# Understanding Freehand Diagrams: Combining Appearance and Context for Multi-Domain Sketch Recognition

by

Tom Yu Ouyang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012

Author . . . [illegible] . . . . . . . . . . . . . . . . . . . . . . . .
Department of Electrical Engineering and Computer Science
January 13, 2012

Certified by . . . [illegible] . . . . . . . . . . . . . . . . . . . . . .
Randall Davis
Professor
Thesis Supervisor

Accepted by . . . . . . . . [illegible] . . . . . . . . . . . . . . . . . . . .
Leslie A. Kolodziejski
Chair, Committee on Graduate Students

# Understanding Freehand Diagrams: Combining Appearance and Context for Multi-Domain Sketch Recognition

by

Tom Yu Ouyang

Submitted to the Department of Electrical Engineering and Computer Science
on January 13, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

## Abstract

As our interaction with computing shifts away from the traditional desktop model (e.g., towards smartphones, tablets, touch-enabled displays), the technology that drives this interaction needs to evolve as well. Wouldn't it be great if we could talk, write, and draw to a computer just like we do with each other? This thesis addresses the drawing aspect of that vision: enabling computers to understand the meaning and semantics of free-hand diagrams. We present a novel framework for sketch recognition that seamlessly combines a rich representation of local visual appearance with a probabilistic graphical model for capturing higher level relationships. This joint model makes our system less sensitive to noise and drawing variations, improving accuracy and robustness. The result is a recognizer that is better able to handle the wide range of drawing styles found in messy freehand sketches. To preserve the fluid process of sketching on paper, our interface allows users to draw diagrams just as they would on paper, using the same notations and conventions. For the isolated symbol recognition task our method exceeds state-of-the-art performance in three domains: handwritten digits, PowerPoint shapes, and electrical circuit symbols. For the complete diagram recognition task it was able to achieve excellent performance on both chemistry and circuit diagrams, improving on the best previous results. Furthermore, in an on-line study our new interface was on average over twice as fast as the existing CAD-based method for authoring chemical diagrams, even for novice users who had little or no experience using a tablet. This is one of the first direct comparisons that shows a sketch recognition interface significantly outperforming a professional industry-standard CAD-based tool.

Thesis Supervisor: Randall Davis
Title: Professor

# Acknowledgments

I have learned and experienced so much during my time here at MIT, and I am so thankful to the many people who have helped me along the way. This thesis would not have been possible without them.

First and foremost, I would like to thank Randy Davis for being a wonderful mentor and advisor. His vision and guidance have helped shape this work from the very beginning, and our frequent intellectual discussions have been some of my most formative and valued moments here. I am constantly amazed at how easily he can distill the most complicated ideas down to their core issues, and communicate them in such a clear and concise manner. He has taught me how to always look at the big picture and ask the right research questions, to not shy away from the hard problems, and at the same time to avoid falling into the bottomless pits.

I want to thank my committee members Bill Freeman and Rob Miller. This work would not be what it is today without their input and encouragement, and the insights from their diverse backgrounds in machine learning and HCI. I also want to thank Seth Teller for his support and advice on my M.S. thesis, and helping me find the direction for my dissertation.

I am also very grateful to have had wonderful collaborators and fellow lab members over the years, many of whom have become close friends: Aaron Adler, Sonya Cates, Danica Chang, Chih-Yu Chao, Andrew Correa, Jacob Eisenstein, Tracy Hammond, James Oleinik, Mike Oltmans, Lakshman Sankar, Metin Sezgin, Jeremy Scott, Yale Song, and Ying Yin.

A special thanks to Nira Manokharan, who was always there to help me and point me to the right resources. Thanks also to all of the great people in The Infrastructure Group (TIG) for making the lab run so smoothly.

I would also like to thank all of my friends here who have made this a truly unforgettable experience. I won't try to list them all because I know I will forget someone.

Finally, my biggest thanks to my family for their support, their patience, and their love. To my girlfriend Jennifer, who over the years has been my source of strength and inspiration. Thank you for sharing the journey with me.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

With the growing popularity of mobile devices such as smart-phones (e.g., Android, iPhone), tablets (e.g., iPad), and touch-enabled displays (e.g., SMART Board, Microsoft Surface), computing is becoming an ever more ubiquitous part of our daily lives. As our interaction with computing shifts away from the traditional desktop model, the technology that drives this interaction needs to evolve as well. For example, while it would be possible to jot down notes on a tablet using the software keyboard or draw diagrams using your finger, neither of these interactions feel very natural. Furthermore, to the computer these drawings are just pixels on the page without any deeper meaning. Wouldn't it be better if we could talk, write, and draw to a computer just like we do with each other, and then actually have the computer *understand* us? What kinds of new interactions and experiences could this enable? These questions lie at the intersection of human computer interaction and artificial intelligence, and are part of the long-term vision of making interacting with a computer as easy and natural as interacting with another human being.

This thesis addresses the *drawing* part of that vision: enabling computers to understand the meaning of free-hand diagrams. Most people have been drawing since they were toddlers, and sketches offer a natural, easy, and flexible medium for expressing and communicating ideas. More than just simple drawings, sketches and diagrams can be an essential means of communicating information in many domains. They are also an important part of the early design process, where they help people explore rough ideas and solutions in an informal environment. Many professionals, such as architects, chemists, and engineers,

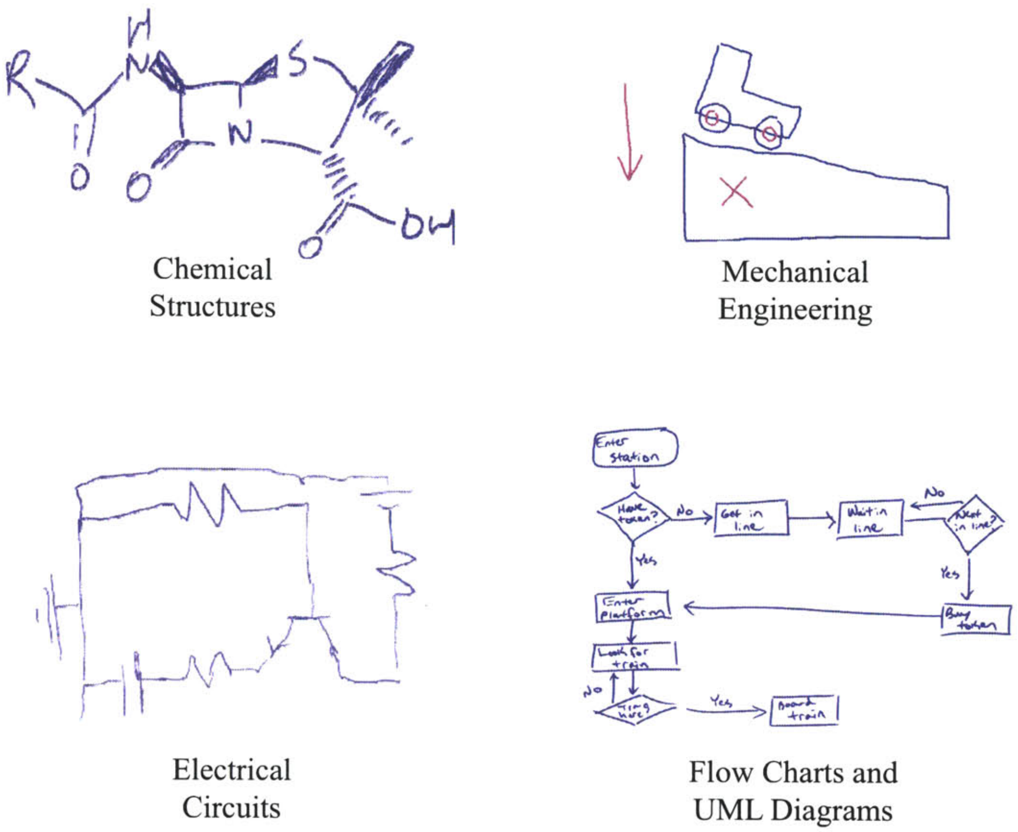


Figure 1-1: Sketches are used to express and communicate ideas in a number of different domains, including chemistry, mechanical engineering, electrical engineering, and flow charts.

use sketches every day to help them throughout the design process. Figure 1-1 shows some examples of these types of diagrams and the areas where they play an important role, including chemistry, electrical engineering, mechanical engineering, and flow charts.

Despite the naturalness and ease of drawing on paper, the problem is that these drawings are inherently static - merely ink on a page without any inherent meaning besides that inferred by the human observer. Even if the sketch is captured digitally as an image or a vectorized sequence of pen samples, we still can't easily ask the computer to look up the diagram in a database (except as an image), visualize it in another form (e.g., a 3-D rendering of a molecular diagram), or clean it up for a presentation or publication.

The goal of this thesis is to develop a new sketch recognition framework that provides the naturalness and speed of drawing on paper (as illustrated in figure 1-2). To accomplish this task we identify four main requirements. 1) It should handle the range of drawing styles found in messy freehand sketches, and be competitive with or exceed existing CAD-like authoring tools in terms of usability and speed. This means that the recognition accuracy needs to be high enough so that the user does not need to spend much time or cognitive load on making corrections. 2) It should be general enough that it can be applied to multiple real-world domains, and be easily extensible to new domains. 3) It should preserve, as much as possible, the familiar experience of using pen and paper. 4) It should be fast enough to keep up with the user in real time.

## 1.1 Background

Current work in sketch recognition can, very broadly speaking, be separated into two groups. The first approach focuses on relationships between geometric primitives such as lines, arcs, circles, etc. However, in real-world sketches, it is difficult to extract these primitives reliably. Circles may not always be round or closed, line segments may not be straight, and stroke artifacts like pen-drag (not lifting the pen between regions that are usually separated), over-tracing (drawing over a previously drawn stroke), and stray ink may introduce false primitives that lead to poor recognition.

A second approach focuses on the visual appearance of shapes and symbols. The main advantage of vision-based approaches is their robustness to variations in drawing styles and to the types of artifacts mentioned earlier. However, these methods do not model the spatial relationships between neighboring shapes, relying on local appearance to classify a symbol. Hence, they do not take advantage of the context around a symbol that can make the recognition task easier. While the promise of sketch recognition interfaces has widespread and growing appeal, there are still open challenges in terms of recognition accuracy, robustness, and scalability before they are ready to be deployed in the real world.

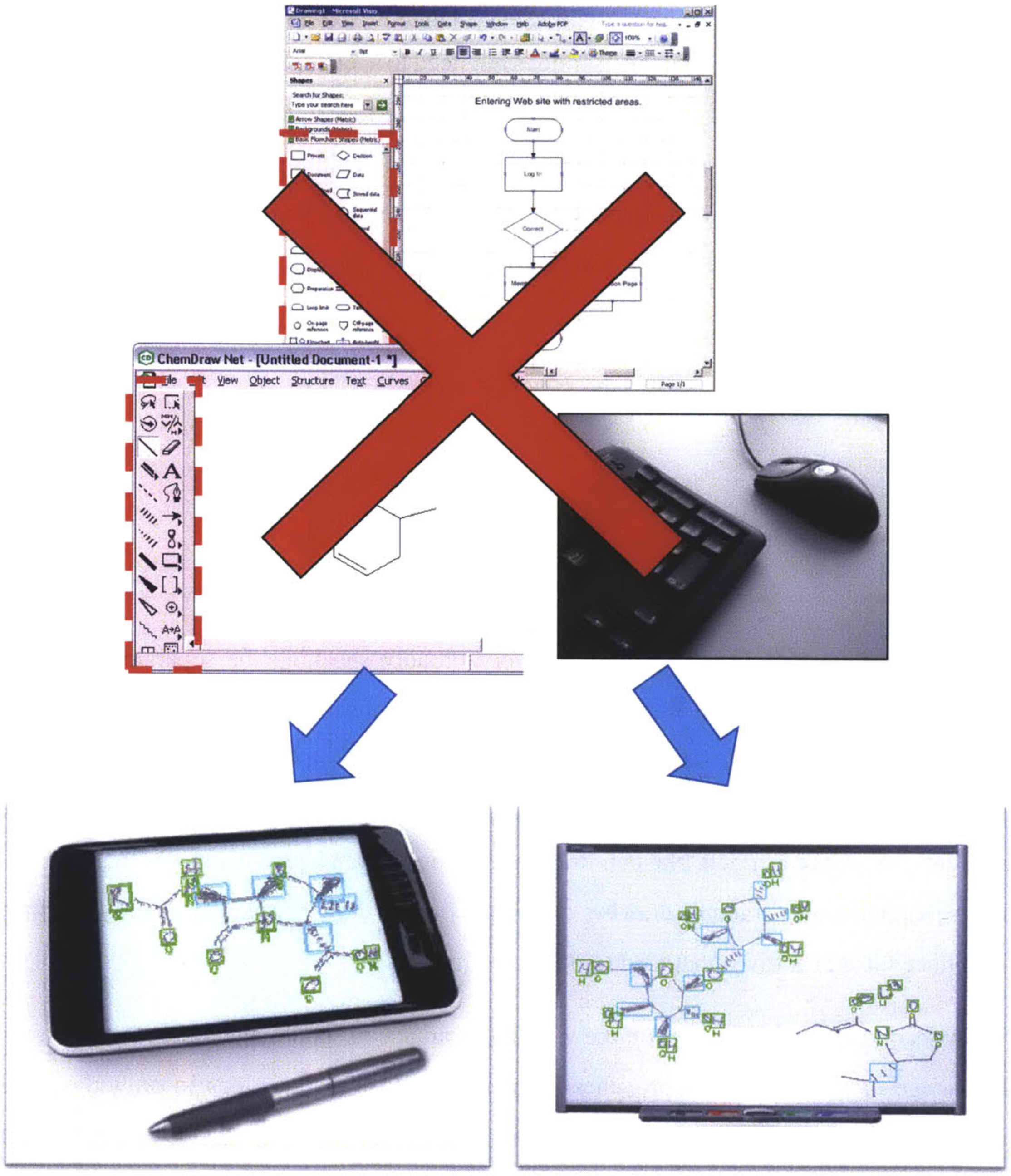


Figure 1-2: The goal of our research is to move from traditional mouse-and-menu based CAD interfaces to a more natural sketch-based interaction that recreates the experience of drawing on paper.

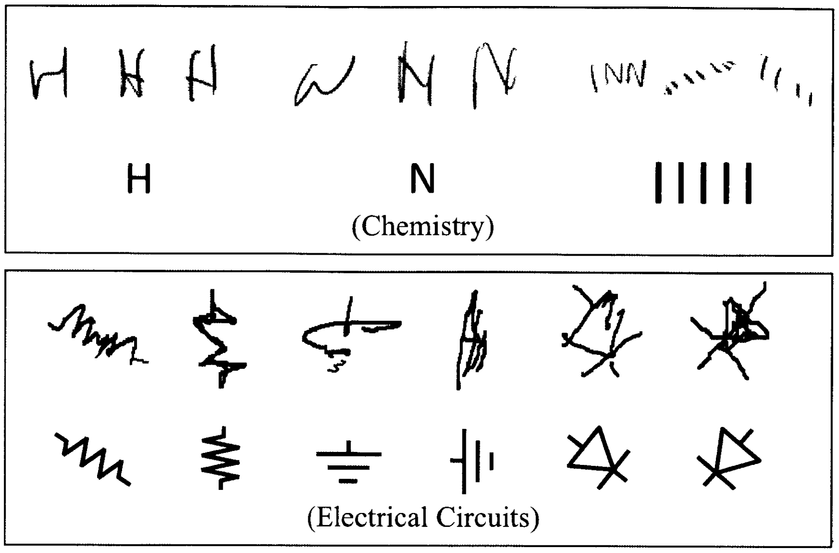


Figure 1-3: Symbols taken from a dataset of hand-drawn chemical and electrical circuit diagrams, illustrating some of the types of variations found in freehand sketches.

## 1.2 Challenges

Making sense of messy real-world sketches can be difficult for several reasons. First, symbols like the ones in Figure 1-3 can exhibit a great deal of intra-class variation due to local distortions, rotations, and non-uniform scaling. Second, symbols may exhibit artifacts like over-tracing and pen-drag. Finally, in addition to these types of visible differences, two seemingly identical symbols can be drawn differently at the stroke level (e.g., the strokes may be drawn in a different order or in opposite directions). These types of variations present major challenges for traditional sketch recognition systems [27].

## 1.3 Natural Sketch Interaction

This thesis presents a novel sketch understanding architecture that seamlessly combines both visual appearance and spatial context. The resulting interface, demonstrated in Figure 1-4, provides a more natural way to specify diagrammatic information (such as chemical structures) to a computer. To preserve the familiar experience of using pen and paper, our system supports the same symbols, notations, and drawing styles that people are already accustomed to. However, unlike physical pen and paper, sketches created and interpreted

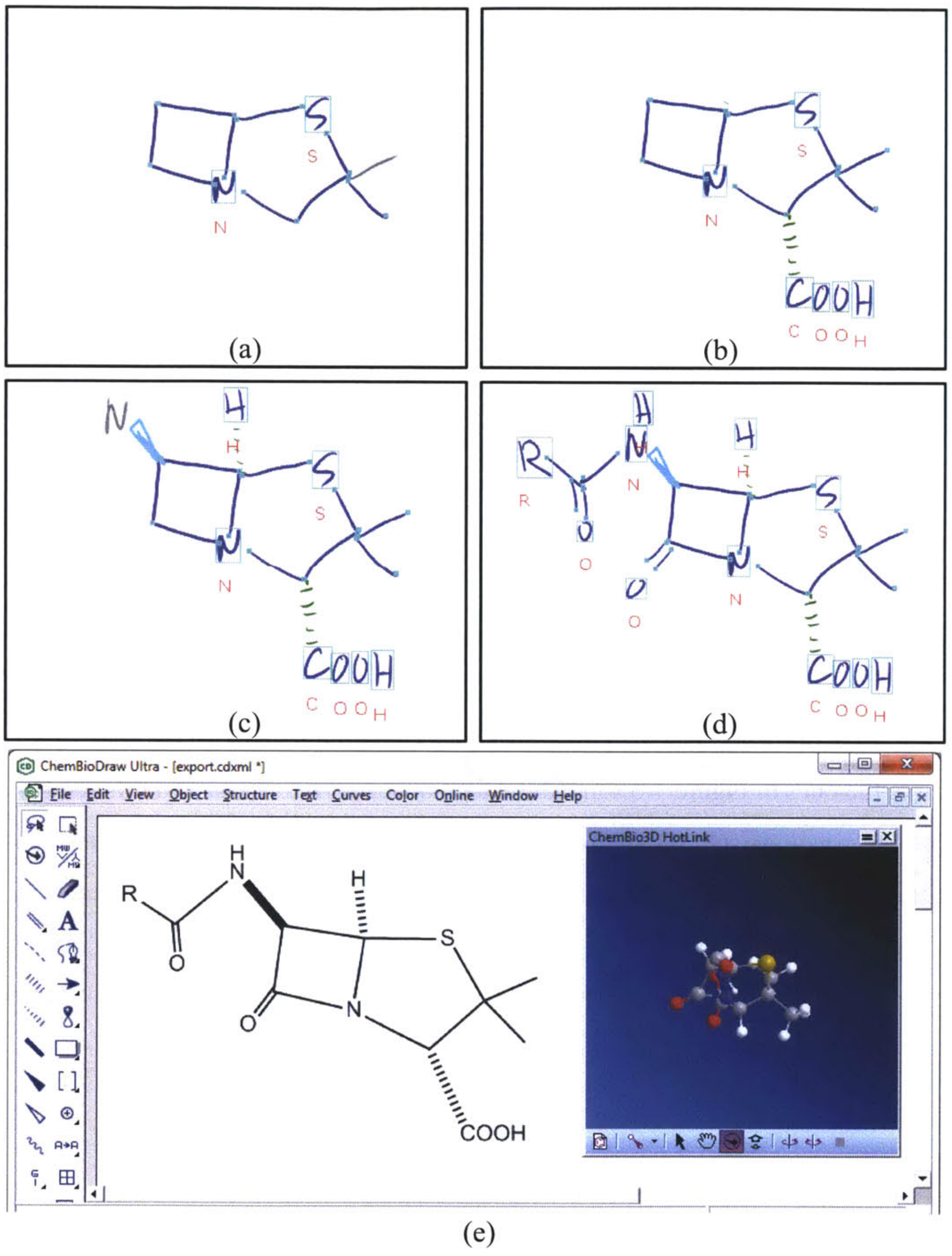
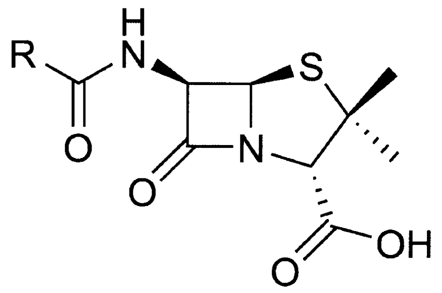

(a)
(b)
(c)
(d)
(e)
S
N
COOH
H
R
O
ChemBioDraw Ultra - [export.cdxml *]
File Edit View Object Structure Text Curves Color Online Window Help
ChemBio3D HotLink

Figure 1-4: Figures (a-d) show the progression of a chemical diagram being drawn using our system in real time. The interface automatically provides feedback about the recognition progress so far. Alphanumeric symbols are boxed in blue with letters below, straight bonds are shown in blue, wedge-bonds are colored light blue, hash-bond are colored green, and bond endpoints are indicated using small boxes. Figure (e) shows the user-drawn diagram exported and displayed in ChemDraw, a third party tool for analyzing chemical structures.

