineffective because it would simply mark individual pixels as "characters" on the basis of their color.

Fortunately, solutions to this problem exist. One is to execute one of the many existing noise reduction algorithms. Another is to find the characters on a separate black and white image, kept separate from the original scanned image. This black and white image could be derived through a simple "thresholding" technique which forces every pixel to take on either of two colors. Once the characters are found in this modified image, the image could be discarded. A third solution is to modify the definition of "connected" to include similar colors, not merely the same color.

3.4.2 Character Joining

Another set of detrimental effects are those of ink spread and low resolution scanning equipment. Problems such as those illustrated in Figures 12 and

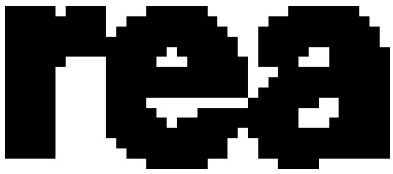

Figure 12: Ink spread and low resolution scans may cause several characters to become connected.

Figure 13: Characters may also become connected with graphical elements.

13 are consequences of these effects. In the first figure, several individual characters have joined together to form a single, large, connected component. Because of the unusual width and color distribution of this connected component, the character-finding algorithm may have difficulty concluding it is, in fact, a set of characters. In Figure 13, the number 25 is connected to the circle surrounding it. Since the character-finding algorithm does not attempt to split a connected component into graphical and textual subcomponents, connected components like these will never be handled properly. A third, related problem is shown in Figure 14, in which an underline has caused the characters with descenders, namely the Ps and Qs, to be joined into a single large connected component. Because this component is so large, the character-finding algorithm will probably never mark it for extraction.

Several solutions to these problems exist. One such technique, known as a mathematical morphology [7], appears to be well suited to solving the problems shown in Figures 12 and 13. The basic idea of that approach is to "erode" all the connected components such that they become disconnected. We have not yet considered a workable solution for the problem in which the

Figure 14: Underlining is a particularly troublesome piece of formatting which can prevent the character-finding algorithm from properly identifying all the characters in the image.

text is underlined.

3.4.3 Ambiguous Connected Components

The final problem is also among the hardest to solve. In most cases, given an example connected component, a human is able to tell whether it is a character or something else. However, certain characters are ambiguous because their classification depends on the context in which they are used. For example, a symbol like the letter "O" should usually be treated as a character and extracted. This symbol may also be used as a graphic, however, in which case it should be left as is. Similarly, small dots like periods are used frequently in both text and graphics.

These types of problems are likely unavoidable with the current character-finding algorithm, as determining the appropriate course of action relies on the situation's context. Since such context is not considered, these kinds of problems are inevitable.

3.5 Results

Despite the various problems with this approach, the character-finding algorithm does exhibit reasonably high accuracy. This section discusses two closely related experiments to test this assertion. The first test is intended to reflect real world usage, in which a user trains an implementation of the character-finding algorithm with the first few images in a book and then has it find the characters automatically in the remaining images. The second test represents ideal conditions because the training images were chosen more carefully and the images used included only bar charts.

3.5.1 Test 1: Real World Usage

In this test, the character-finding algorithm was trained using the first three figures and diagrams in [3] to create a font database consisting of 475 con-