$R\text{-}C_9H_{11}N_2O_4S$

(Chemical Formula)



(Structure Diagram)

Figure 1-5: The chemical formula and structural diagram for the organic molecule Penicillin.

digitally can be readily exported to other software programs, making possible tasks like simulation, visualization, and database search. Furthermore, since the user's input is interpreted in real-time, this interface can provide constant feedback as the user is drawing, and can form a basis for new interactions such as sketch manipulation and error correction.

## 1.4 Chemistry and Analog Circuits

This work focuses on two real-world domains where sketches are especially widely used: chemistry and analog circuit design. When chemists need to describe the structure of a compound to a colleague, they typically do so by drawing a diagram. This is because the structure encoded in a diagram provides essential information about a molecule's identity, chemical properties, and potential reactions. They are, as a result, often considerably more informative than the chemical formula alone (illustrated in Figure 1-5).

Chemical diagrams like the one being drawn in Figure 1-6 are an essential part of the drug design process, and the depiction in image itself is actually very representative of real world useage: we have observed first-hand pharmaceutical research labs where not only the whiteboards but even the glass cubicle dividers were filled with these types of drawings. When people need to convey the same structure to a computer, however, they must re-create the diagram by hand using existing CAD based tools. These software for specifying chemical structures, such as the widely-used ChemDraw package, still rely on a traditional point-click-and-drag style of interaction, as shown in the top of Figure 1-2. These require

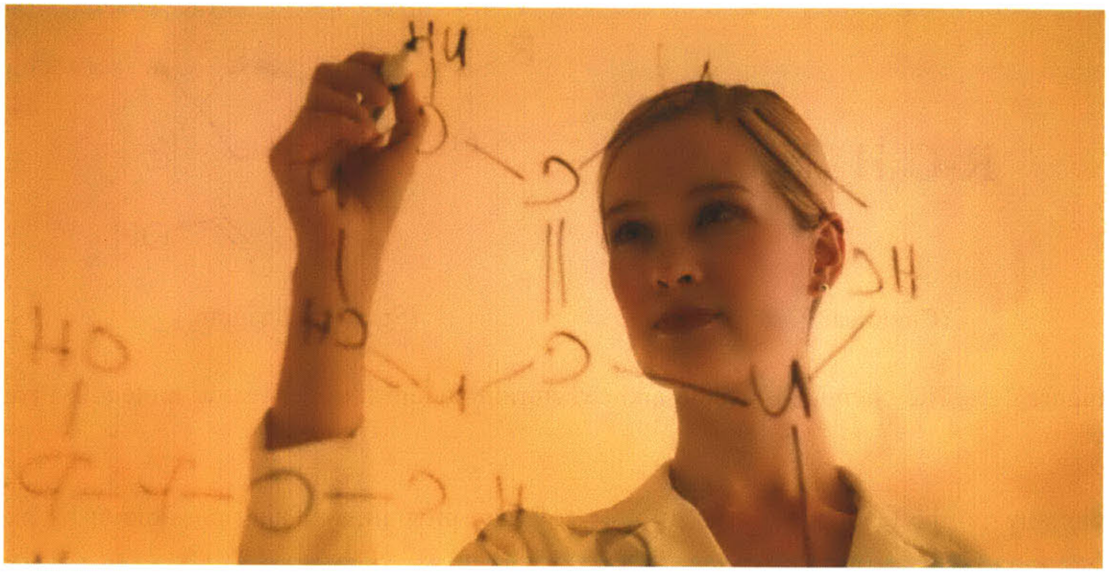


Figure 1-6: A chemist interacting with a drawing of a chemical structure.

users to first select a chemical component (e.g., a wedge or hash bond) from a toolbar before it can be added to the diagram. Experienced users can use keyboard shortcuts and other tricks to speed up the authoring process, but these require a significant learning curve.

Programs like ChemDraw are very popular because they allow users to create machine-readable diagrams that are easily exported to other chemistry software packages. However, they lack the ease of use, naturalness, and speed of simply drawing on paper. The work we present in this thesis takes a major step towards bridging this gap between how people naturally express drawings and how computers interpret them today.

In addition to chemical drawings, we have also applied our recognition architecture to the problem of understanding freehand drawings of analog electrical circuits. A circuit diagram, like the one shown in figure 1-1, is composed of a set of circuit components (such as resistors, capacitors, batteries, etc.) connected by a series of wires. In addition to the widespread use of these diagrams in classroom settings, the circuits domain is a popular one in the sketch recognition community and thus very useful in evaluating the performance and generalizability of our approach.

## 1.5 Contributions

The main contribution of this dissertation is a novel framework for sketch recognition that seamlessly combines a rich representation of local visual appearance with a probabilistic model for capturing higher level relationships. By "visual appearance" we mean a local image-based representation that preserves the pictoral nature of the ink. By "higher level relationships" we mean the spatial relationships between different symbols. The combination of these two sources of information is accomplished using a graphical model (a discriminatively trained conditional random field) that classifies each symbol jointly with its context, allowing neighboring interpretations to influence each other. This allows our method to be less sensitive to noise and drawing variations, significantly improving robustness and accuracy. The result is a recognizer that is better able to handle the range of drawing styles found in messy freehand sketches.

A second contribution is the development of a fast, robust, and accurate isolated symbol recognizer. In a departure from much of the earlier work in this area, this recognizer preserves the pictoral nature of the ink while at the same time taking advantage of the temporal patterns in the input strokes. This emphasis on visual appearance makes it more robust to stroke level differences like pen-drag and overtracing. We also present a symbol classification technique that is computationally efficient and robust to rotation and local deformations. We demonstrate that this method is able to exceed state-of-the-art performance for all three domains we evaluated: handwritten digits, PowerPoint shapes, and electrical circuit symbols.

Finally, another important contribution is the development of a prototype sketch based interface and its evaluation. To preserve the fluid process of sketching on paper, our interface allows users to draw diagrams just as they would on paper, using the same notations and conventions. In an on-line study our new interface was over twice as fast as the existing CAD-based method for authoring chemical diagrams, even for novice users who had little or no experience using a tablet. This is one of the first direct comparisons that shows a sketch recognition interface significantly outperforming a popular industry-standard professional CAD-based tool.

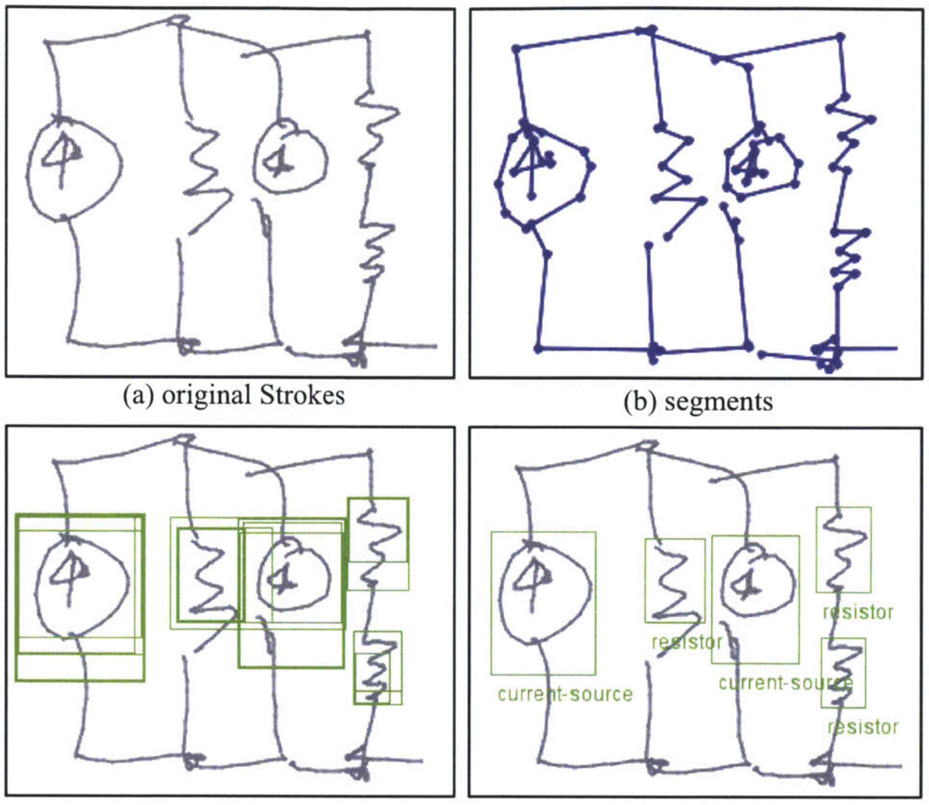


Figure 1-7: An illustration of the recognition process presented in this thesis. Frames consist of (a) an example sketch of a circuit diagram, (b) the segments after preprocessing, (c) a subset of the candidate groups extracted from the sketch, and (d) the final set of symbol detections.

Figure 1-7 illustrates the steps in our sketch recognition architecture. First the system decomposes the strokes into smaller segments (b) to facilitate recognition. It then searches for symbols among groups of temporally and spatially contiguous segments (c), evaluating each group to determine how likely it is to represent a valid symbol. Next it searches for an optimal set of symbol predictions from among these groups, taking into account both their visual appearance and their spatial relationships to each other. Finally, it displays the set of predicted symbols to the user in real time (d). While this example shows the process for interpreting an electrical circuit diagram, the same concepts apply for chemistry diagrams. This reflects one contribution of this thesis, which is developing a common recognition platform that can be used across multiple domains.

The remainder of the thesis is structured as follows. Chapter 2 describes our visual appearance based approach to isolated symbol recognition. Chapter 3 presents our framework for complete sketch recognition that combines appearance and spatial context. Chapter 4 presents an evaluation of our prototype interface, including a comparative user evaluation of our interface against the industry standard chemistry CAD program. Chapter 5 covers previous work in sketch recognition and how they relate to our approach. Finally, Chapter 6 summarizes the main ideas and contributions of this dissertation and discusses directions for future work.

# Chapter 2

# Isolated Symbol Recognition

## 2.1 Symbol Recognition in Context

A complete sketch recognition system needs to solve two important problems: locating the symbols in a sketch (i.e., symbol localization) and recognizing them (i.e., symbol recognition). This chapter will focus on the latter step: the problem of recognizing an isolated symbol assuming that its already been successfully localized from a sketch.

We begin by briefly discussing how the recognition step fits into the larger context of complete sketch understanding, and give some insights into how deeply the two are deeply intertwined. One possible approach is to treat these two problems independently: first parse the sketch into multiple regions that correspond to symbol locations and then use an isolated symbol recognizer to classify each of those regions. While this seems simple enough, a major limitation of this method is that it can be difficult to determine the correct symbol locations without detailed knowledge about the symbols themselves. Previous work has explored using hand constructed heuristics to predict symbol locations [13], but these heuristics can be difficult to scale to new symbols and new domains. Furthermore, decisions made during the detection stage can produce errors that are difficult to recover from in later stages, leading to mistakes that could have been prevented if the system had taken into account other sources of information earlier in the process.

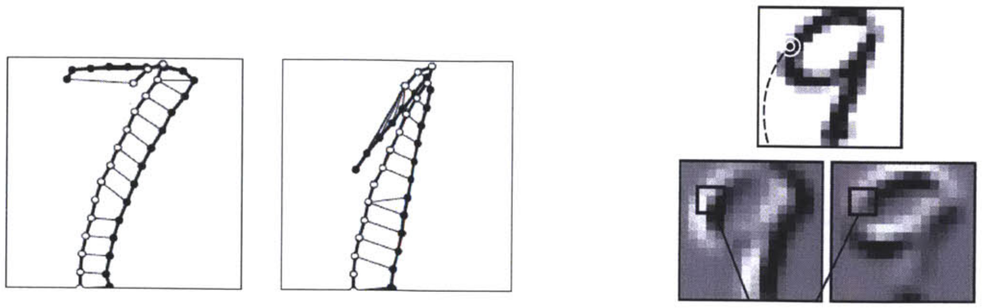
Our approach leverages the predictive power of the symbol recognizer to guide the symbol localization process. It does this by applying the symbol recognizer to a wide range of possible symbol locations, evaluating each candidate to determine how likely it matches one of the valid symbols in the domain. This means that the system does not require a separate set of heuristics or rules specifically for symbol localization, and improvements in symbol recognition should automatically lead to similar improvements in symbol localization (i.e., a more accurate recognizer would be better at distinguishing between locations that match valid symbols and those that don't).

Since symbol recognition is such an important part of the recognition framework, we dedicate this chapter to describing our solution to the problem. The key properties of our recognizer are summarized below:

- The recognizer represents symbols based primarily on visual appearance rather than geometric primitives or temporal patterns. This allows our approach to be more robust to differences in drawing style.
- At the same time, the recognizer also takes advantage of certain on-line stroke properties derived from the temporal sequence.
- The recognizer employs a classification technique that is computationally efficient and robust to rotation and local deformations.

### 2.1.1 Relation to Handwriting Recognition

Before we move on to the details of our symbol recognition approach, it's useful to take a quick look at the relationship between sketch recognition and handwriting recognition. First, an early motivation for our approach came from the observation that current off-line handwriting recognizers, which operate on scanned images, perform very well despite the fact that they lack any information about pen trajectories (see Figure 2-1). For example, state-of-the-art techniques are able to achieve error rates in the range of 0.5% on a corpus of 70,000 scanned digits [21]. While a direct comparison between on-line and off-line handwriting recognition is difficult, a survey of past literature suggests that off-line methods [21, 18] can perform as well as, or even better than, their on-line counterparts [8, 4, 24]. This lead us to ask, can advances in off-line handwriting recognition be adapted to make better on-line sketch recognizers?



Dynamic time warping
(Mitoma 2004)

Image deformation mapping
(Keysers 2007)

Figure 2-1: On-line and off-line handwriting recognition. The work on the left [24] focuses on temporal patterns in the input strokes while the work on the right [18] focuses on local intensity gradients in the image.

Second, unlike much of the existing work on sketch recognition, we designed our recognizer to handle handwritten characters as well as graphical shapes. This is important because letters and digits are often an essential part of sketched diagrams, where they may appear either as annotations (e.g., in electrical circuits) or as part of the underlying structure (e.g., in chemical diagrams).

## 2.2 Visual Symbol Recognition

Following this intuition we designed our approach to focus primarily on the visual properties of the symbols. However, unlike purely off-line methods, we also try to exploit the extra information we have about the temporal nature of the strokes, as listed below.

- **Stroke orientation**. The direction the pen is moving at each sample point in the stroke. This orientation is computed as the angle along the stroke between temporally adjacent points (relative to the x-axis).

- **Stroke endpoints**. The locations of the endpoints of strokes, where the user either lifts up or presses down the pen.

An overview of the recognition process is shown in Figure 2-2. The discussion below walks through each step of this process.

### 2.2.1 Input Normalization

The first step in our approach is to minimize the variations between different symbols due to pen sampling, scale, and translation. This improves the robustness of our recognizer to differences in symbol size, drawing speed, and location in the sketch.

Since on-line strokes are typically sampled at a constant temporal frequency, the distance between neighboring points in the pen trajectory varies based on the speed of the pen, producing more samples when the user draws slowly and fewer when she draws quickly. This phenomenon is especially prevalent in corners or regions of high curvature, where the pen speed is often much slower than usual. This spatial variability can make it difficult to make consistent and accurate orientation measurements, since the distance between adjacent points can vary dramatically. For example, in regions where the pen is moving slowly the distance between adjacent points may be very small, producing noisy measurements for the orientation. In order to make feature extraction more reliable we resample each stroke at a constant spatial frequency.

Next we remove differences due to scale and translation. A traditional solution to this problem is to transform all of the symbols so that they have the same bounding box dimensions, but we found this technique to be overly sensitive to artifacts like long tails at the ends of strokes or stray ink. In response, we normalize each symbol by translating its center of mass to the origin, and scaling it horizontally and vertically so it has unit standard deviation in both axes.

### 2.2.2 Feature Representation

A key part of our approach is that we convert the on-line stroke sequences into a set of low resolution feature images. We begin by computing five features for each sample point in the pen trajectory, four concerned with stroke orientation and one concerned with stroke endpoints.

- The four orientation features correspond to four reference angles, at 0, 45, 90, and 135 degrees. They measure how nearly horizontal, vertical, or diagonal the stroke is at each point. The feature values are calculated as the difference between the

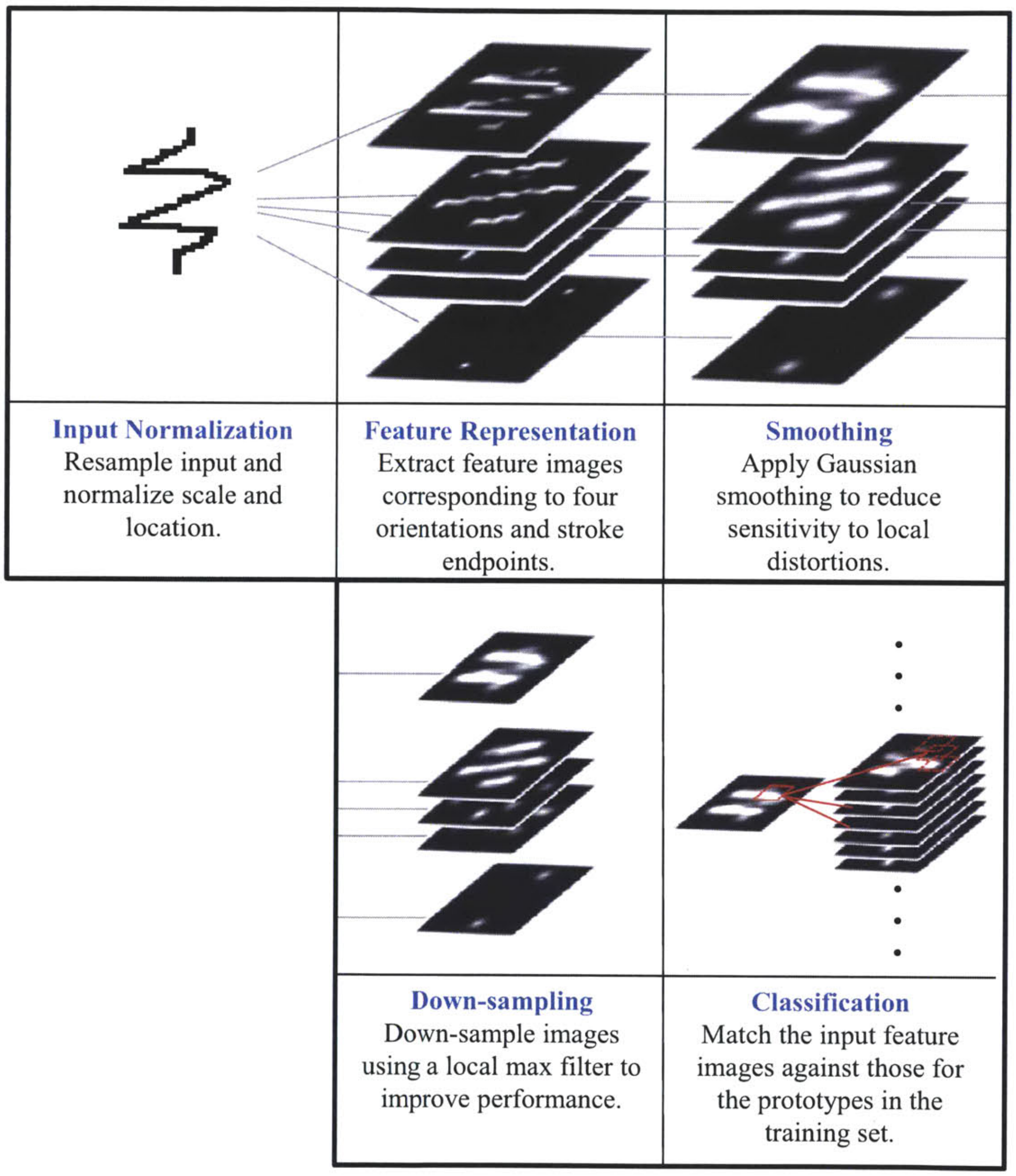

Input Normalization
Resample input and normalize scale and location.
Feature Representation
Extract feature images corresponding to four orientations and stroke endpoints.
Smoothing
Apply Gaussian smoothing to reduce sensitivity to local distortions.
Down-sampling
Down-sample images using a local max filter to improve performance.
Classification
Match the input feature images against those for the prototypes in the training set.