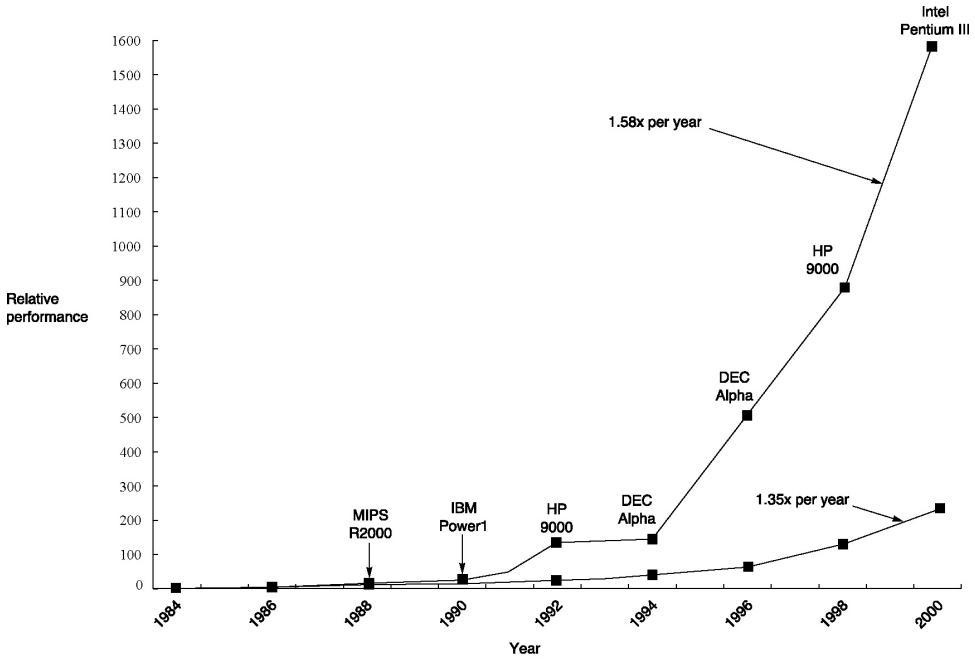

Figure 15: An image used for training (Ch1-fig01 from [3]). This particular image contains 234 connected components, 160 of which are characters or small groups of characters.

nected components. These three images are all line graphs. Figure 15 shows one of the images used in the training set.

The character-finding algorithm was then run on the 25 remaining figures and diagrams in the first two chapters. These images include line graphs, bar charts, and diagrams.

Table 2 summarizes the character recognition accuracy of the character-finding algorithm by showing the total number of connected components in the image, the number classified as characters, the number of false positives, and the number of false negatives. The last column provides an error percentage, calculated by adding the number of false positives and negatives and dividing by the total number of connected components in the image. The last line in the table provides various summary statistics. The parameters used to achieve these results are $T_W = 1.0$, $T_H = 1.0$, $T_A = 1.0$, and $T_C = 1.0$.

Training image	Total CCs	Character CCs
Ch1-fig01	234	160
Ch1-fig05	205	140
Ch1-fig06	1561	175

Table 1: Images used for training (test 1). These are the first non-photographic images in [3] and thus represent a very basic selection of images to use for training.

Furthermore, the character-finding algorithm divided each character into 5 pie slices.

Most of the false positives were nearly the same in every image; they consisted of small black boxes surrounding shaded regions in the image's legend. These boxes share many of the characteristics of the letter "O" which explains why they were consistently marked as characters.

The performance of the character-finding algorithm on Ch2-fig01 also warrants additional explanation. This figure contains a number of decorative dot-like graphics. The algorithm incorrectly classified all these dots as characters, resulting in the high error rate.

The false negatives present in Ch2-fig24 and Ch2-fig41 occurred because the graphic used a different font for some of the text: the character-finding algorithm missed the decimal points and the dots on the lowercase letter i. The false negatives in Ch2-fig27 consisted exclusively of 3 equal signs. In all three of these cases, the false negatives could be eliminated by employing a better (larger) training set.

3.5.2 Test 2: Ideal Usage

In this test, three images were again used for training, but these three were chosen more carefully because they better reflect the appearance of the remainder of the images. Furthermore, the images used include only bar charts, in contrast to the first experiment in which bar charts, line graphs, and miscellaneous diagrams were all included. The training set included 515 characters.

After training, the character-finding algorithm was run on 19 more bar charts from the first three chapters. Other than the training set and the images used, the same parameters were used as in the previous test. Table

Figure	# CCs	Char. CCs	False (+)	False (-)	% Error
Ch1-fig10	293	287	0	0	0
Ch1-fig19	786	298	2	0	0.25
Ch1-fig20	1326	293	2	0	0.15
Ch1-fig22	986	336	2	0	0.2
Ch1-fig23	1130	305	2	0	0.18
Ch1-fig25	575	182	5	0	0.87
Ch1-fig26	599	183	5	0	0.83
Ch1-fig27	519	177	5	0	0.96
Ch1-fig28	832	173	0	0	0
Ch2-fig01	193	183	39	0	20.21
Ch2-fig07	274	273	0	0	0
Ch2-fig08	1263	190	0	0	0
Ch2-fig09	174	172	2	0	1.15
Ch2-fig10	1396	200	0	0	0
Ch2-fig12	197	195	2	0	1.02
Ch2-fig19	195	193	2	0	1.03
Ch2-fig20	1130	185	0	0	0
Ch2-fig22	257	255	2	0	0.78
Ch2-fig23	482	476	3	0	0.62
Ch2-fig24	667	664	0	2	0.3
Ch2-fig26	328	326	4	0	1.22
Ch2-fig27	557	547	1	6	1.26
Ch2-fig34	417	413	10	0	2.4
Ch2-fig38	227	225	4	0	1.76
Ch2-fig41	1342	166	0	9	0.67
Total	16145	6897	92	17	0.68

Table 2: Character recognition results (test 1). "CC" stands for "connected component," and "Char." is an abbreviation for "Character." In this experiment, the user trained the character-finding algorithm using the first three images in the book, and then ran it on all the figures and diagrams in the first two chapters.

Training image	Total CCs	Character CCs
Ch1-fig19	840	240
Ch1-fig25	606	117
Ch1-fig08	228	158

Table 3: Images used for training (test 2). These images were chosen on the basis of their similarity to the remainder of the images being processed, and thus represent a better choice than the first three bar charts.

4 summarizes the results. Clearly, performance improved very little over the first test. Despite the fact that there were no false negatives, the same false positives were present.

4 Grouping Characters Into Labels

Although finding the existing characters is a major step toward automatically converting an image to a tactile format, inserting a set of Braille characters in their place would be difficult if done one at a time. Because Braille is generally several times the width and height of normal text in order to be legible, simply replacing the original text with Braille is not viable as the Braille characters would overlap.

Therefore, we have developed a companion label-finding algorithm which, when presented with a set of isolated characters (the connected components classified as characters by the character-finding algorithm) and their locations, will group them into short labels. Once in this form, their management becomes much simpler. Furthermore, we can take advantage of the technique shown in Figure 5 on page 9 to convert images of these labels into Braille using conventional OCR software.

This section presents the details of the label-finding algorithm. As before, we begin with a training phase in which the user provides the algorithm with some examples of appropriate labels. Then, once trained, the label-finding algorithm groups characters in subsequent images similarly.

Figure	# CCs	Char. CCs	False (+)	False (-)	% Error
Ch1-fig20	1326	293	2	0	0.15
Ch1-fig22	986	336	2	0	0.2
Ch1-fig23	1130	305	2	0	0.18
Ch1-fig26	599	183	5	0	0.83
Ch1-fig27	519	177	5	0	0.96
Ch1-fig28	832	173	0	0	0
Ch2-fig09	174	172	2	0	1.15
Ch2-fig19	195	193	2	0	1.03
Ch2-fig22	257	255	2	0	0.78
Ch2-fig26	328	326	4	0	1.22
Ch2-fig38	227	225	4	0	1.76
Ch3-fig09	309	307	2	0	0.65
Ch3-fig15	377	375	3	0	0.8
Ch3-fig17	199	198	0	0	0
Ch3-fig35	146	145	0	0	0
Ch3-fig39	233	231	5	0	2.15
Ch3-fig40	277	275	3	0	1.08
Ch3-fig42	232	230	6	0	2.59
Ch3-fig44	200	198	4	0	2
Total	8546	4597	53	0	0.62

Table 4: Character recognition results (test 2). In this experiment, the user trained the character-finding algorithm using three specific bar graphs which, subjectively, seemed to represent the bar graphs best. Then, the user ran the character-finding algorithm on all the bar graphs in the first two chapters plus the first several in the third chapter. This scenario represents ideal conditions.

Figure 16: By forming the minimum spanning tree that connects the centroids of each character in a label, the label-finding algorithm can derive the distance between the centroids of each character. In this example, the trailing "ce" is treated as a single character because they are connected.

4.1 Label Training

The first step is to train the label-finding algorithm with some examples of character labels. This step is accomplished quickly with an appropriate graphical user interface. Once trained with some example labels, the label-finding algorithm examines several defining characteristics of each label. These defining characteristics include:

- The angle of the line of best fit
- The mean squared error of the line of best fit
- The spacing between the adjacent characters

The line of best fit is calculated using the centroids of each of the characters in the label. The centroid is essentially the "center of mass:" the average location of all the points making up the character. Because text may be aligned vertically, perpendicular regression (a kind of principal component analysis) [4] is used to find the equation of the line. Perpendicular regression is similar to linear regression, except instead of minimizing the sum of the squares of the vertical distances of points to the line, it minimizes the sum of the squares of the perpendicular distances.

In addition to the line through the characters, the label-finding algorithm maintains information on the distances between characters within the label. These numbers are obtained by calculating a minimum spanning tree using the centroids of each character as nodes. The lengths of the edges are the distances between the centroids of the characters. Figure 16 provides an example of this step.