Figure 2-2: System Overview: First, a set of feature images representing the 4 orientations (top) and the endpoints (bottom) are extracted from the online stroke trajectory. Next, these images are smoothed and down-sampled to improve performance and increase tolerance to distortions. Finally, the images are compared against all of the prototypes in the training set.

stroke angle and the reference angle, and vary linearly between 1.0 (if the two are equal) and 0.0 (if they differ by more than 45 degrees). One major advantage of this representation is that it is independent of stroke direction (e.g., a stroke drawn left to right has the same orientation as one drawn right to left.)

- The endpoint feature identifies stroke endpoints. It is equal to 1.0 if the point is at the beginning or end of a stroke and 0.0 otherwise. This feature helps us distinguish between symbols like "3" and "8", which look similar but often differ in their endpoint locations.

The result is an ordered sequence of feature values, five for each point in the symbol. In order to preserve the spatial nature of the original input, we render these five features onto five 24 by 24 feature grids. The horizontal and vertical dimensions of the grid span 2.5 standard deviations of the original symbol's space in each direction. We call these grids "feature images", in which the intensity of a pixel is determined by the maximum feature value of the sample points that fall within its cell. For example, the intensity of the 0-orientation image in Figure 2-3 is high in regions where the stroke direction for a resistor symbol is nearly horizontal. This representation resembles the annotated images used by LeRec for handwriting recognition [5], but to our knowledge this is the first time it has been applied to sketched symbol recognition.

### 2.2.3 Smoothing and Downsampling

In the next stage we smooth and downsample the feature images to increase tolerance to local shifts and distortions. First we apply a Gaussian smoothing function to each image, spreading feature values to neighboring pixels. This ensures that small spatial variations in the symbol produce gradual changes in the feature values. We then downsample the images by a factor of 2 using a MAX filter, where each pixel in the downsized image is the maximum of the four corresponding pixels in the original. This further reduces sensitivity to small shifts and improves runtime performance.

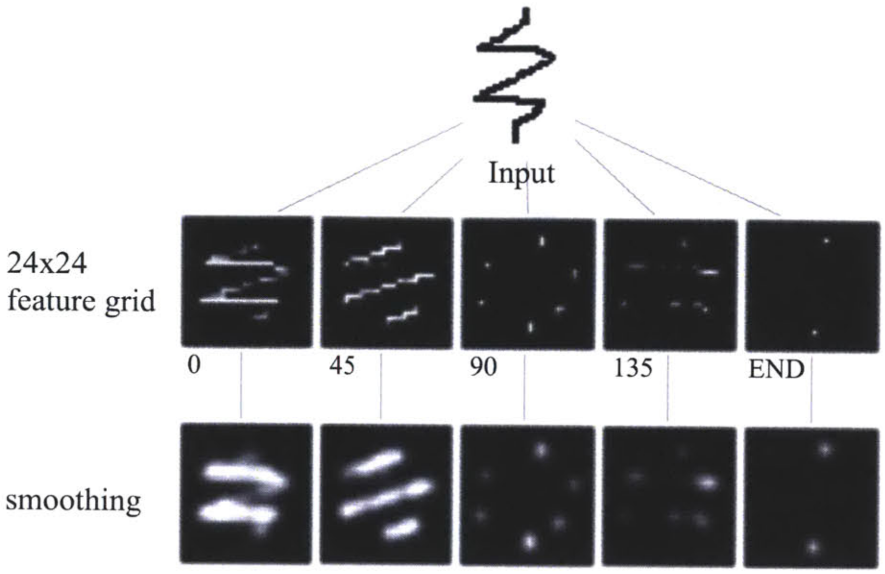


Figure 2-3: System Overview: First, a set of feature images representing the 4 orientations (top) and the endpoints (bottom) are extracted from the online stroke trajectory. Next, these images are smoothed and down-sampled to improve performance and increase tolerance to distortions.

### 2.2.4 Classification

For the symbol recognition task we use a deformable template matching algorithm that is robust to local shifts and distortions. Our image deformation model (IDM) allows every point in the input image to shift within a 3x3 local window to form the best match to the prototype image. The individual shifts are independent, so computing this displacement mapping is computationally efficient. To avoid overfitting, we include the local context around each point, i.e., we shift 3x3 image patches instead of single pixels. The distance between two points is then calculated as the sum of squared differences between the five feature images at their respective patch locations. An illustration of this process is shown in Figure 2-4.

The IDM distance between two symbols $I_1$ (the input) and $I_2$ (the template) is defined as:

$$D^2 = \sum_{x,y} \min_{d_x,d_y} ||I_1(x+d_x, y+d_y) - I_2(x,y)||^2 \tag{2.1}$$

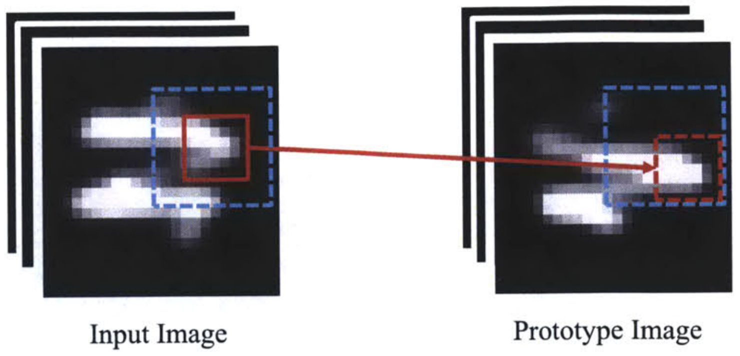


Figure 2-4: Image deformation model with local context matching. Each point in the input image can shift within a local window to form the best match to the prototype.

where $d_x$ and $d_y$ represent pixel shifts and $I_i(x, y)$ represents the 3x3x5 feature values in $I_i$ from the patch centered at $x, y$.

This deformation model is similar to the one proposed by [18] for off-line character recognition. Here, we extend their approach to on-line symbols using the orientation and endpoint features described earlier.

### 2.2.5 Performance Optimizations

One major limitation of the deformable template model is that it needs to match the input symbol against all of the training examples. As a result, computing the full set of IDM matches can take several seconds on even a modestly sized training set of 7000 templates. This section describes two performance optimizations that, whose combination can reduce this runtime by over two orders of magnitude.

**Coarse Candidate Pruning**

Since applying IDM to the full training set is too slow, the first optimization is to use a fast "coarse" metric to prune the set of candidates before applying the more expensive "exact" metric. In our implementation, this is equivalent to passing only the first $N$ nearest neighbors found by the coarse metric to the IDM matcher.

While simple Euclidean $L_2$ distance would work fairly well as the pruning metric, it would still involve comparing all 720 values in the feature images. We can improve performance even further by indexing these images using their first $K$ principal components. Then, we use the distance between these reduced feature sets to find the nearest candidates, as shown in equation (2):

$$\hat{D}^2 = \sum_{k=1}^{K}(v_1(k) - v_2(k))^2 \tag{2.2}$$

where $v_i(k)$ is the $k$-th principal component of the $i$-th image.

**Hierarchical Clustering**

The second optimization is a branch and bound technique to speed up the coarse classifier even further. It begins by applying agglomerative hierarchical clustering to the training examples in each class, organizing them into groups based on complete-link distance. This process first initializes each symbol into its own cluster, then progressively merges the two nearest clusters until there is only one cluster per class. At each step, it records the two sub-clusters that were merged to form the parent.

The result is a hierarchical tree structure with the largest clusters at the top and progressively smaller sub-clusters below. For each cluster and sub-cluster, it selects a representative prototype, the cluster center. This is defined as the example that is maximally similar to all of the other examples in the cluster. Next, it computes the cluster radius as the maximum distance between the center and any of its members. Figure 2-5 shows the result of this process on a collection of resistors. The leaves in the tree represent individual templates and the nodes represent cluster centers.

During inference, the algorithm compares the input symbol to the set of cluster centers, starting at the top level of the hierarchy. It keeps track of the best match discovered so far, discarding clusters when it knows they cannot improve this match. Since the Euclidean distance metric used for pruning follows the triangle inequality, the lower-bound on the best match in cluster $c$ is the distance to the cluster center $d_c$ minus the cluster radius $r_c$. If $d_c - r_c$ is larger than the best distance discovered so far, we can safely ignore the entire cluster. If not, the algorithm expands the cluster and repeats the process for its children.

Since we want to find the $N$-nearest neighbors, we need to make a couple of modifi-

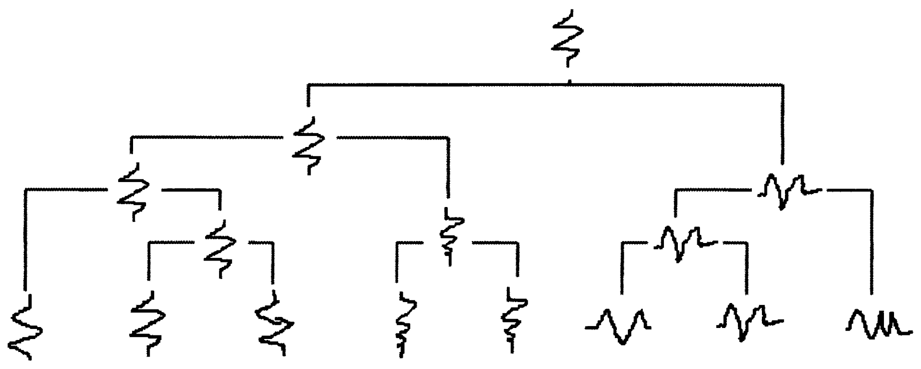


Figure 2-5: A hierarchical clustering tree for a set of resistors.

cations to the above algorithm. First, instead of keeping track of only the best match, we store a list of $N$-best matches. Second, we discard a cluster only if its lower bound distance is worse than the $N$-th best match discovered so far.[1]

### 2.2.6 Rotational Invariance

The recognition process described so far is robust to differences in translation, scale, and local deformation. The next step is to make it invariant to rotation. Our solution is to generate and match rotated versions of the input symbol to each of the training examples. We use 32 evenly spaced orientations from 0 to 360 degrees. To improve performance, in the hierarchical clustering stage we perform rotation matching only for the top level clusters.[2] For the lower level clusters, we can assume that all of the members are similar enough that they share the same rotation as the parent. Similarly, in the template matching stage, we assume that the optimal rotation found by the coarse metric is correct and reuse it in the exact match.

[1]In our implementation we use the first $K$=128 principal components and keep the first $N$=10 coarse nearest neighbor candidates. These parameters were chosen empirically; lower values degrade accuracy while higher values do not seem to offer any improvement.

[2]We use rotation matching for the top 64 clusters in each class.

## 2.3 Experimental Evaluation

We evaluated our approach on three datasets: handwritten digits, PowerPoint shapes, and circuit symbols. The following tables compare our performance against four benchmark classifiers (described below) and previous results reported in literature (shown in *italics*). Note that in all three evaluations we use the same optimized version of our recognizer, incorporating hierarchical clustering and PCA candidate pruning. The only exception is that we omit rotation invariance on the digits dataset because it would have made it impossible to distinguish between digits like "6" and "9".

**Benchmarks**

We include the following four benchmark classifiers:

- Pixel Nearest Neighbor (PIXEL NN): A baseline nearest neighbor classifier that uses the $L_2$ distance between the raw intensity images (no feature images).
- Feature Nearest Neighbor (FEATURE NN): A nearest neighbor classifier that uses the $L_2$ distance between the five feature images.
- Hausdorff Nearest Neighbor (HAUSDORFF): A nearest neighbor classifier that uses the Modified Hausdorff distance [9, 17]. This metric has been used previously for sketch and object recognition.
- SVM: A Support Vector Machine that uses a single 720-length feature vector to represent the five 12x12 feature images. We evaluated the performance using a LINEAR kernel and a RBF kernel.

### 2.3.1 Pen Digits

This dataset, first presented in [2], contains 10,992 isolated digits. It is divided into 30 writers for the training set and 14 writers for the test set, with no overlap between the two groups. Therefore, this evaluation is writer-independent and indicates how well our system is able to generalize to new users.

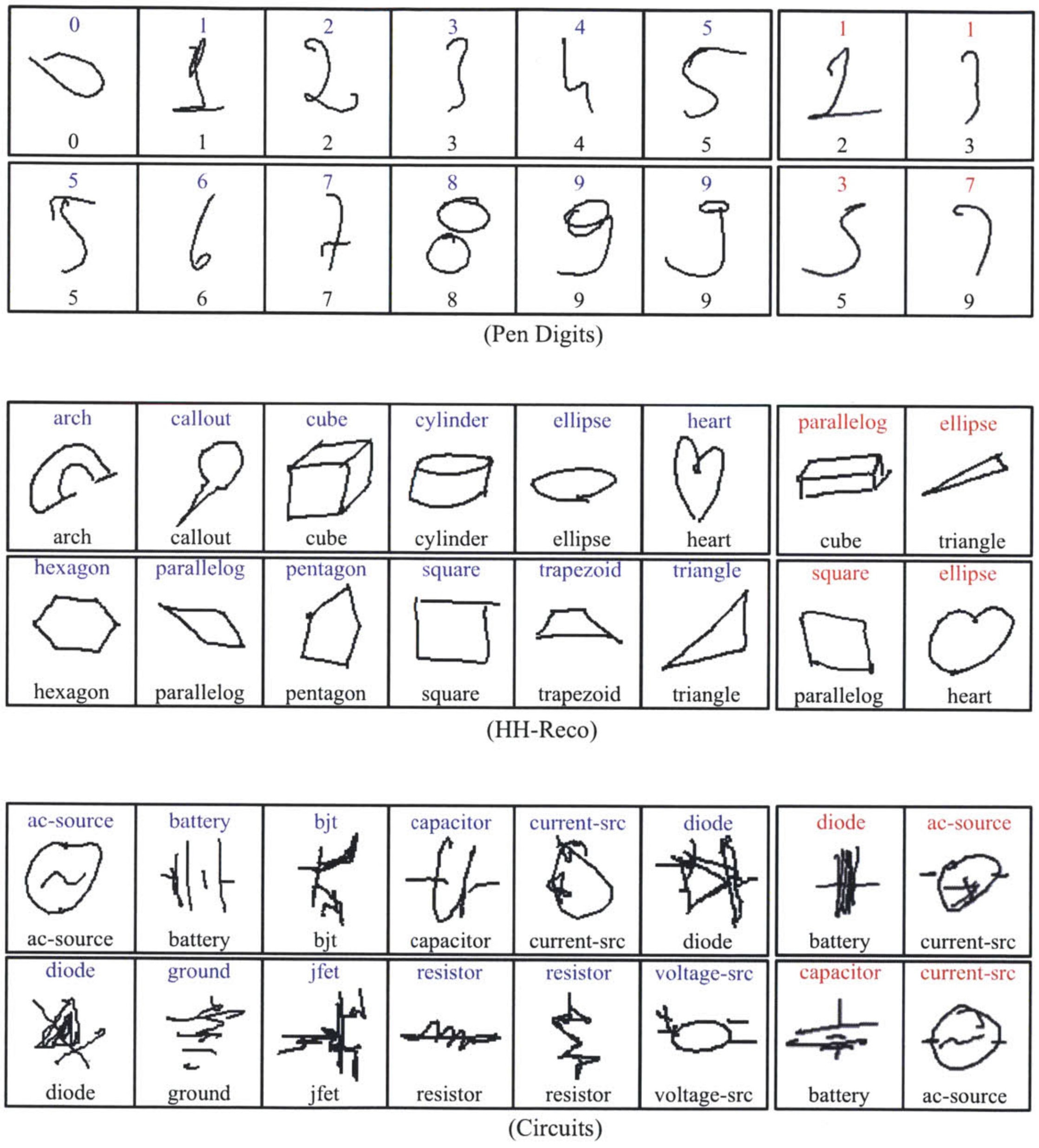

0 1 2 3 4 5 1 1
0 1 2 3 4 5 2 3
5 6 7 8 9 9 3 7
5 6 7 8 9 9 5 9
(Pen Digits)
arch callout cube cylinder ellipse heart parallelog ellipse
arch callout cube cylinder ellipse heart cube triangle
hexagon parallelog pentagon square trapezoid triangle square ellipse
hexagon parallelog pentagon square trapezoid triangle parallelog heart
(HH-Reco)
ac-source battery bjt capacitor current-src diode diode ac-source
ac-source battery bjt capacitor current-src diode battery current-src
diode ground jfet resistor resistor voltage-src capacitor current-src
diode ground jfet resistor resistor voltage-src battery ac-source
(Circuits)

Figure 2-6: Examples of symbols that our recognizer classified **correctly** (left) and **incorrectly** (right). The predictions made by our system are shown above each symbol and the ground truth labels are shown below.

As we see in Table 2.1, our method correctly recognized 99.2% of the digits in this corpus, outperforming both external benchmarks. Compared to [24], we were able to reduce the relative error rate by 56%, eliminating over half of the errors. The examples in Figure 2-6 (top) show that our method successfully handled a wide range of writing styles, and many of the symbols it missed could be considered difficult even for a human reader. Here the SVM-RBF model did slightly better than our method, but at the cost of greater computational requirements (see Table 2.4, below).

### 2.3.2 HHReco PowerPoint Shapes

The HHReco dataset [16] includes 7,791 PowerPoint shapes like boxes, ellipses, cylinders, and callouts. The examples were collected from 19 different people: each person drew at least 30 symbols per category. In our user independent cross-validation trials, we test on the examples from each user after training on the remaining 18 users.

As Table 2.2 shows, on this dataset we achieved an accuracy rate of 98.2%. Compared to the 96.7% result reported by [16], the best external benchmark, our method offers a 45% reduction in relative error. Figure 2-6 shows a number of examples that our system identified correctly and incorrectly.

### 2.3.3 Electrical Circuits

The Electrical Circuits dataset [27] contains 1,012 examples of circuit symbols like the ones in Figure 2-6 (bottom). Unlike the previous two domains, these symbols were extracted from complete sketches. As a result, they seem to exhibit a wider range of variations and drawing styles. We again evaluate using user-independent cross validation.

For this corpus our method achieved an accuracy rate of 96.2% (see Table 2.3). This represents a 64% relative error reduction over the best published benchmark of 89.5% [27]. As we can see in Figure 2-6, our method was able to correctly identify several very difficult examples, including ones that exhibit significant over-tracing and pen-drag.

| **Pen Digits Dataset** | **Accuracy** |
|---|---|
| SVM-RBF | 99.4% |
| OUR METHOD | 99.2% |
| SVM-Linear | 99.1% |
| FEATURE NN | 98.9% |
| *Eigen-Deformation [24]* | *98.2%* |
| Hausdorff | 97.6% |
| PIXEL NN | 97.1% |
| *Combination [2]* | *97.0%* |

Table 2.1: Comparison of recognition results for the Pen Digits dataset. Results listed in descending order of accuracy.

| **HHReco Dataset** | **Accuracy** |
|---|---|
| OUR METHOD | 98.2% |
| *Zernike Moments [16]* | *96.7%* |
| OUR METHOD (no rotation) | 95.2% |
| FEATURE NN | 95.1% |
| SVM-RBF | 95.0% |
| *Visual Parts [27]* | *94.4%* |
| Hausdorff | 93.0% |
| SVM-Linear | 92.3% |
| PIXEL NN | 92.2% |

Table 2.2: Comparison of recognition results for the HHReco dataset.

| **Circuits Dataset** | **Accuracy** |
|---|---|
| OUR METHOD | 96.2% |
| OUR METHOD (no rotation) | 93.9% |
| SVM-RBF | 91.9% |
| FEATURE NN | 91.2% |
| SVM-Linear | 90.1% |
| *Visual Parts [27]* | *89.5%* |
| Hausdorff | 79.9% |
| *Zernike Moments [16]* | *76.9%* |
| PIXEL NN | 75.9% |

Table 2.3: Comparison of recognition results for the Electrical Circuits dataset.

| **Runtime Performance** | **Time** |
|---|---|
| SVM-Linear | 2.4 ms |
| OUR METHOD | 8.1 ms |
| FEATURE NN | 40.8 ms |
| SVM-RBF | 90.3 ms |
| Hausdorff | 750 ms |
| OUR METHOD (unoptimized) | 3952 ms |

Table 2.4: The average time required to classify a symbol in the Pen Digits corpus.

### 2.3.4 Runtime Performance

Finally, we evaluate the runtime performance of the different classifiers presented in this chapter, measuring the average time required to recognize one symbol in the Pen Digits dataset (running on a 2.4 GHz machine). As Table 2.4 shows, our method is very computationally efficient, able to process over 100 symbols per second. Compared to the unoptimized version, it improves classification speed by over two orders of magnitude. This speedup is essential since our eventual goal is to achieve real time recognition, especially since we will be running this symbol recognizer over a large number of candidate locations within each sketch.

## 2.4 Discussion

In this chapter we presented a new visual approach to on-line symbol recognition. Unlike much of the previous work in this area, we represent symbols as visual feature images rather than as temporal sequences or geometric primitives. As a result, our method is less sensitive to the variations in drawing style that can pose major challenges for other sketch recognition systems. We also presented a classification technique that is computationally efficient and robust to rotation and local deformations. Finally, we showed that our method is able to exceed state-of-the-art performance for all the domains we evaluated, including handwritten digits, PowerPoint shapes, and electrical circuit symbols.