4.2 Label-Finding Algorithm

Once trained with a representative set of labels, the label-finding algorithm can mark similar labels in subsequent images. It considers only those connected components previously marked as characters because it would not make sense to group them with graphical elements.

After the training phase, the label-finding algorithm consists of the following steps:

1. Form a minimum spanning tree which connects the centroids of every character in the image.

2. Remove edges whose lengths and angles are sufficiently different from the gaps between adjacent characters seen previously.

3. Remove edges which are inconsistent with surrounding edges.

4. Form labels from all characters which are themselves connected with each other via an edge in this resultant graph, and then merge them if the result is sufficiently similar to a label in the training set.

The following four subsections discuss the details of these steps.

4.2.1 Minimum Spanning Tree

The first step of this process is straightforward. Using the centroids of the characters as nodes, the label-finding algorithm forms a minimum spanning tree [1]. An example of such a formation is shown in Figure 17.

4.2.2 Prune Invalid Edges

The second step is to remove edges which could never represent an adjacent pair of characters. This process involves iterating over every edge in the tree, and comparing it with pairs of adjacent characters in the training set. A comparison consists of the following calculations:

1. Calculate the absolute value of the x- and y-distances between the pair of characters in the database and the pair being examined in the minimum spanning tree. Denote the values for the characters stored in the database as Dx_s and Dy_s, and the values for the characters under examination as Dx_c and Dy_c.

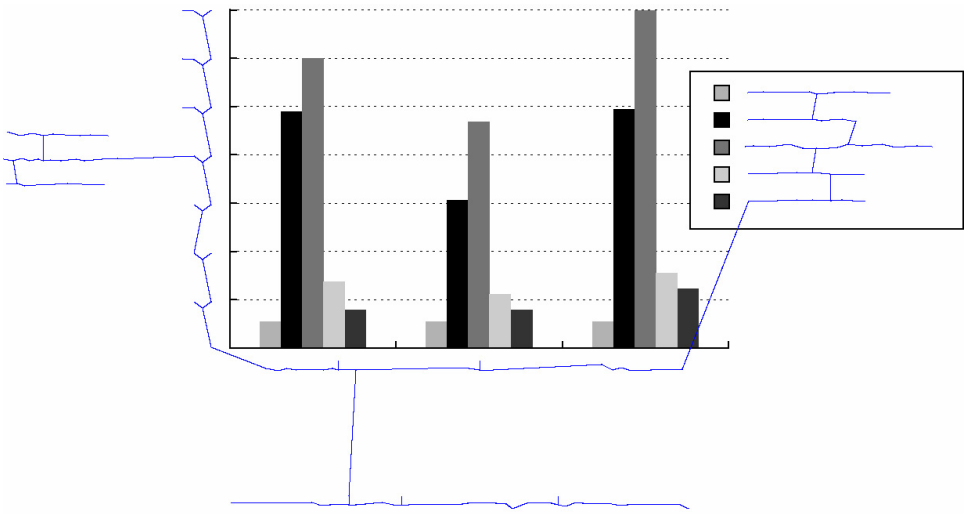

Figure 17: Finding labels step 1. The minimum spanning tree.

2. Calculate the angle of the line connecting each pair of characters. Denote the angle between the stored characters as θ_s and the angle between the candidate characters as θ_c.

3. Calculate the absolute value of the difference between these distances. Denote these values as ΔDx and ΔDy.

4. Calculate the absolute value of the difference between the angles. Denote this value as $\Delta\theta$.

5. If $\Delta Dx/Dx_s < T_m$ and $\Delta Dy/Dy_s < T_m$ and $\Delta\theta < T_a$, then these two characters may belong to the same label and the edge should be left in place. As in the character-finding algorithm, the parameters T_m and T_a are "magic numbers" tunable by the user.