This work focused on developing a fast, accurate, and robust sketched symbol recognizer that works for multiple domains. However, symbols in real sketches are not drawn in isolation; neighboring symbols may be close together or even touching, and multiple symbols may be drawn using a single pen stroke. As mentioned in the beginning of the chapter, a complete recognition system needs to jointly address the problem of symbol localization and recognition. We cover our solution to this problem in the next chapter.

# Chapter 3

# Complete Sketch Understanding

## 3.1 Overview

In the previous chapter we presented an accurate and robust recognizer for isolated hand-drawn symbols. In this chapter we move on to the problem of localizing and recognizing symbols in the context of a complete sketch. This is a significantly more challenging problem because 1) symbols in a sketch may be drawn in any order, 2) they may be drawn close together or even overlapping each other, and 3) they may be composed of both complete and partial strokes.

At a high level, our approach is to apply our symbol recognizer to many different locations in the sketch (which we will henceforth refer to as candidates), predicting how well each candidate matches one of the valid symbol in the domain. Since only a small fraction of these candidate locations will correspond to actual symbols, the next step is to search for the subset of these candidates that produces the most likely interpretation for the sketch.

An overview of our approach is shown in Figure 3-1, and is composed of the following components:

- **Stroke Segmentation**: Decompose strokes into smaller segments to facilitate recognition. Segments are more robust than complete strokes, because working with segments allows different parts of a stroke to be classified independently. This is necessary because not all symbols are composed of complete strokes, so our system needs

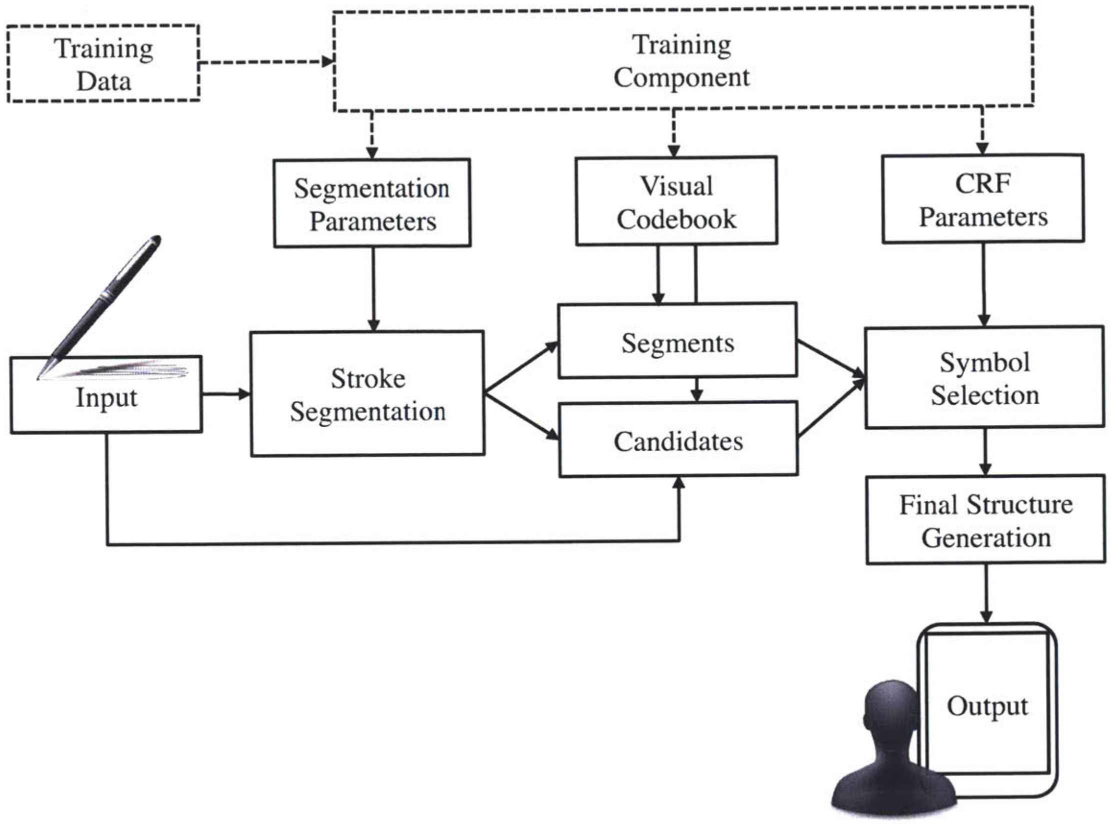


Figure 3-1: An overview of the sketch recognition system architecture presented in this thesis.

to consider candidates that may contain only parts of strokes (e.g., a wire and resistor drawn together without lifting the pen).

- **Candidates**: Extract a set of candidate locations from the sketch, each of which is a potential location for a symbol.
- **Visual Codebook (candidates)**: Apply the isolated symbol recognizer to each of the candidate locations, generating predictions for how likely each candidate represents a valid symbol.
- **Visual Codebook (segments)**: Apply the isolated segment recognizer (similar to the one used for candidates) to each of the segments extracted from the sketch, generating predictions for the likely labels for each segment.
- **Symbol Selection**: Combine the information from all of the candidate and segment

predictions, as well as spatial relationships between neighboring candidates and segments, to construct the optimal set of symbol detections.

- **Structure generation**: Predict the connectivity of the symbol detections to produce the final interpreted structure. The result is a machine-readable encoding of the diagram that can be handed off to other programs for analysis, search, publication, etc.

- **Training component**: The system uses training data to learn the segmentation model, visual codebook, and CRF parameters used for sketch interpretation. This training step needs to be performed only once, beforehand, in an off-line step.

## 3.2 Segmentation

The first step of our recognition architecture is stroke segmentation. These segments are generated by dividing strokes at corner points (Figure 3-2). In the chemistry domain corners have a special meaning because they determine the breaks between straight bonds. This is because chemists often draw multiple straight bonds using a single polyline stroke (Figure A-1), relying on the reader to infer that they are actually drawing multiple individual bonds connected by implicit carbons.[1] A more detailed description of this and other chemical drawing notations is provided in Appendix A.

Corner detection is a well-studied problem in sketch recognition. Previous approaches have explored looking at extremes in curvature and pen speed [34], temporal patterns in pen direction [31], and alternative approximations to stroke curvature [44]. These methods often use hand-coded thresholds to achieve good performance and custom heuristics to deal with common errors. Prior work has also focused primarily on finding well defined corners in isolated shapes, where there is a often a clear distinction between corners, curves, and lines. However, as seen in Figure 3-2, corners in real-world chemical drawings are often messy and unclear.

[1]Carbons and hydrogen atoms are so common in chemistry that they are typically left out of the drawing, and are assumed to be present anywhere that two bonds connect without an intermediate atom.

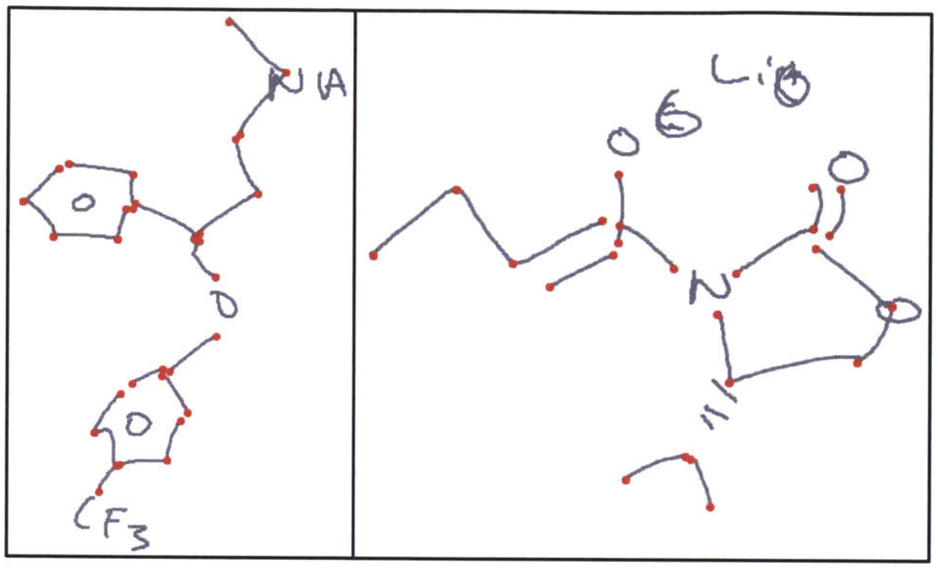


Figure 3-2: The result of our segment extraction algorithm on two chemical drawings. Detected corners are highlighted in red. Note that we only show corners from strokes that represent straight bonds.

Instead of immediately trying to decide which points are corners, our system instead repeatedly removes the point that is *least likely* to be a corner. This process stops when the system decides that all of the remaining points are likely to be corners. Specifically, our algorithm repeatedly discards the point $p_i$ that introduces the smallest cost when removed:

$$cost(p_i) = \sqrt{\text{mse}(\boldsymbol{s}_i; p_{i-1}, p_{i+1})} \cdot \text{dist}(p_i; p_{i-1}, p_{i+1}) \tag{3.1}$$

where $\boldsymbol{s}_i$ is the subset of points in the original stroke between point $p_{i-1}$ and point $p_{i+1}$ and $\text{mse}(\boldsymbol{s}_i; p_{i-1}, p_{i+1})$ is the mean squared error between the set $\boldsymbol{s}_i$ and the line segment formed by $(p_{i-1}, p_{i+1})$. The term $\text{dist}(p_i; p_{i-1}, p_{i+1})$ is the minimum distance between $p_i$ and the line segment formed by $(p_{i-1}, p_{i+1})$.

Under this framework the main challenge is to decide when to stop removing points, concluding that all remaining ones are intended corners. To address this challenge, we designed a novel corner detection algorithm that *learns* how to segment a stroke. Instead of forcing the developer to define thresholds and parameters beforehand, we train our corner detector from labeled sketch data. This allows our detector to learn a specific model of what it means to be a corner for chemical diagrams, which may be different from what

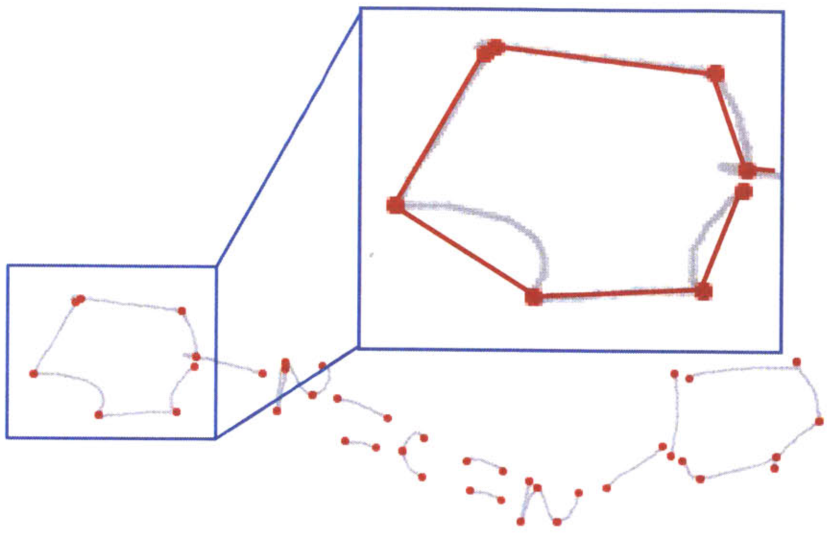


Figure 3-3: An illustration of the stroke segmentation process. Detected corners are highlighted in red. Note that the segments in the highlighted region form a 6-bond ring.

it means to be a corner in another domain. To the best of our knowledge this is the first trainable corner detector used as part of a complete sketch recognition system.

Our system predicts the likelihood of a vertex being a corner or not using a binary logistic regression model. For each vertex elimination candidate $\hat{p}$ (the point with the lowest cost) it extracts the set of features shown in Table 3.1. During classification, if the classifier decides that $\hat{p}$ is not a corner, it removes the vertex and continues to the next elimination candidate, repeating the process. If, on the other hand, it decides that $\hat{p}$ is a corner, the process stops and all remaining vertices are returned as corners. An overview of our segmentation approach is show in Figure 3-3.

One important feature of our approach is that in each iteration the system makes its decision based on the set of corner candidates that are still remaining, taking advantage of the partial solution generated so far. To illustrate this, consider the ring in the bottom-left of Figure 3-2, where there are two high-curvature points close to each other and only one of them is an intended corner (the other has high curvature due to noise, a common problem in corner detection since noise is easily mistaken for a corner). When both high-curvature points still remain in the polyline approximation, removing either one of them will not change the local shape by very much (i.e., have low cost). However, after one of them is

| **Feature** | **Description** |
|---|---|
| Cost | The cost of removing the vertex, from Equation 1. |
| Diagonal | The diagonal length of the stroke's bounding box. |
| Ink Density | The length of the stroke divided by the diagonal length. |
| Max Distance | The distance to the farther of its two neighbor ($p_{i-1}$ or $p_{i+1}$) normalized by the distance between the two neighbors. |
| Min Distance | The distance to the nearer of its two neighbors normalized by the distance between the two. |
| Sum Distance | The sum of the distances to the two neighbors normalized by the distance between the two. |

Table 3.1: List of features for corner detection.

removed, the cost of removing the remaining point becomes much larger. This leads to the correct behavior of eliminating only one of the points. Of course in our implementation the other features from Table 3.1 will factor into the decision, so this is an illustrative but much simplified description.

After segment extraction the system records the length of the longest segment in the drawing as $L$ (excluding the top 5% as outliers). This value is later used as an estimate for the scale of the sketch.

### 3.2.1 Segment Features

After the segments have been extracted from the sketch, the next step is to generate a set of features that describes the segments so that they can be used for classification. For each segment, we model the surrounding patch of ink using a set of rich local descriptors very similar to those used for symbols in Chapter 2. Since these descriptors are based on the same feature images as before, they are less sensitive to stroke level differences like stroke direction. We further make these descriptors invariant to scale by normalizing the size of the ink patch based on $L$, an estimate of the scale of the sketch, including both images of size $L$ and of size $2L$. We also incorporate a second pair of feature images that are invariant to rotation by reorienting them so that the orientation of the segment is horizontal. This dual

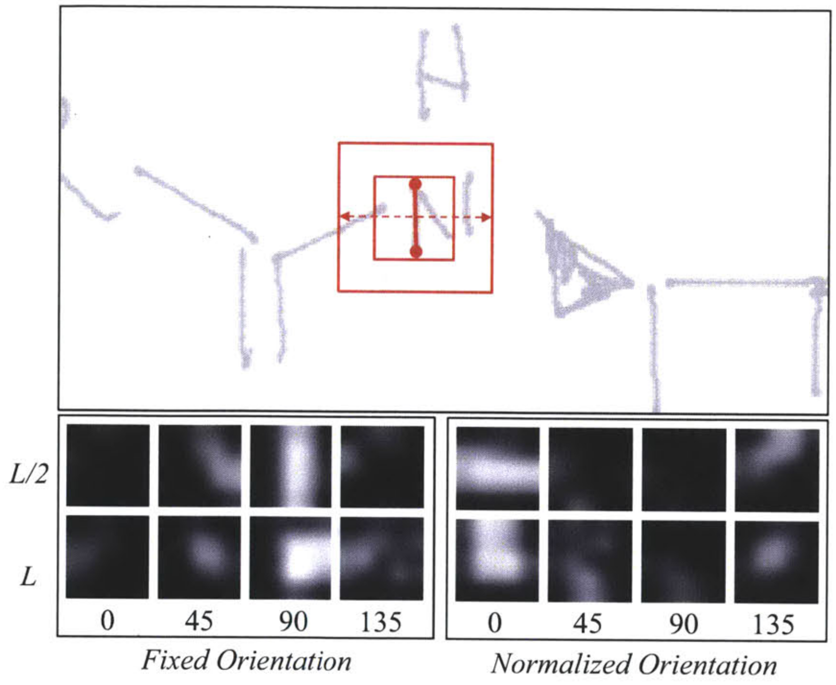


Figure 3-4: An illustration of the 4 sets of feature images used to encode the visual appearance of the region around a segment.

representation helps the system model both variable-orientation symbols like bonds as well as fixed-orientation symbols like elements and group abbreviations. An example of these features is shown in Figure 3-4.

The set of visual ink features are rendered onto sixteen 10x10 pixel feature images. We perform Gaussian smoothing on each image to improve robustness and reduce sensitivity to small distortions and noise. We then downsample each image by a factor of 2 (using a MAX filter as before) to a final size of 5x5 pixels to improve computation speed. The result is a set of sixteen 5x5 pixel images, producing a total of 400 feature values per segment. In addition to these feature images, we also use the set of geometric properties listed in Table 3.2 to further describe each segment.

## 3.3 Candidates

The second step of our process is generating and evaluating candidate locations that potentially correspond to intended symbols. We define a symbol as a group of one or more

| Feature | Description |
|---|---|
| Length* | The length of the segment. |
| Ink Density | The length of the stroke region matching the segment divided by the length of the segment. |
| Segment Count | The total number of segments in the parent stroke (discrete, ceiling=10). |
| Stroke Diagonal* | The diagonal length of the parent stroke's bounding box. |
| Stroke Ink Density | The length of the parent stroke divided by the diagonal length of the parent stroke's bounding box. |

Table 3.2: List of geometric features for segment classification. (*) means we include two version of this feature, one normalized by $L$ and the other unnormalized.

segments that represents a complete entity in the domain (e.g., bonds, elements, etc.). Our algorithm searches for symbols among groups of temporally or spatially contiguous strokes. It forms the set of temporal candidates by considering all possible sequences of up to $n = 8$ consecutively drawn strokes. It forms the set of spatial candidates by combining groups of strokes that are close to each other. An illustration of this component is shown in Figure 3-5. This process starts with all possible groups of size 2 (each stroke and its nearest neighbor) and successively expands each group by including the next nearest stroke (e.g., each stroke and its 2 nearest neighbors, then its 3 nearest neighbors, etc.). This expansion ends when either the size of the group exceeds a spatial constraint or when the group contains more than 4 strokes. This spatial grouping algorithm allows temporal gaps in candidates, so symbols need not be drawn with consecutive strokes.

The features we use for candidates encode the visual appearance of the candidate, based again on the feature image approach presented in Chapter 2. For each symbol we generate a set of five 20x20 feature images, four orientation filter images and one "endpoint" image that captures the location of stroke endpoints. These feature images contain only the strokes that belong to the candidate (unlike feature images in the segment level, which include all the ink in a local patch). In order to improve robustness to differences in aspect ratio, we stretch each candidate location so that it has the same standard deviation of ink in both the x and y axes. As before, we smooth and downsample each image by a factor of 2. An

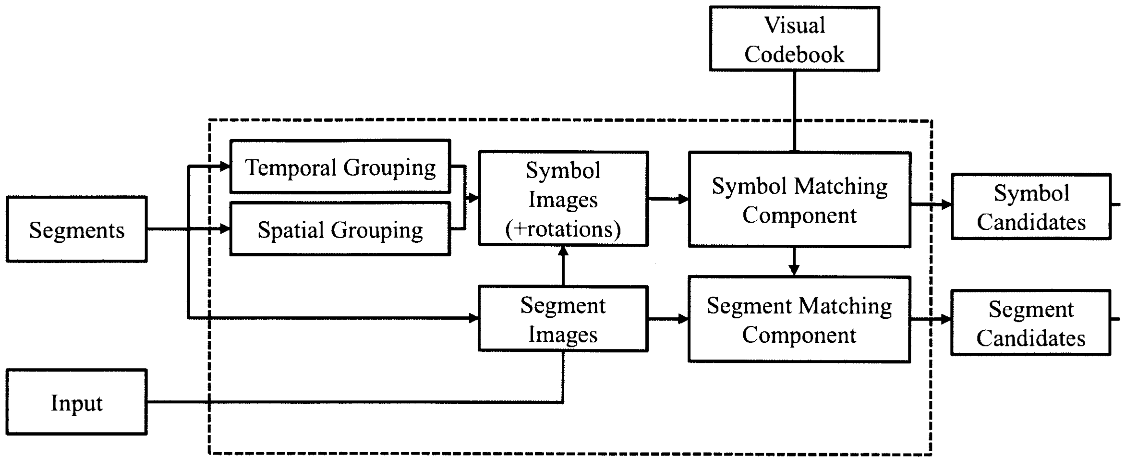


Figure 3-5: Visual Component: Generate symbols by grouping together segments that are temporally or spatially connected.

example is shown in Figure 3-6. Notice that the "S" (candidate 2) is stretched horizontally to ensure equal standard deviation of ink in both axes.

In addition to these five feature images, we include another set of four images that describe the ink in a patch around the candidate. These are identical to those used for segments, but are centered at the center of the candidate with a region size of $L$. The result is a total of 600 feature image values. We also include as features the set of geometric properties listed in Table 3.3.

| **Feature** | **Description** |
|---|---|
| Stroke Count | The number of strokes in the candidate (discrete, ceiling=10). |
| Segment Count | The number of segments in the candidate (discrete, ceiling=10). |
| Diagonal* | The diagonal length of the candidate's bounding box. |
| Ink Density | The cumulative length of the strokes in the candidate divided by the diagonal length of the candidate. |

Table 3.3: List of geometric features for candidate classification. (*) means we include two version of this feature, one normalized by $L$ and the other unnormalized.

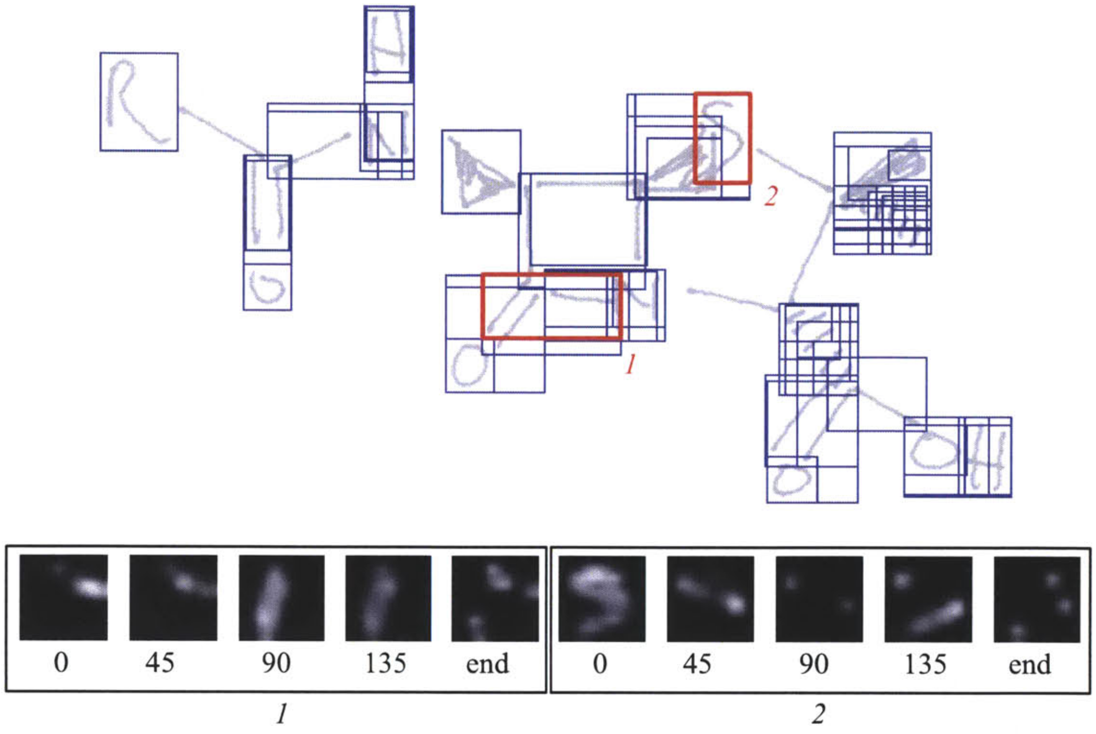


Figure 3-6: The set of candidates extracted from a chemical diagram, shown boxed in blue. The feature images generated for the two candidates highlighted in red are also shown below.

## 3.4 Feature Image Templates

In the sections above we described how the system generates sets of feature images for each classification entity (i.e., segments and candidates). However, we do not just use the image values directly as features for classification. Instead, we also compare the images against a set of stored templates taken from the training data and record the match distances to the nearest template neighbor in each class. In order to make matches at the candidate level rotation invariant we test 8 evenly-spaced rotations of the candidate[2]. Next, we convert these distances into match scores (score = 1.0 - distance) and use as features both the label of the nearest neighbor and the best match scores to each class. For example, a candidate whose nearest neighbor is an "N" (Nitrogen) symbol might have the following features: (nearest="N", score.N=0.7, score.H=0.5, etc.).

[2]Note that compared to Chapter 2 we reduce the number of rotations evaluated to improve runtime performance. We did not notice any significant performance cost for this optimization.

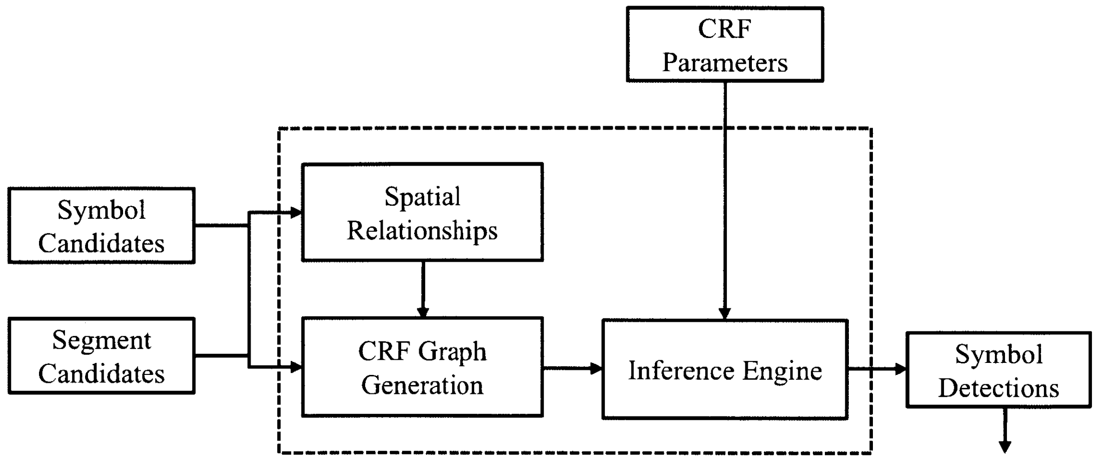


Figure 3-7: CRF Inference Component: The CRF inference component takes all of the candidates from the two levels (segments and symbols) and decides which candidates are actual symbols and which are mis-groupings. It does this using a probabilistic graphical model (a conditional random field, CRF) to encode the relationships between and among segments and candidates.

To improve the speed and memory usage of the template matching process described above, we use principal component analysis to reduce the dimensionality of the feature images for each entity to 256. For example, we compress the 400 image values from a segment to 256 principal components. We then calculate match distances based on these principal components rather than the original image values.

## 3.5 Candidate Selection using a Graphical Model

We propose a new model for sketch recognition based on conditional random fields (CRF) that combines the features from the three levels in the classification hierarchy. A CRF can be seen as a probabilistic framework for capturing the statistical dependencies between the different entities we wish to model (i.e., segments and candidates). An overview of this process is shown in Figure 3-7.

An alternative architecture is to train an independent classifier at each level, then use some type of voting scheme to combine the predictions. This approach has two disadvantages. First, by treating each layer in isolation it ignores any joint dependencies between features at different levels. Second, it requires the designer to specify a weighting scheme for each layer (e.g., deciding that the predictions in the candidate layer should be worth 2x those in the segment layer) either manually or by some separate learning process.

Figure 3-8 shows an illustration of our CRF graph structure. The nodes in the bottom row represent labels for segments ($V_s$) and each has as fixed set of possible labels, such as in the chemistry domain, "bond" (straight bond), "hash", "wedge", and "text". We found that we achieved better performance by combining the specific letters and abbreviations (e.g., "H", "O", "R") into a single "text" label during classification. When the classification is finished, they system converts the symbols recognized as "text" back to a specific letter abbreviation by using the identity of the nearest template match for the symbol (as described in the previous section).

The nodes at the top level represent symbol candidates ($V_c$). Notice that our model creates one candidate node for each segment rather than one for each candidate. This node contains, as possible labels, all of the candidates that the segment could belong to. At the end of the inference process the system chooses the best candidate for each segment and adds the candidate to the set of final symbol detections. For example, if the system decides that the correct label for $y_{c,2}$ (the candidate node for segment 2) is a "wedge" candidate containing segments [1,2,4], then the "wedge" candidate is added to the final symbol detections. Note that the candidate node labels can contain multiple interpretations of each candidate, so $y_{c,2}$ also has "hash" and "text" versions of candidate [1,2,4] as possible labels (the "bond" label is only applied to single-segment candidates).

The edges in our CRF model encode four types of relationships between nodes:

- **Entity features to label mapping**: We define $\phi_s$ as the local potential function that determines the compatibility between a segment's features and its label. This is analogous to a local classifier that classifies each entity independently of the others.

$$\phi_s(y_{s,i}, \mathbf{x}_i; \theta) = \sum_K f_{s,k}(y_{s,i}, \mathbf{x}_i)\theta_k \tag{3.2}$$

Here $\mathbf{x}_{s,i}$ is the set of features for segment $i$, $y_i$ is a potential label for the segment,

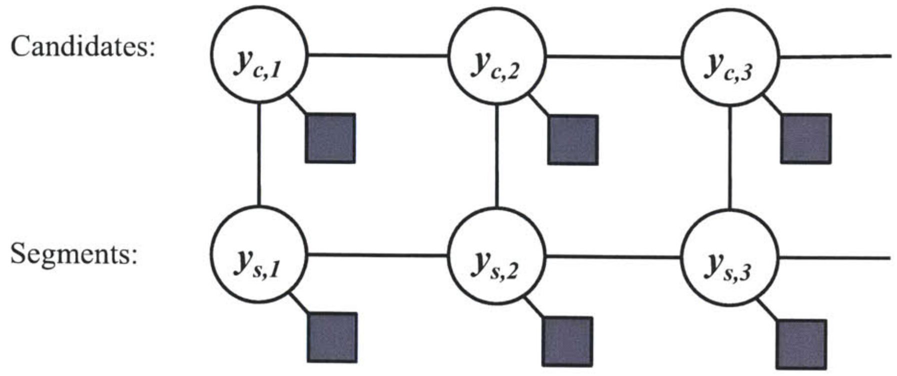


Figure 3-8: An illustration of our conditional random field model. Circles represent label nodes ($y$), edges represent relationships, and dark boxes represent evidence nodes ($x$) that connect the label nodes to their corresponding features.

and $f_{s,k}$ is a feature function defining the set of features for the given entity (e.g., the 256-valued feature image vector and geometric properties). Note that $\phi$ is linear to the parameters $\theta$, making the joint model (Equation 3.8 below) log-linear.

For the candidate nodes we have an analogous version of this local potential function. Note that here the situation is more complicated because the labels can map to different candidates. Therefore the feature function here depends on the candidate of label $y_{c,i}$.

$$\phi_c(y_{c,i}, \mathbf{x}_i; \theta) = \sum_K f_{c,k}(y_{c,i}, \mathbf{x}_i)\theta_k \tag{3.3}$$

- **Cross-level label consistency**: This is a pairwise constraint stating that predictions at each level need to be consistent with predictions at other levels. A segment and its parent symbol candidate should have the same label.

$$\psi(y_{s,i}, y_{c,j}) = \begin{cases} 0, & \text{if } y_{s,i} = y_{c,j} \\ -\text{inf}, & \text{otherwise} \end{cases} \tag{3.4}$$

- **Segment to segment spatial context**: This pairwise relationship captures the spatial compatibility between pairs of segments given their respective labels. It enables our system to classify each segment jointly with its context, allowing neighboring

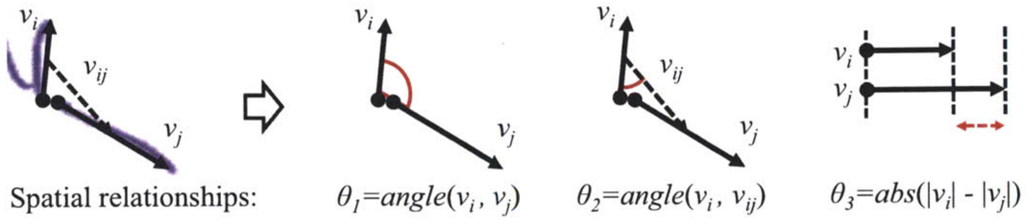

$v_i$
$v_{ij}$
$v_j$
Spatial relationships:
$\theta_1 = angle(v_i, v_j)$
$\theta_2 = angle(v_i, v_{ij})$
$\theta_3 = abs(|v_i| - |v_j|)$

Figure 3-9: The three pairwise relationships used in the spatial context compatibility between segments.

interpretations to influence each other. This relationship is active for all pairs of segments that are in close spatial proximity to each other[3].

$$\psi_s(y_{s,i}, y_{s,j}, \mathbf{x}_i, \mathbf{x}_j; \theta) = \sum_K f_{ss,k}(y_{s,i}, y_{s,j}, \mathbf{x}_i, \mathbf{x}_j)\theta_k \tag{3.5}$$

Here the feature function $f_{ss,k}$ contains the 3 spatial relationships shown in Figure 3-9. The system discretizes $\theta_1$ and $\theta_2$ into bins of size $\pi/8$ and $\theta_3$ into bins of size $L/4$.

- **Candidate to candidate spatial context**: This is a similar relationship that captures the spatial compatibility between pairs of symbol candidates given their respective labels. This pairwise relationship enables our system to classify each symbol jointly with the symbols around it, another important way of allowing neighboring interpretations to influence each other. It is active for all pairs of candidates that are in close spatial proximity to each other.

$$\psi_c(y_{c,i}, y_{c,j}, \mathbf{x}_i, \mathbf{x}_j; \theta) = \sum_K f_{cc,k}(y_{c,i}, y_{c,j}, \mathbf{x}_i, \mathbf{x}_j)\theta_k \tag{3.6}$$

- **Candidate to candidate overlap consistency**: This is a pairwise constraint that prevents the system from choosing two different candidates that share any of the same segments, resulting in conflicting interpretations for those segments. For example, if the system decides that the label of $y_{c,2}$ (the candidate node for segment 2) is a "wedge" candidate that spans segments [1,2,4], then the labels for $y_{c,1}$ and $y_{c,4}$ also

[3]We use $L/2$ as the distance threshold for both segment and candidate relationships in our implementation.

need to be assigned to the same "wedge" candidate. This relationship is active for all pairs of candidate nodes share any of the same candidate labels.

$$\psi_{cc}(y_{c,i}, y_{c,j}) = \begin{cases} 0, & \text{if } y_{c,i} = y_{c,j} \text{ or } y_{c,i} \text{ does not overlap } y_{c,j} \\ -\inf, & \text{otherwise} \end{cases} \tag{3.7}$$

Combining all of the relationships described above, the joint probability function over the entire graph is:

$$\begin{aligned} \log P(\mathbf{y}|\mathbf{x}, \theta) = \sum_{i \in V_s} \phi_s(y_{s,i}, \mathbf{x}_i; \theta) + \sum_{i,j \in E_{sc}} \psi(y_{s,i}, y_{c,j}) + \sum_{i,j \in E_{ss}} \psi_s(y_{s,i}, y_{s,j}, \mathbf{x}_i, \mathbf{x}_j; \theta) \\ \sum_{i \in V_c} \phi_c(y_{c,i}, \mathbf{x}_i; \theta) + \sum_{i,j \in E_c} \psi_c(y_{c,i}, y_{c,j}, \mathbf{x}_i, \mathbf{x}_j; \theta) + \sum_{i,j \in E_{cc}} \psi_{cc}(y_{c,i}, y_{c,j}) - \log(Z) \end{aligned} \tag{3.8}$$

where $E_{sc}$ is the set of label consistency edges from segments to candidates, $E_c$ is the set of spatial context edges from candidates to candidates, $E_{cc}$ is the set of overlap consistency edges, and $E_{ss}$ is the set of spatial context edges from segments to segments. $Z$ is a normalization constant.

### 3.5.1 Inference and Parameter Estimation

During training the system estimates the parameters $\theta$ in a maximum likelihood framework. The goal is to find $\theta^* = \text{argmax } L(\theta)$, where, following the previous literature on CRFs [20], we define:

$$L(\theta) = \log P(\mathbf{y}|\mathbf{x}, \theta) - \frac{1}{2\sigma^2}||\theta||^2 \tag{3.9}$$

Here the second term is a regularization constraint on the norm of $\theta$ to help avoid overfitting. We optimize $L(\theta)$ with a gradient ascent algorithm, calculating the gradient for each parameter $\frac{\delta}{\delta\theta_i} L(\theta)$. This process requires us to compute the marginals $P(y_i|\mathbf{x}, \theta)$. Since loops in the graph make exact inference intractable, we calculate these marginals using loopy belief propagation [25], an approximate inference algorithm. We employ a randomized message passing schedule and run the BP algorithm for up to 100 iterations.

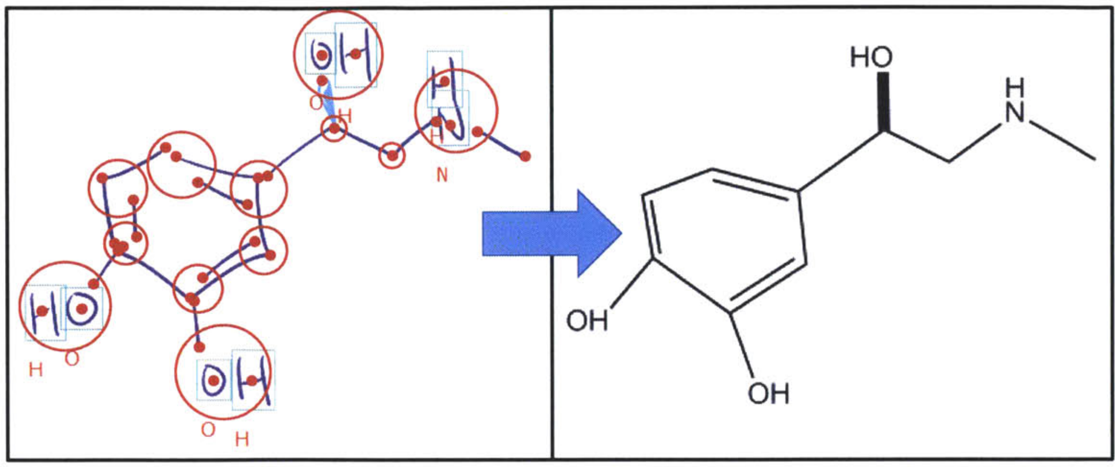


Figure 3-10: An illustration of the structure interpretation process: (left) an interpreted sketch with detected symbols highlighted and (right) the generated structure exported and rendered in ChemDraw. The connection points (for both bonds and symbols) and the final connected clusters are highlighted in red.

For gradient ascent we use L-BFGS [26], which has been applied successfully to other CRF-based problems in the past [36]. We use the same belief propagation algorithm during inference.

## 3.6 Structure Generation

After choosing the final set of symbol detections, our system builds a connectivity graph between the symbols to produce the complete structure. An example is shown in Figure 3-10. This symbol connectivity analysis is based on three pairwise distance metrics:

- Bond-element distance: The distance between a bond and an element is the distance from the bond endpoint to the nearest point in the element symbol. We impose an additional penalty if the bond does not point towards the element. For hash and wedge bonds, we define the direction of the bond as the principal axis based on PCA.

- Element-element distance: The distance between two letter symbols is defined as the distance between the two at their closest point.

- Bond-bond distance: The distance between two bonds is defined as the distance between their respective endpoints. We impose a penalty if the bonds do not point towards each other (e.g., if one bond is pointed to the midpoint of the other) or if they are nearly parallel (though parallel double bonds are technically connected to each other, we are interested in determining the elements to be joined at either of their endpoints).

We use an agglomerative clustering algorithm to generate the set of symbol connections. The algorithm iteratively merges the two nearest symbols or symbol clusters, using the maximum distance between the entities in the two groups as the clustering metric (i.e., complete-link). We empirically set the threshold to stop clustering at $0.4L$. Since as a general rule all symbols should be connected to at least one other symbol, the system reduces the distance value by a factor of two when it is considering whether or not to merge a singleton cluster. This encourages the algorithm to connect isolated symbols first and effectively lowers the threshold for single connections. The system is also able to incorporate domain specific constraints in its connectivity model. For example, in chemistry it imposes a penalty if the cluster makes connections that violate the rules of chemical valence (e.g., connecting three bonds to an "H", as Hydrogen should form only one bond).