If the label-finding algorithm compares the current pair with every pair in the database, and the test fails in every case, then it concludes that this pair of characters could never be members of the same label and it removes the edge connecting them from the minimum spanning tree. Figure 18 continues with the previous example, showing the resultant graph after performing this edge removal step. All the long edges have been removed: since no labels in the training set have adjacent characters so far apart, the label-finding

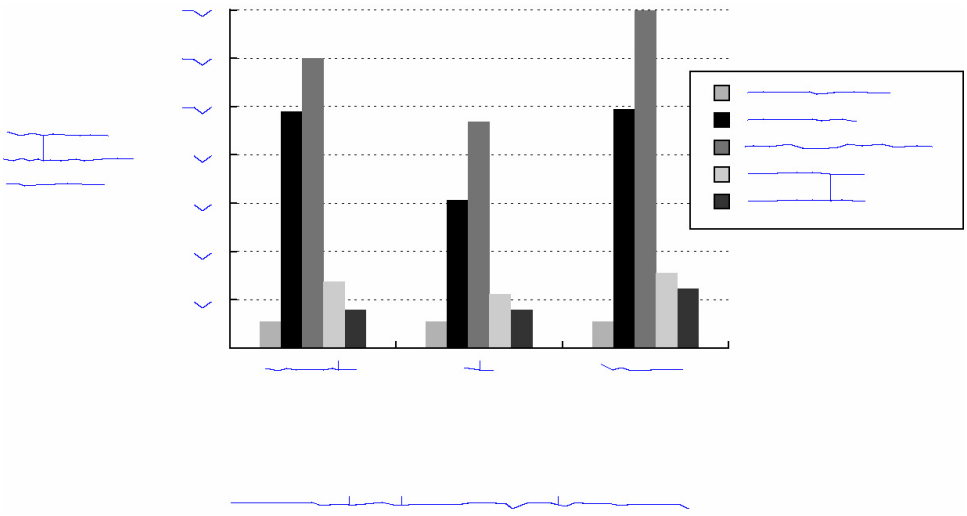

Figure 18: Finding labels step 2. The reduced graph.

algorithm can conclude that the characters are not members of the same label.

4.2.3 Prune Inconsistent Edges

The third step in this process is to remove edges "inconsistent" with their neighbors. The label-finding algorithm accomplishes this step by iterating over every edge of each vertex. For a particular edge, it tests whether the edge is consistent with at least one other edge around the current vertex, and one other edge around the vertex to which it is connected. This consistency check compares only the angles of the two edges. If the test is successful, the label-finding algorithm leaves the edge in place. Otherwise the edge is discarded.

The purpose of this test is to remove edges connecting pairs of characters which could be members of the same label when examined in isolation, but are likely not when looked at in context. For example, vertical edges frequently appear connecting two characters on adjacent lines of text. These edges are typically not consistent with surrounding edges, which connect pairs of characters on the same line. Figure 19 continues with the same example. The label-finding algorithm has removed the vertical edges on the left and right of the diagram—these edges were inconsistent with their surroundings.

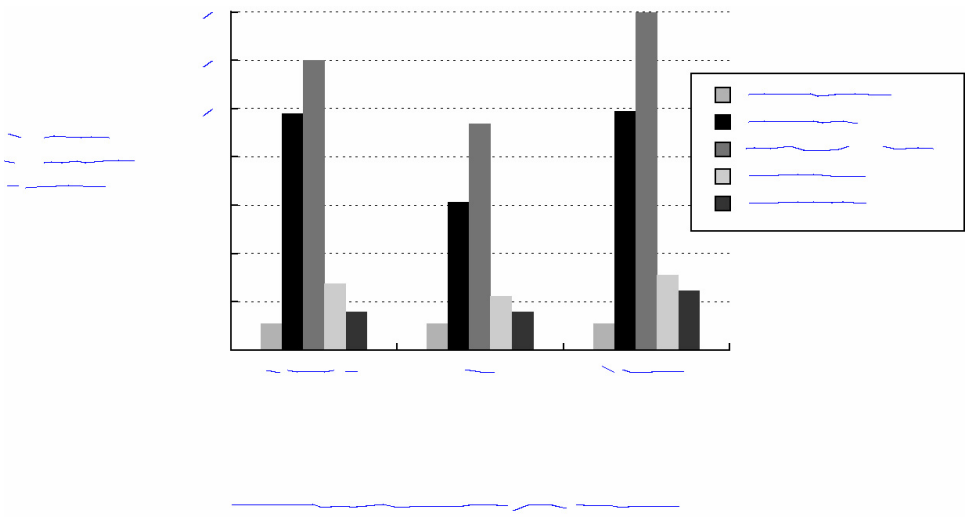

Figure 19: Finding labels step 3. The consistent graph.