### 3.6.1 Real-Time Recognition

Our system takes about 1 second to classify a sketch on a 3.7ghz processor running in a single thread. While this is likely sufficient for real time recognition, we also took steps to make sure that our system is fast enough to run on slower Tablet PCs. First, we implemented an incremental recognition model that updates the interpretation only of strokes and segments that have been modified or added since the last pass. Second, we made the most time consuming step of the process, generating features and template matches, parallel so that we could take advantage of multi-core CPU's. We also support the ability to erase strokes anywhere in the sketch by using the eraser end of the stylus or by selecting an eraser option on the interface. After each modification the system automatically updates its interpretation for the affected region. In our on-line user study a 1.8ghz Tablet PC was

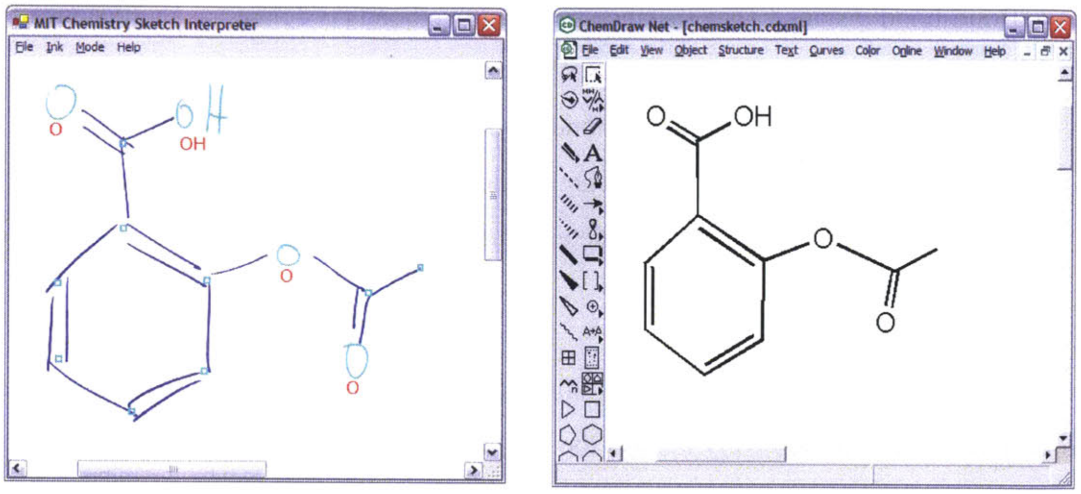

MIT Chemistry Sketch Interpreter
File Ink Mode Help
O
OH
O
O
ChemDraw Net - [chemsketch.cdxml]
File Edit View Object Structure Text Curves Color Online Window Help
O
OH
O
O

Figure 3-11: An illustration of our prototype real-time sketching interface. (left) A chemical structure drawn on a Tablet PC and interpreted using our system. Bonds are colored blue, elements are colored turquoise, and their interpretations are displayed in red. Turquoise boxes represent implicit carbons from bond-bond connections. (right) The same chemical structure exported to and redrawn by ChemDraw, a popular compound authoring tool.

able to easily keep up with the users' drawings. An illustration of our interface is shown in Figure 3-11.

# Chapter 4

# Evaluation

This chapter presents the evaluation of our sketch recognition architecture both in terms of recognition accuracy and usability, including a direct comparison between our interface and an industry standard mouse-and-menu-based CAD program. The major portion of this chapter focuses on the version of the system that is trained to recognize chemical diagrams, which we named ChemInk. We will also examine how well this same architecture can generalize to analog electrical circuit diagrams.

## 4.1 Off-Line Evaluation

We recruited 10 participants who were familiar with organic chemistry and asked each of them to draw 12 real world organic compounds (e.g., Aspirin, Penicillin, Sildenafil, etc.) on a Tablet PC. An example of one of the collected drawings is shown in Figure 4-1. We performed a set of user-independent performance evaluations, testing our system on one user while using the examples from the other 9 as training data. By leaving out sketches from the same participant, this evaluation demonstrates how well our system would perform on a new user.

Because one goal of our research is to build a system that can handle the range of drawing styles found in natural real world diagrams, the program used to collect these drawings behaved simply like a piece of paper, i.e., capturing the sketch but providing no recognition or feedback. This ensured that the system did not inadvertently provide

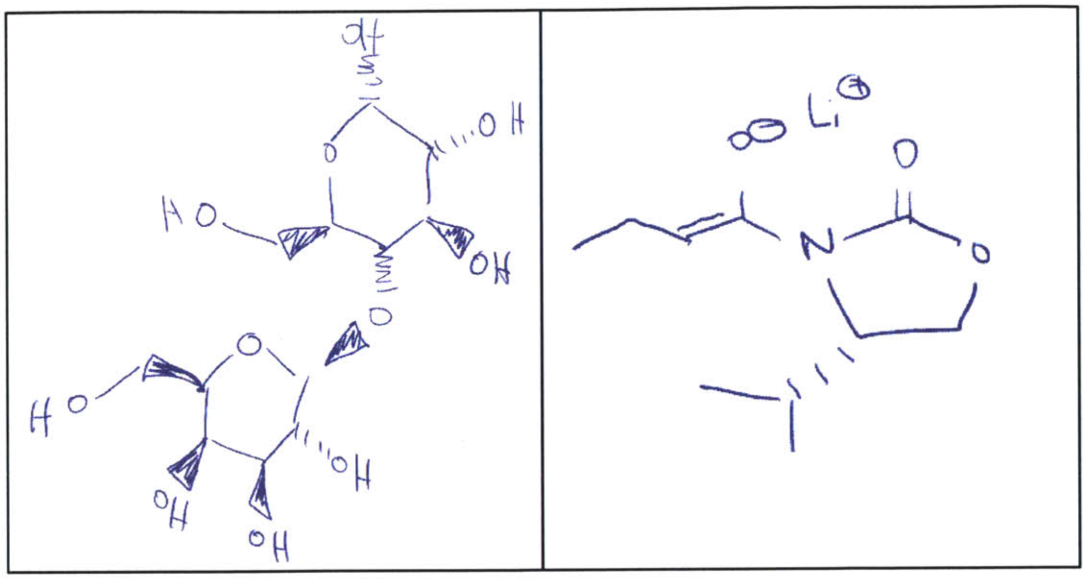


Figure 4-1: An example of a sketch collected and recognized by our system during the off-line evaluation. Text symbols are highlighted in green and hash and wedge bond are highlighted in light blue.

guidance in how to draw.

### 4.1.1 Corner Detection

We first evaluate the accuracy of our trainable corner detector in finding the correct corners in bond strokes, where they indicate the breaks between multiple straight bonds (as shown in Figure 4-2). For this task we used the dataset and user-independent evaluation metric described in the previous section. The results in Table 4.1 show that our algorithm was able to correctly detect 99.91% of the corners, with a precision of 99.85% (these measurements include stroke endpoints and single-segment strokes). For comparison purposes we also evaluated a version of our corner detector that does not use the learning based component. Instead, it uses a manually-tuned fixed threshold on the cost metric from Equation 3.1.

| **Method** | **Precision** | **Recall** |
|---|---|---|
| Trained detector | **0.9985** | **0.9991** |
| Fixed threshold | 0.9904 | 0.9879 |

Table 4.1: Evaluation of the corner detection component of our system. We only count corners in strokes labeled as bonds and compare against the hand labeled ground truth.

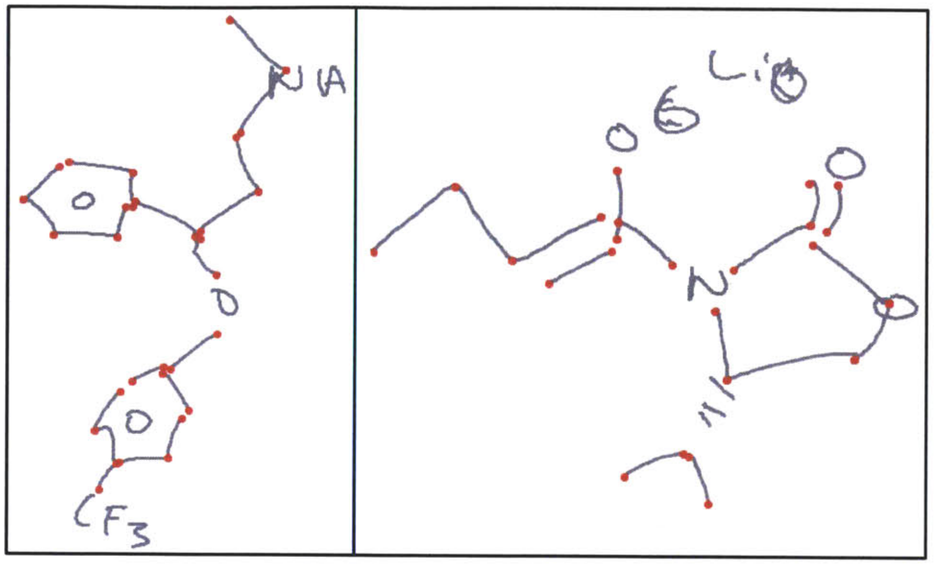


Figure 4-2: The result of our segment extraction algorithm on two chemical drawings. Detected corners are highlighted in red. Note that we only show corners from strokes that represent straight bonds.

### 4.1.2 Symbol Detection

The results in Table 4.2 indicates that our system was able to accurately detect and classify 98% of the symbols from the sketches in the dataset. As we showed in Figure A-1, these symbols consisted of straight bonds, wedge bonds, hash bonds, and textual abbreviations. Our result also shows that including context in the recognition architecture (i.e., the spatial relationships between segments and symbols) improves performance compared to a version that does not consider context (96.1%). While the increase in performance seems modest, it is worth noting that performance on the dataset was already very high and by adding these relationships we were able to remove 49% of the remaining errors.

| **Method** | **Precision** | **Recall** |
|---|---|---|
| ChemInk (context) | **.965** | **.980** |
| ChemInk (no context) | .948 | .961 |

Table 4.2: Evaluation of the recognition accuracy of our system. The (no context) version does not employ spatial relationships between segments.

While for completeness we report precision as well as recall, for this task we believe that recall (the fraction of true symbols detected) is a more appropriate metric than preci-

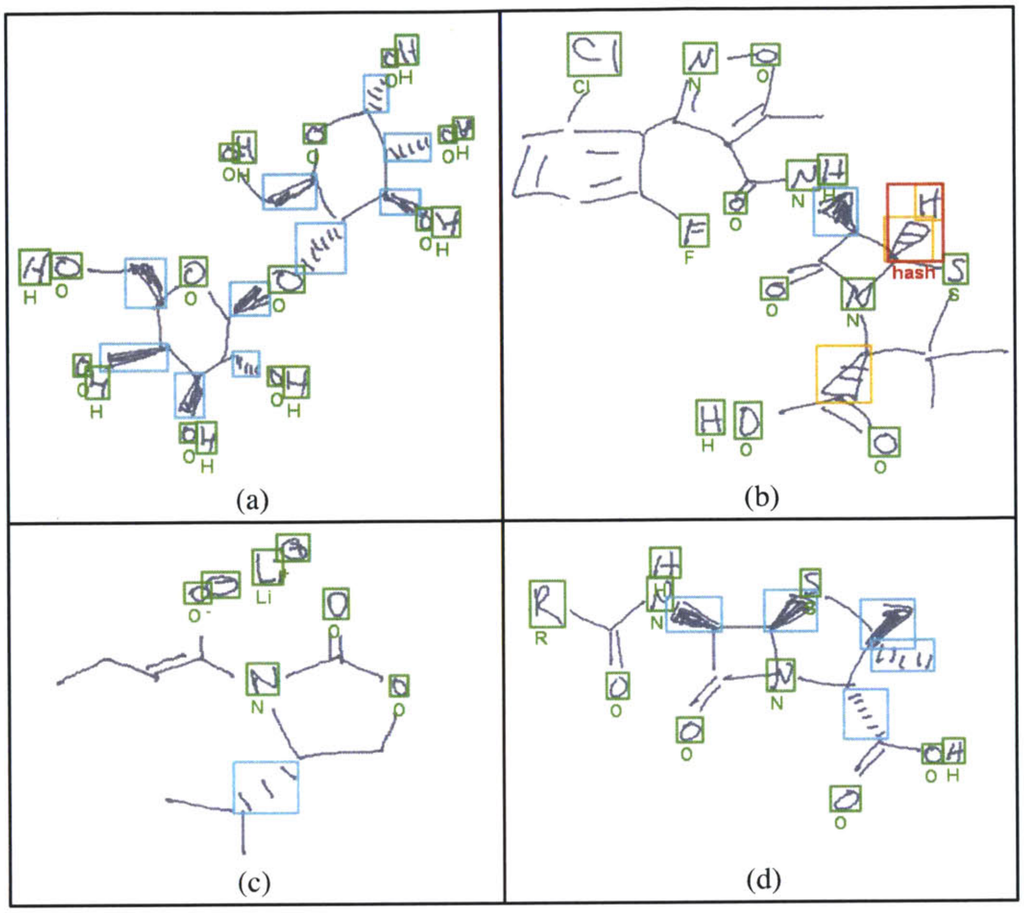


Figure 4-3: Examples of chemical diagrams recognized by our system. Correct detections are highlighted in green (for text) and teal (for hash and wedge bonds); errors are highlighted in red.

sion (the fraction of detections that are true symbols) because, unlike in traditional object detection, there are no overlapping detections (i.e., no two detections can share any strokes or segments) and every stroke is assigned to a symbol. Thus, a false positive always causes a false negative. Also, precision can be a less reliable metric because similar mistakes are not always counted equally. Misclassifying a 3-segment "H" as straight bonds, for instance, generates 3 false positives, while misclassifying it as a hash bond generates only one.

Some examples of the sketch recognitions generated in this evaluation are shown in Figure 4-3. Note that in sketch (b) the user drew hash bonds differently from all of the other users in the evaluation, enclosing them inside a triangular wedge. The resulting recognition errors are not surprising since the evaluation was user-independent, and the system had no

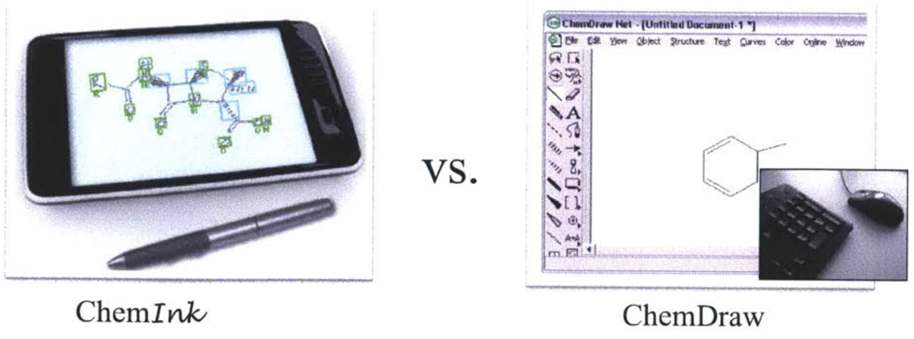


Figure 4-4: Comparative evaluation between our ChemInk interface and the industry standard CAD tool ChemDraw.

way of learning this particular user's special notation for hash bonds since there were no examples in the training data.

## 4.2 On-line Comparative Evaluation

We conducted a second user study to evaluate the usability and speed of our system, asking a number of chemistry graduate students to draw a set of five pre-selected diagrams on a Tablet PC[1]. Users were asked to draw in a natural manner, using the same styles and notations as they would on paper. While they were drawing, our recognition engine was running in real time and constantly providing feedback about the recognition progress by highlighting symbols detected so far. Users were asked to correct any errors the system made by simply erasing and redrawing the misinterpreted strokes.

We compared our system to an existing popular chemistry authoring tool called ChemDraw, asking users to produce the same diagrams using its traditional mouse-and-keyboard interface. We recorded each session and measured the amount of time taken to construct the diagrams using both interfaces. Afterwards we also asked the users for their opinions about how fast and easy it was to use each program.

[1]This study was conducted using an earlier version of our recognition engine, combining parts of our work in [28].

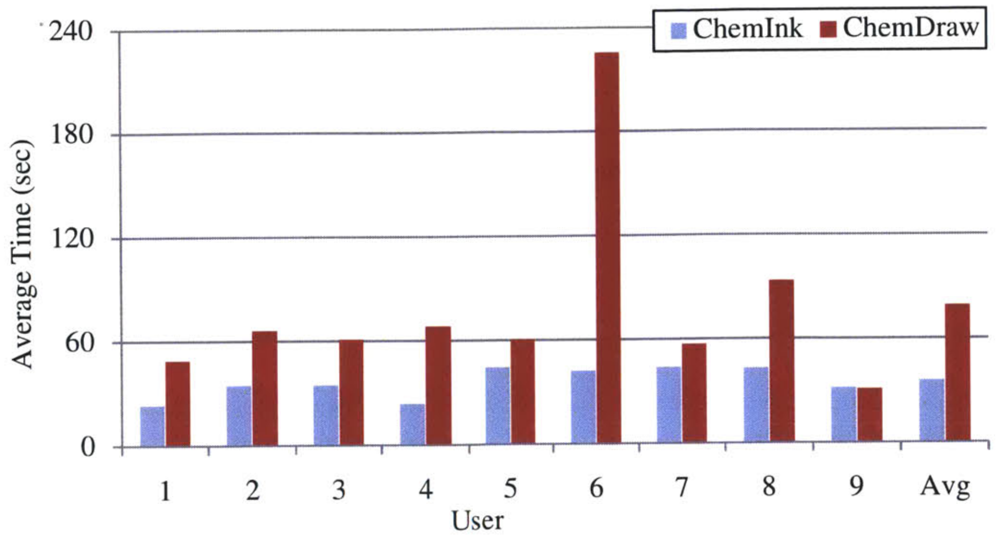


Figure 4-5: The average time taken by each of the study participants to draw a chemical diagram using ChemInk and ChemDraw.

### 4.2.1 Demographics

We had a total of 9 participants, all with prior experience with chemistry either through coursework only (1 user) or research only (1 user) or both (7 users). All of them had experience drawing chemical compounds on paper, rating themselves an average of 5.9 out of 7 on a scale of experience from novice (1) to expert (7). Most also had extensive prior experience using ChemDraw, rating themselves on average a 5.0 out of 7. Conversely, most had little or no prior experience using Tablet PCs, rating themselves an average of 2.2 out of 7.

### 4.2.2 Quantitative Analysis

Figure 4-5 shows the average time that the users took to complete a diagram using both ChemInk and ChemDraw. It shows that they were on average more than twice as fast using our ChemInk sketching interface, averaging 36 seconds per diagram, compared to ChemDraw's average of 79 seconds. The difference in drawing time between the two interfaces is statistically significant (paired one-sided $t$-test, $p < .05$). This was a very encouraging finding for us since many of the participants mentioned that they had years of experience using ChemDraw and use it daily in their research. This finding would also

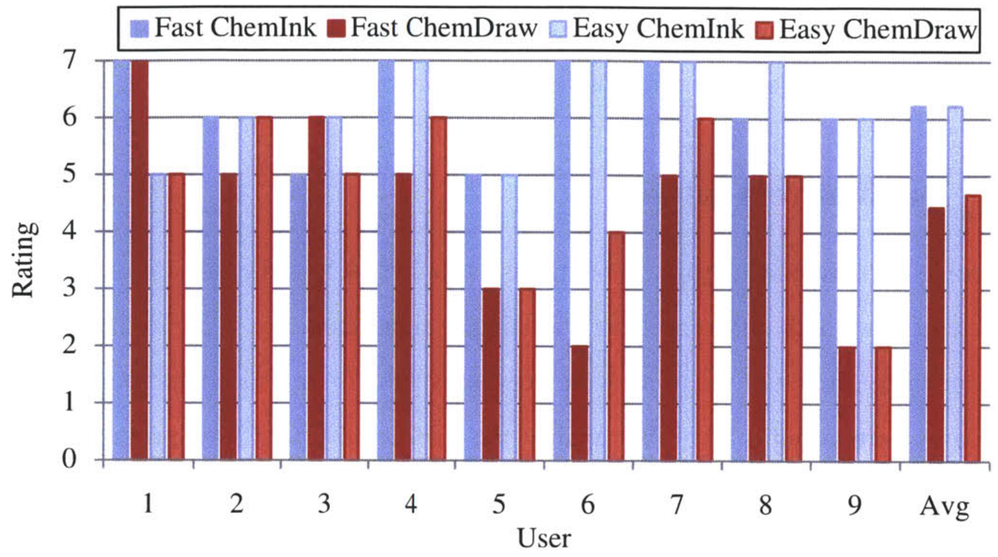


Figure 4-6: The ratings given by the study participants on how fast and how easy it was to use ChemInk and ChemDraw. Higher ratings indicate faster and easier.

likely surprise those users who did not rate ChemInk as being significantly faster in the subsequent survey (Figure 4-6).

As Figure 4-5 shows, User 6 had an especially difficult time using ChemDraw, taking on average over 3 minutes per sketch. To make sure that the outlier was not biasing our results we repeated the analysis with User 6 omitted. The average time spent per sketch becomes 35 seconds for ChemInk and 61 seconds for ChemDraw, showing that our interface is still nearly twice as fast, and the difference is still statistically significant ($p < .05$).

Figure 4-7 shows two timelines from one user, as he created the same diagram using ChemInk and ChemDraw. With our sketching interface, the user was able to complete the diagram in about 24 seconds, while he later took more than 1 minute to complete the same diagram using the ChemDraw. Note that this user rated himself a 7 out of 7 (maximum) for past experience using ChemDraw. Figure 4-8 shows some of the other sketch collected during this study along with our system's interpretation.

Not surprisingly, the two users with the lowest prior ChemDraw experience (both rated themselves 1 out of 7) were also the slowest ChemDraw users (#6 and #8), and there was a highly negative correlation between reported ChemDraw experience and time spent per sketch (corr = -0.738, $p < .05$). This suggests that prior training is very important to using

ChemDraw proficiently. In contrast, all of the users were able to use ChemInk effectively regardless of prior experience with Tablets PCs (corr = 0.111, $p > .05$).

### 4.2.3 Qualitative Analysis

On average users rated ChemInk as being faster (6.3 vs. 4.5) and easier to use (6.3 vs. 4.7) than ChemDraw (both differences are statistically significant, $p < .05$). In their comments about our system most of the users were very satisfied with the speed and performance, in many cases comparing it favorably against the traditional ChemDraw program. Some of the comments were:

*Awesome!*

*The program was very good at recognizing my drawing even though I have fairly messy handwriting...*

*In classroom setting, ChemDraw is too slow, whereas this is almost as fast as paper for taking notes.*

Some also had suggestions for making it faster:

*It would be even faster if you have predrawn sections that you could insert...*
*...if other letters or abbreviations were recognized, it would be even faster (for example recognizing that ph = [phenyl group]).*

One user made the interesting comment that there is something fundamentally different about sketching vs. click-and-drag:

*I like drawing structures by hand rather than in ChemDraw. Just like w/ typing hand-written notes are easier to remember / visualize / understand. Ability to sketch in 3D is very important too. great work guys! :)*

The comment that hand-written diagrams being easier to remember and understand may suggest some relationship to muscle memory in the learning process, and is in line with some of the prior work by Sharon Oviatt [30] on how different types of interfaces (e.g., keyboard vs. pen based) affect education efficiency.

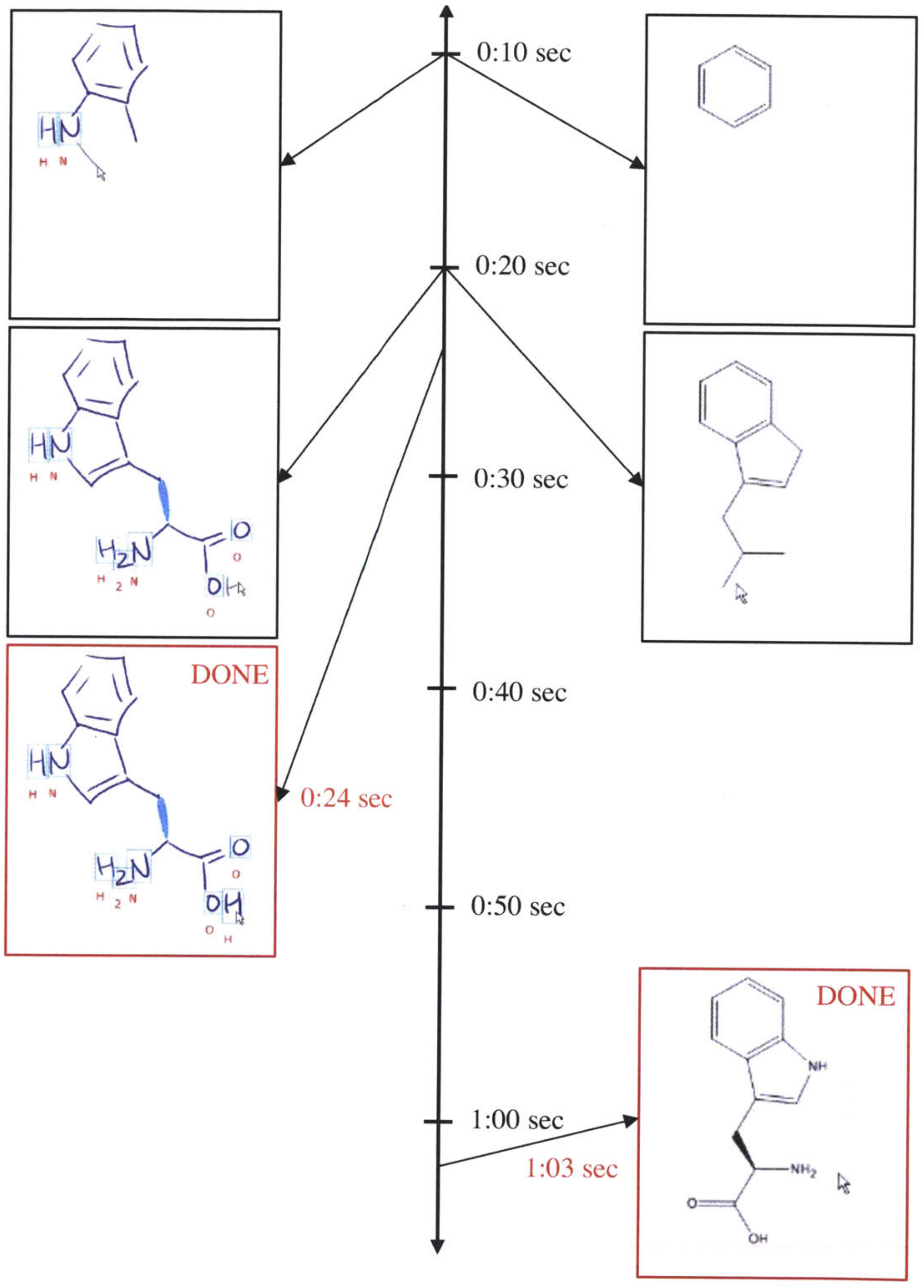


Figure 4-7: Timeline of the same user creating the same sketch using both ChemInk and ChemDraw. Note that user rated himself an expert (7/7) for experience using ChemDraw.

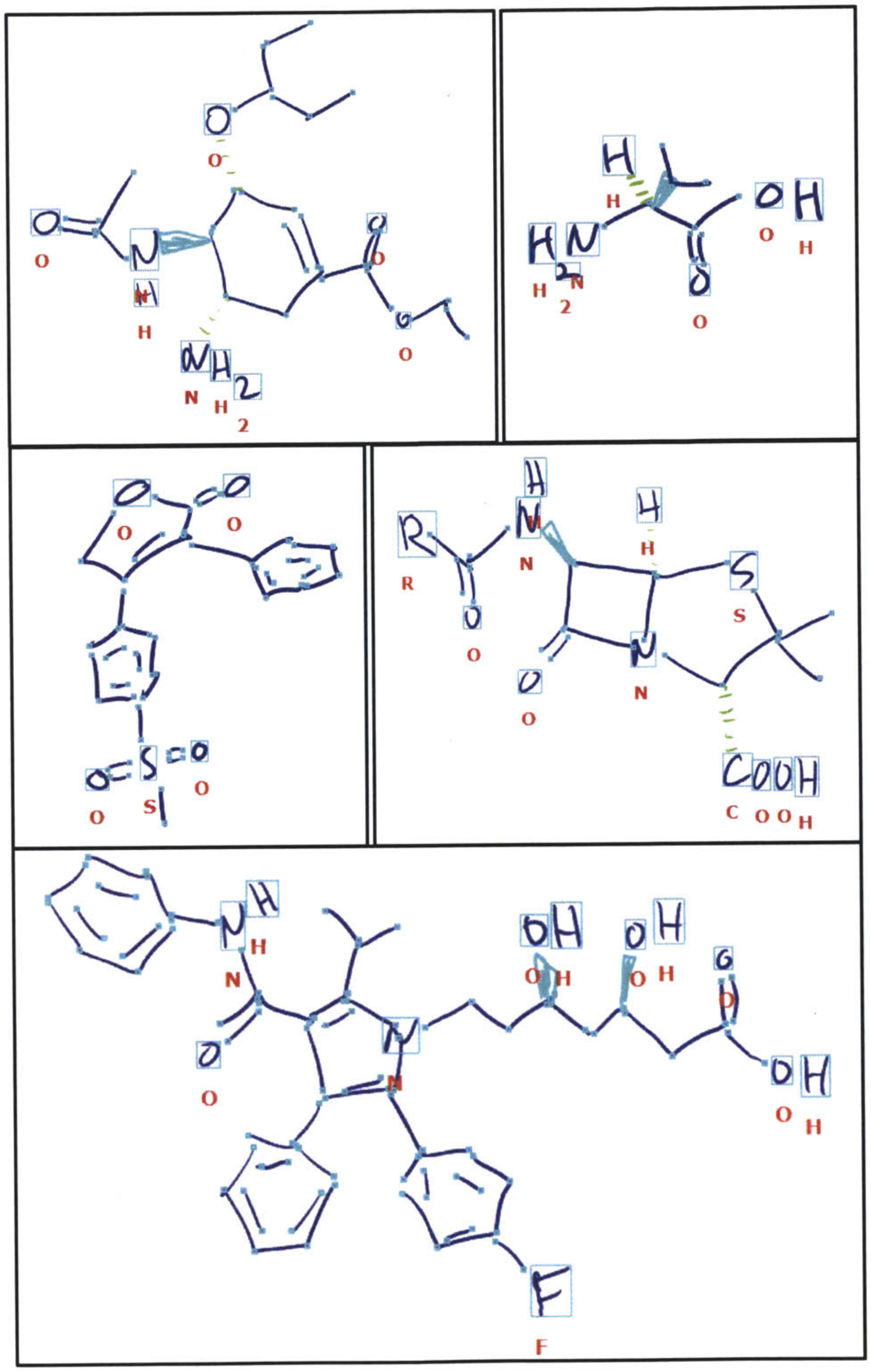


Figure 4-8: Examples of sketches collected from the on-line user study. The system's interpretation is highlighted as: text = green blue with letters below, straight bonds = blue, wedge-bond = light blue, hash-bond = green, bond endpoints = small boxes.

## 4.3 Electrical Circuit Drawings

So far this chapter has focused on chemical diagrams, but another goal of this thesis is to develop a set of sketch recognition algorithms that can be applied to multiple domains. As part of this goal, we applied the same approach described so far to build a symbol detector for electrical circuit diagrams, improving upon the best existing benchmark for that dataset.

For this evaluation we used a set of circuit diagrams collected by Oltmans and Davis[27]. The examples were collected from 10 users who were experienced in basic circuit design. Each user drew ten or eleven different circuits, and every circuit was required to include a pre-specified set of components. In this study users were not given specific instructions about how to draw each symbol and were free to lay out their drawing any way they wished. A more detailed description of the symbols in this domain is provided in Appendix A.

We again performed a set of user-independent performance evaluations. Because the exact locations of the ground truth labels are somewhat subjective (i.e., it is not obvious whether the resistor label should include the short wire segments on either end), we adopt the same evaluation metric used in the Pascal Challenge [10] and in [27]: a prediction is considered correct if the area of overlap between its bounding box and the ground truth label's bounding box is greater than 50% of the area of their union. Also, since we do not count wire detections for this dataset (as in [27]), we report precision as well as recall.

| **Class** | **Precision** | **Recall** |
|---|---|---|
| ac-source | 0.732 | 0.938 |
| battery | 0.768 | 0.778 |
| bjt | 0.786 | 0.786 |
| capacitor | 0.814 | 0.787 |
| current-source | 0.816 | 0.909 |
| diode | 0.875 | 0.939 |
| ground | 0.955 | 0.909 |
| jfet | 0.840 | 0.618 |
| resistor | 0.965 | 0.962 |
| voltage-source | 0.976 | 0.870 |
| **Our method** | **0.904** | **0.900** |
| *Oltmans 2007 [27]* | 0.257 | 0.739 |

Table 4.3: Overall recognition accuracy for the circuit diagram dataset.

The results are given in Table 4.3. They show that our method was able to recognize 89.3% of the circuit symbols correctly. As Figure 4-9 shows, this is a very messy dataset with significant variations in how symbols are drawn (e.g., overtracing, pen drag). On this dataset Oltmans and Davis [27] were able to achieve a best recall of 73.9% at a precision of 25.7%. Relative to their reported results, we were able to achieve a much higher recall and more than triple the precision.

## 4.4 Discussion

The fact that our sketch recognition approach was able to perform well for both chemical diagrams as well as electrical circuit diagrams, with only a change in the training data, is a very encouraging result. This suggests that the ideas presented in this thesis may be applicable to other domains as well. A deeper exploration of the limits and extensibility of this approach (e.g., to domains like flow-charts and course-of-action diagrams) is a promising area for future work, discussed in more detail in Chapter 6.

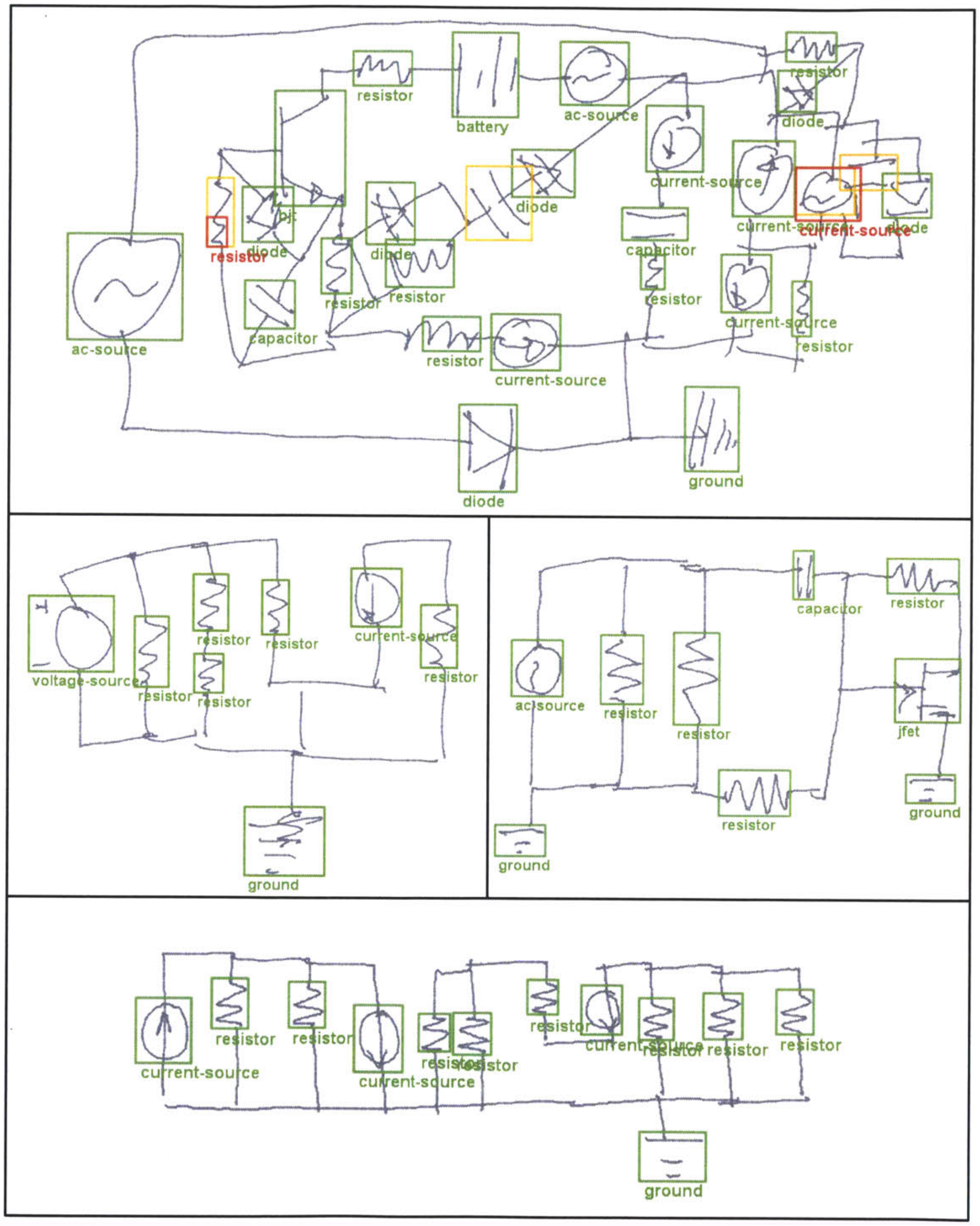


Figure 4-9: Examples of circuit diagrams recognized by our system. Correct detections are highlighted in green and false positives in red. False negatives are highlighted in yellow.

# Chapter 5

# Related Work

The field of sketch recognition started with Ivan Sutherland's seminal work on SketchPad [39] and his vision of how computers can become an essential part of the design process - through the medium of line drawings rather than written statements. In it he demonstrated how to use a light pen drawing on a monitor to communicate shapes to a computer, including mechanical parts and electrical circuits. The work presented in this thesis follows very much the same vision and spirit, and advances the state of the art by showing that a natural drawing interface (one that allows users to draw the same way they would on paper) can be superior to a industry-standard professional CAD-based interface.

## 5.1 Single-Stroke Gestures

Some of the earliest work in sketch recognition [32] focused on single stroke symbols or "gestures". Symbols are recognized based on simple features such as the angle of the gesture, the locations of endpoints, and the properties of the bounding box of the stroke. These properties were then used as features to train a linear classifier. Long et al. [22] developed an approach that analyzes the similarity between gestures to help developers design ones that will not be easily confused by the computer. Wobbrock et al. [43] present a simple gesture recognizer that could be implemented in 100 lines of code.

While this body of work has led to practical applications like the Graffiti handwriting system used by Palm (shown in Figure 5-1), a major limitation of gesture-based approaches

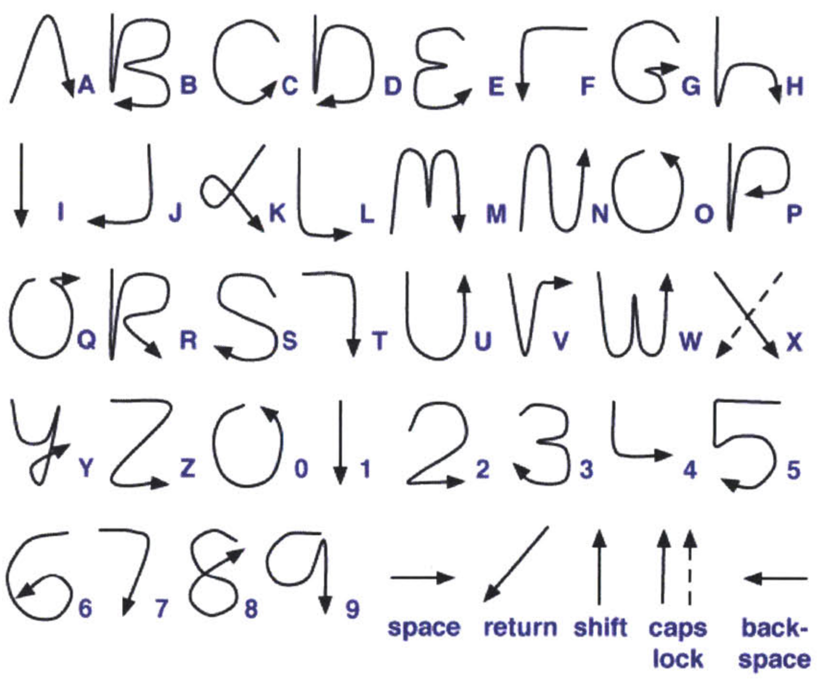
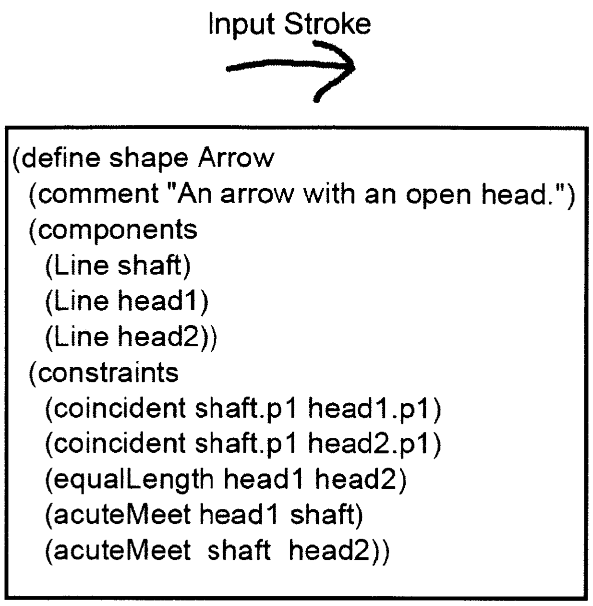


Figure 5-1: Example gestures for the Palm Graffiti handwriting system.

is the restrictions they place on how each symbol can be drawn. Due to this constraint the gestures themselves may not resemble the symbols they are meant to represent, as we can see for letters like "F" and "K" in Figure 5-1. Our system, on the other hand, supports natural input that allows symbols to be drawn with one or more strokes (or partial strokes).

## 5.2 Geometric Primitives and Shape Descriptions

The second group focuses on relationships between geometric primitives (e.g., lines, arcs, etc), specifying them either manually [15, 14, 3] or learning them from labeled data [40, 35] (see Figure 5-2). Here the base vocabulary is typically composed of simple geometric primitives such as lines, arcs, and ellipses. Recognition is then posed as a constraint satisfaction problem, as in [15, 14], or as an inference problem on a graphical model, as in [40, 35, 3]. Shilman et al. [37] used a hand coded visual grammar to describe shapes in the domain, treating recognition as an ambiguous parsing problem. Calhoun et al. [6] model geometric primitives (e.g., lines and arcs) and the relationships between them as graphs, with recognition posed as a graph isomorphism problem.



```
(define shape Arrow
 (comment "An arrow with an open head.")
 (components
  (Line shaft)
  (Line head1)
  (Line head2))
 (constraints
  (coincident shaft.p1 head1.p1)
  (coincident shaft.p1 head2.p1)
  (equalLength head1 head2)
  (acuteMeet head1 shaft)
  (acuteMeet  shaft  head2))
```

Figure 5-2: Shape description for an "arrow" used in Hammond and Davis [15].