4.2.4 Regroup If the Result Is Consistent

The final step is to treat the remaining groups of characters as labels, and then merge them if the result is consistent with a label in the training set. A pair of characters are members of a label at this stage if they are connected via some path in the graph outputted during the previous step.

The idea is simple: compare every label with every other label and merge them only if the result is similar to a label in the training set. The label-finding algorithm determines this similarity by examining the line of best fit. If the angle and mean squared error of this line is sufficiently close to that of a label in the training set, the label is kept. Otherwise, the label-finding algorithm attempts to merge the current label with a different label. Figure 20 shows the final labels achieved after executing this step.

4.3 Results

Table 5 summarizes the results obtained for label creation. Using the same first three figures and diagrams in [3] as a training set, we supplied the label-finding algorithm with 126 examples of character labels. These images contained only horizontal and diagonal labels. Using parameters $T_a = .5$ (angle tolerance) and $T_m = .7$ (distance tolerance), we then ran the label-finding algorithm on the remaining images.

Image	Labels	Actual #	Mis-grouped CCs	False (+)	False (-)
Ch1-fig10	39	39	0	0	0
Ch1-fig19	48	48	0	0	0
Ch1-fig20	53	53	0	0	0
Ch1-fig22	44	45	0	1	0
Ch1-fig23	42	42	0	0	0
Ch1-fig25	20	20	0	0	0
Ch1-fig26	20	20	0	0	0
Ch1-fig27	19	19	0	0	0
Ch1-fig28	19	21	0	2	0
Ch2-fig01	12	12	0	0	0
Ch2-fig07	46	44	0	0	2
Ch2-fig08	31	31	0	0	0
Ch2-fig09	19	19	0	0	0
Ch2-fig10	32	32	2	0	0
Ch2-fig12	24	24	0	0	0
Ch2-fig19	18	18	0	0	0
Ch2-fig20	35	35	0	0	0
Ch2-fig22	28	28	1	0	0
Ch2-fig23	40	40	0	0	0
Ch2-fig24	35	35	0	0	0
Ch2-fig26	32	32	0	0	0
Ch2-fig27	40	40	1	0	0
Ch2-fig34	65	65	4	0	0
Ch2-fig38	35	35	0	1	1
Ch2-fig41	28	26	0	0	2
Total	824	823	8	4	5

Table 5: Label formation results.

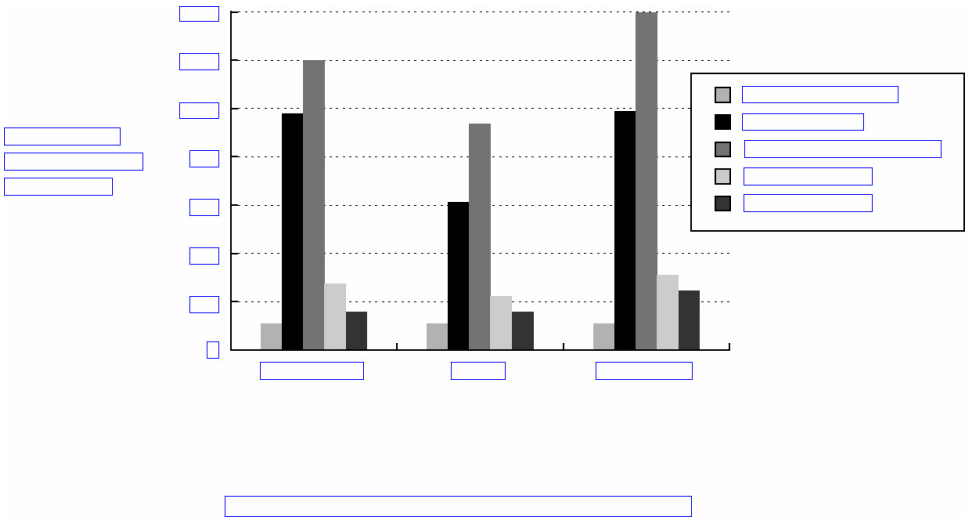

Figure 20: Finding labels step 4. The final labels.

The first column in the table is the image name. The second is the number of labels the label-finding algorithm created, and the third is the actual number of labels in the image. The fourth column is the number of characters assigned to one group which should have been assigned to another. Therefore, the value in this column could exceed the number of labels in the image under the right circumstances. The fifth and sixth columns are the number of false positives and negatives: situations in which two labels were incorrectly joined (a false positive) or left unjoined (a false negative).

Overall, the accuracy was quite high. In 16 of the 25 images, the label-finding algorithm made no errors of any kind. Given the characters in these images, the label-finding algorithm grouped them all in the most appropriate fashion.

The images in which the characters were placed in the wrong label often contained areas in which several short labels existed on adjacent horizontal lines of text. The label-finding algorithm mistakenly concluded these characters should be joined into diagonal groups.

False positives occurred when several labels along the x-axis of a graph were long enough to be near each other: the label-finding algorithm assumed that since the gap between the labels was not much greater than a couple of spaces, the labels should be merged. This kind of error would likely not affect OCR accuracy or the ability to reinsert the text as Braille because the

gap between the two groups of characters is maintained whether the label is split in two or not.

False negatives often appeared in which the dot on a lowercase letter i or j was left in its own label, when it should have been merged. Such an error would not affect OCR accuracy, as both versions of the software that we have worked with can recognize such characters even if the dots are missing. The label-finding algorithm chose not to group the dot with the rest of the label because doing so would significantly increase the mean squared error of the line of best fit. A possible solution to this problem would be to assign less weight to smaller characters during the mean squared error calculation, thereby preventing a dot from inappropriately skewing the quality of the fit.

5 Erasing and Outputting Text

Once the text has been grouped into labels, it needs to be erased and output in a format better suited to existing OCR software.

Erasing the text is relatively straightforward. Since the text is simply a set of connected components, the only problem is choosing the color with which to replace each connected component. The strategy employed is to find the most frequently occurring color in each "slice" aside from the color of the connected component itself. The same slices are used here as in the character-finding algorithm for determining whether a given connected component is a character. Then, all the connected component pixels in that slice are replaced with the new color. Better algorithms likely exist for this problem, but given the small number of colors present in most scientific figures and diagrams, this method works fine.

To output the text in a format usable by OCR software, the all the labels in the image are concatenated into a single, large bitmap. The algorithm outputs the location of the text within the image separately. With this information, the OCR software can accurately determine the text in the labels, and another algorithm under development by Satria Krisnandi, an undergraduate at the University of Washington, can effectively place the Braille into the image.

6 Conclusion and Future Work

With software to automate the process of finding and grouping characters in scientific figures and diagrams, creating tactile graphics becomes much simpler. Instead of tracing single images and manually inserting text, these algorithms process large batches of images simultaneously. They finds the text, group it, remove it from the image, and prepare it for input into existing optical character recognition software. The user need only briefly train the software, a process which takes just a few minutes.

In the future, we plan several significant enhancements to the algorithms described herein. First, applying a formal machine learning technique such as support vector machines should improve character recognition accuracy further and simplify the user's task by eliminating the need to fine tune parameters to the character-finding algorithm. Since support vector machines are well suited to classification problems, they are a natural technique to apply. Second, through refinement of the user interface for our implementation of these algorithms, it should be possible to simplify the work flow and increase efficiency. In particular, our current implementation does not allow the user to correct errors easily, a shortcoming that prevents this tool from being truly useful. Other problems remain, including placement of the Braille after optical character recognition and automatic recognition of mathematical formulae, but solutions to these problems are also under development by other students.

Over the next several years, blind students will no longer need to waste so much time trying to gain access to their textbooks. By employing software like that described here, tactile graphics specialists around the country will be able to increase their productivity considerably. They will finally be able to provide students like John with complete access to exactly the same literature as everyone else.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.

[2] L. A. Fletcher and R. Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):910–918, November 1988.

[3] J. Hennessy and D. Paterson. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann Publishers, San Francisco, 3rd edition, 2003.

[4] J. Hollmen. Principal component analysis, March 1996. http://www.cis.hut.fi/∼jhollmen/dippa/node30.html.

[5] R. Kasturi, S. T. Bow, W. El-Masri, J. Shah, J. R. Gattiker, and U. B. Mokate. A system for interpretation of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):978–992, October 1990.

[6] R. E. Ladner, M. Y. Ivory, R. Rao, S. Burgstahler, D. Comden, S. Hahn, M. Renzelmann, S. Krisnandi, M. Ramasamy, B. Slabosky, A. Martin, A. Lacenski, S. Olsen, and D. Groce. Automating tactile graphics translation. Assets 2005 to appear.

[7] L. G. Shapiro and G. C. Stockman. *Computer Vision*. Prentice Hall, Englewood-Cliffs NJ, 2001.