Similar to our approach, Szummer [40] proposed using a CRF to classify segments in a diagram by modeling the spatial relationships between neighboring segments. Eric Saund [33] presented a method for labeling edges in a visual scene as either boundaries between regions (e.g., geometric shapes) or thinlines (e.g., lines, twigs). He used a Markov Random Field framework and Loopy Belief Propagation to interpret a wide range of examples include hand-drawn box and connector digrams. Their work differs from ours in that they focused on segments and did not model complete symbols or their contextual relationships.

Alvarado and Davis [3] proposed using dynamically constructed Bayesian networks to parse a sketch, employing both top-down and bottom-up interpretation. Hammond and Davis [15] developed a hierarchical language to describe how diagrams are drawn, displayed, and edited, then used these descriptions to perform automatic symbol recognition. Shilman et. al. [37] proposed an approach that treats sketch recognition as a visual parsing problem.

While these kinds of approaches have been popular and successful in many domains, in real-world sketches it is often difficult to extract geometric primitives reliably. Circles may not always be round, line segments may not be straight, and stroke artifacts like pen-drag, over-tracing, and stray ink may introduce false primitives that lead to poor recognition.

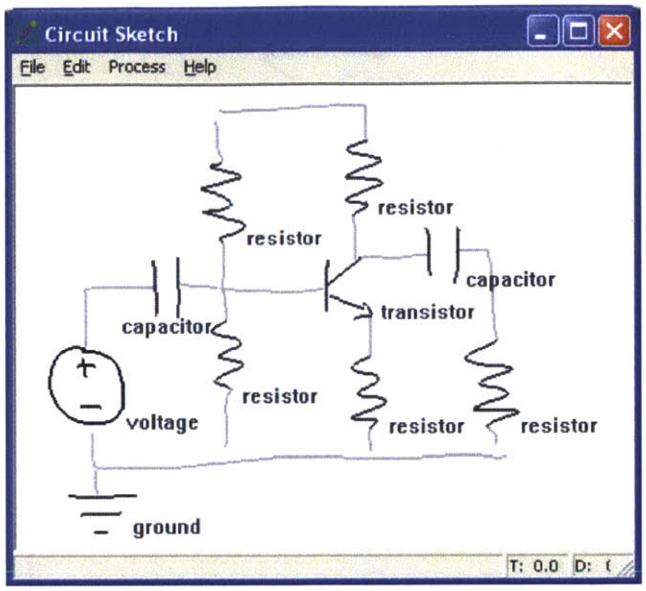


Figure 5-3: Interface for the circuit recognition system presented by Gennari et al. [13].

In addition, recognizers that rely on extracted primitives often discard potentially useful information contained in the appearance of the original strokes. Our work differs from these in that we use a rich model of low-level visual appearance and do not require a pre-defined spatial grammar.

## 5.3 Domain-Specific Models

A third set of related work consists of recognizers that are specially tailored to work in specific domains. Gennari et al. [13] developed a sketch based interface that uses geometry and domain knowledge to interpret hand drawn electronic circuit diagrams (show in Figure 5-3). They employ a set of geometric features (e.g., density, bounding box size, presence of arcs, etc.) and domain constraints (e.g., number of connections) to guide recognition. Kurtoglu and Stahovich [19] presented a similar approach that relies on physical reasoning to resolve ambiguities in sketches of mechanical devices and electronic circuits. In addition to enforcing consistency between pairs of connected components (e.g., a wire cannot connect to a bearing because one is an electrical device while the other is a mechanical one), it uses qualitative simulation to choose between multiple possible interpretations of

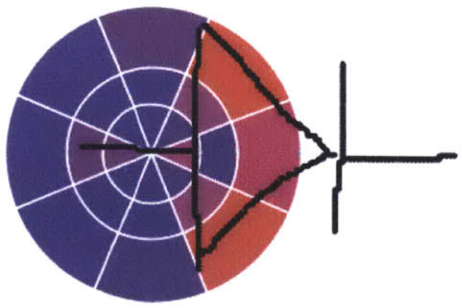


Figure 5-4: Illustration of the visual parts-based descriptor proposed by Oltmans and Davis [27]. The descriptor is centered on a particular interest point, and each bin in the radial histogram encodes the number of ink points contained inside.

the sketch. Their system avoids the parsing problem by requiring users to press a button to indicate that a symbol has been completed. The goal of our work, in contrast, is to develop a trainable multi-domain recognition architecture that can be more easily scaled to new symbols and new domains.

## 5.4 Visual Appearance

The fourth group of sketch recognition approaches focuses on the visual appearance of shapes and symbols. These include parts-based methods [27, 38], which learn a set of discriminative parts or patches for each class, and template-based methods [17, 29], which compare the input symbol to a library of labeled prototypes. Kara and Stahovich [17] developed a trainable symbol recognizer that uses four image-based similarity metrics to perform template matching. Shilman et al. [38] described a method for grouping and recognizing symbols in diagrams and equations, using a Viola-Jones-like detector to search for symbols among spatially connected strokes. Oltmans and Davis [27] proposed a visual parts-based model that uses a library of shape contexts (oriented histograms of gradients, shown in in Figure 5-4) to describe and distinguish between the different symbols in their domain.

The main advantage of vision-based approaches is their robustness to variations in drawing styles, including artifacts such as over-tracing and pen drag. However, these methods typically do not model the spatial relationships between neighboring shapes, relying

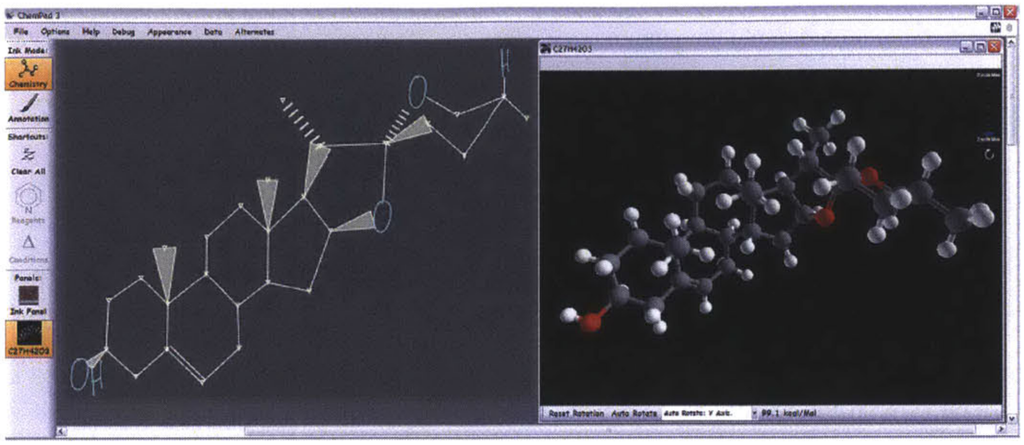


Figure 5-5: ChemPad sketch-based interface for stereochemistry visualization and education [41].

on local appearance to classify a symbol. Our work builds upon these ideas and introduces a new graphical model based architecture for jointly combining information about both appearance and context.

## 5.5 Chemistry

There have also been previous efforts to recognize chemical diagrams. Tenneson and Becker [41] developed a sketch-based system that helps students visualize the three dimensional structure of an organic molecule. Their system was able to avoid many of the challenges in sketched symbol detection by requiring that all symbols be drawn using a single stroke. Casey et al. [7] developed a system for extracting chemical graphics from scanned documents, but their work focused on scanned printed chemical diagrams rather than freehand drawings. Our work differs in that it is designed to support natural free-hand diagrams, allowing users to draw the same styles and notations as they would on paper.

## 5.6 Quantitative Comparisons to Related Work

Sketch recognition is a relatively new field, and we did not find many publicly available benchmarks for the domains we evaluated. In this section, we summarize the performance of existing systems that are similar to ours but were evaluated on other datasets. Alvarado and Davis [3] proposed using dynamically constructed Bayesian networks to represent the contextual relationships between geometric primitives. They achieved an accuracy of 62% on a circuits dataset similar to ours, but needed to manually segment any strokes that contained more than one symbol. Gennari et al [13] developed a system that searches for symbols in high density regions of the sketch and uses domain knowledge to correct low level recognition errors. They reported an accuracy of 77% on a dataset with 6 types of circuit components. Sezgin and Davis [35] proposed using an HMM to model the temporal patterns of geometric primitives, and reported an accuracy of 87% on a dataset containing 4 types of circuit components.

In our evaluation in Chapter 3 we applied our method on two real-world domains, chemical diagrams and electrical circuits (with 10 types of components), and achieve accuracy rates of 98% and 90% respectively. Compared to existing benchmarks in literature, our method achieved higher accuracy even though the other systems supported fewer symbols [13, 35], trained on data from the same user [13, 35], or required manual pre-processing [3].

# Chapter 6

# Conclusion

## 6.1 Contributions

The main contribution of this thesis is a natural sketch recognition system that recreates the familiar experience of drawing with real pen and paper. To preserve the fluid process of sketching on paper, our interface allowed users to draw diagrams using the same notations and conventions. Our system was able to correctly detect 98% of the symbols in a dataset of real-world chemical diagrams and achieved excellent performance on a set of circuit diagrams, improving on the best result previously published in literature. In an on-line study our new interface was over twice as fast as the existing CAD-based method for authoring chemical diagrams, even for novice users who had little or no experience using a tablet. This is one of the first direct comparisons that shows a sketch recognition interface significantly outperforming a popular industry-standard CAD-based tool. We believe that enabling this type of interaction represents one step towards the vision presented in Chapter 1 of being able to interact with a computer just like we would with each other.

Another major contribution was the novel framework for seamlessly combining a rich representation of local visual appearance with a probabilistic model for capturing higher level relationships. This allowed our system to classify each symbol jointly with its context, allowing neighboring interpretations to influence each other. This made our method less sensitive to noise and drawing variations and significantly improved robustness and accuracy.

This thesis also presented a new approach to isolated symbol recognition that preserves the pictoral nature of the ink. This is in contrast to much of the work in the literature, which focuses on geometric primitives and their temporal and spatial relationships. This emphasis on visual appearance made our method less sensitive to stroke level differences like over-tracing and pen-drag. We demonstrated that this method is able to exceed state-of-the-art performance for all the domains we evaluated: digits, PowerPoint shapes, and circuit components.

Finally, we presented a general recognition framework that was successfully applied to two real-world domains, chemistry diagrams and electrical circuit diagrams. This is an encouraging result that suggests that the ideas here may be extensible other domains as well.

## 6.2 Limitations

In this sections we will discuss some of the limitations of our current approach. While the whole template matching model presented in Chapter 2 performed very well for the datasets we evaluated, it can have trouble distinguishing between symbol classes that are visually very similar (e.g., square vs. parallelogram or ac-source vs. current-source). This is due to the fact that the difference between these classes come from a semantic property that is difficult to capture using low-resolution feature images (e.g., parallel, shorter than, etc). This problem can become even more pronounced for domains where symbols are compositional in nature, and thus distinguishable from each other by possibly small differences in their substructures (e.g., the Course of Action diagrams seen in Figure 6-1).

Our approach also makes an assumption about the way symbols are drawn, due to the fact that it only searches for symbols among groups of spatially or temporally neighboring segments. This constraint was not a problem in the two domains we evaluated in our work, but may become more of an issue for a sketch that contains both spatially overlapping symbols and interspersing (i.e., when users start drawing another symbol before finishing the current one). For example, in the flowchart in Figure 6-2 our system could easily mis-group the arrow-shafts and arrow-heads in the center of the sketch if there is interspersing involved (e.g., the user first draw all of the arrow-shafts then came back to add the arrow-heads). In this case it won't be able to rely on the temporal order of the strokes to generate the correct groups.

## 6.3 Future Work

### 6.3.1 Natural Correction and Editing

In free sketching environments we discovered that users often make corrections or edits by simply adding strokes to an existing symbol without erasing the original shape. Some examples of this practice are illustrated in Figure 6-3. Handling these sorts of natural corrections is one interesting area for future work, especially since they are fast and require little additional training to perform.

### 6.3.2 Parts-based Recognition

The idea that objects or symbols can be modeled by their parts has been widely explored in computer vision [12, 11, 23]. Following this intuition, another possible area for future work is to develop a probabilistic parts-based model for recognizing sketched symbols. This approach can supplement the whole symbol visual representation presented in this thesis, and can be especially useful for compositional symbols like Course of Action diagrams shown in Figure 6-1.

### 6.3.3 Learning from Offline Structured Data

One major challenge in sketch recognition research is the relatively small amount of training data that is available for each task. Most collected datasets contain a limited number of sketches (e.g., around a hundred for the ones used in this thesis), compared to the thousands or millions of training examples commonly used in computer vision [11, 42]. Another area for future work is to explore ways to alleviate this shortage, by taking advantage of the large collections of offline structured data available for many domains. For example, the NCBI PubChem structure database [1] contains millions of examples of chemical diagrams

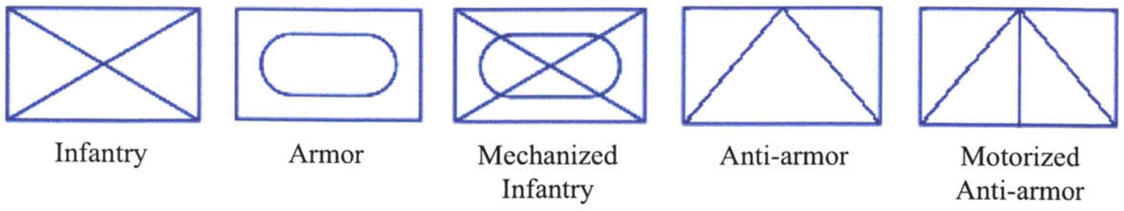


Figure 6-1: Examples of symbols from the military Course of Action domain, highlighting the compositional nature of the symbols in this domain.

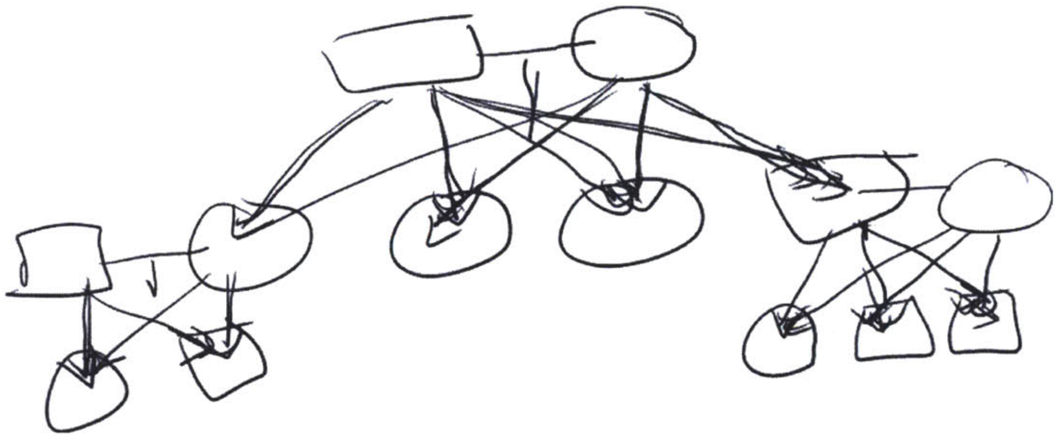


Figure 6-2: An example of a messy flowchart representing a family tree diagram.

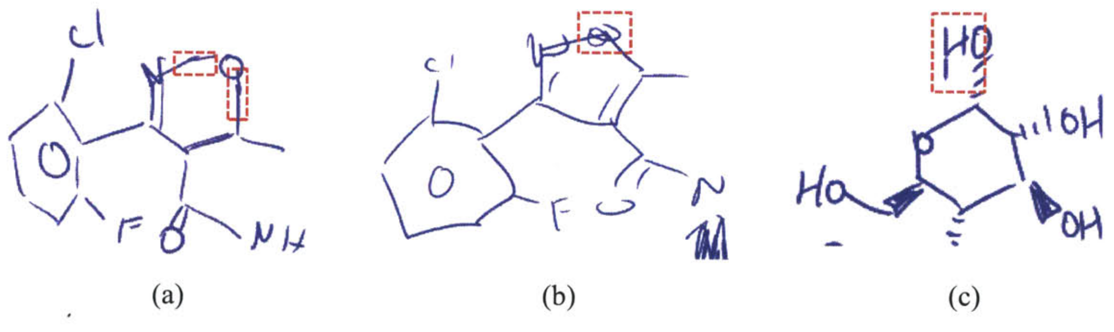


Figure 6-3: Examples of users adding strokes to a previous drawn symbol to clarify it. In (a), the user extended the lengths of two bonds by drawing two additional strokes, in (b) the user drew over and existing "O", and in (c) the user went back to extend the left line of an existing "H".

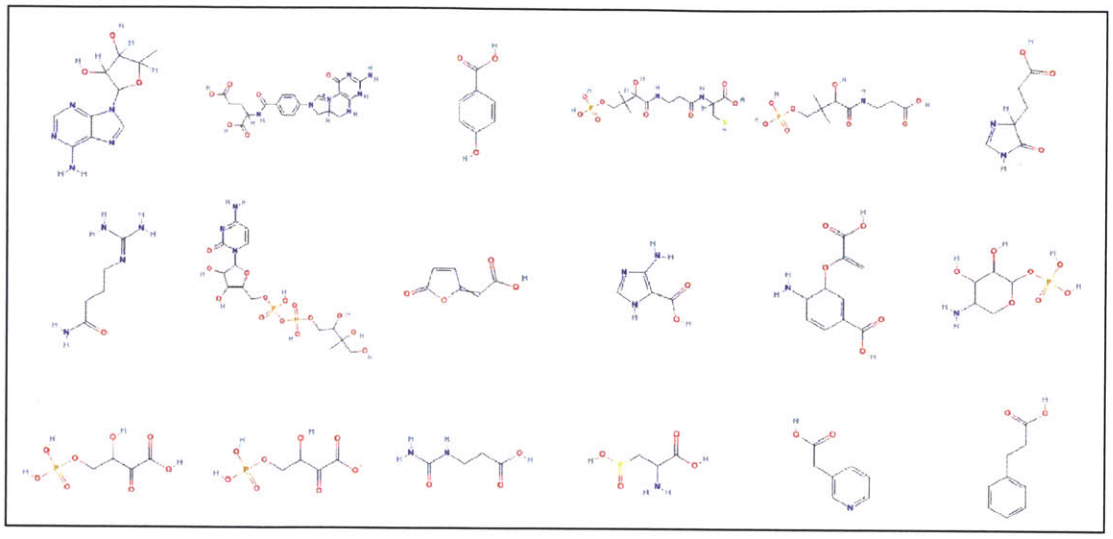


Figure 6-4: A small sample of the off-line chemical structures in the NCBI PubChem database.

in MDL format, shown in Figure 6-4. While these diagrams are not hand-drawn, they do indicate the locations of element and bonds as well as their connectivity structure. This offline data can be used, for example, to train the contextual component of our recognition model (e.g., connectivity patterns like rings, common groups, other substructures, etc.), giving us many orders of magnitude more training examples.

### 6.3.4 Continuous Learning and Adaptation

An important feature of our interface is that it learns how to recognize diagrams from real world sketches, but so far this is done through an one-time offline training process. The next step could be a system that can continually improve itself with usage, adapting to a user's particular drawing style over time. This data can come directly from the user's corrections: each time the user corrects a mistake that corrected symbol could be used as a new training example for the learning algorithm. This potential for automatic adaptability can be especially useful for a new user whose drawing style is very different from those found in the training data.

Furthermore, we can imagine sharing this learning process across multiple users. In this hypothetical scenario a central server (which may be a distributed system residing in the cloud) can collect examples and corrections from the entire user population, periodically updating its recognition model to take into account the ever growing collection of training data. This cloud-based approach could help overcome two of the current challenges in sketch recognition research: the expense of collecting and labeling training data and the computational limitations of recognizing complex diagrams on slower mobile devices.

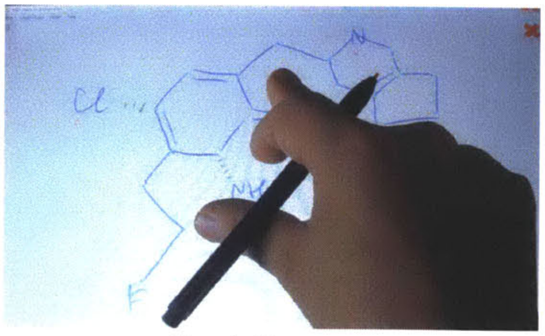
### 6.3.5 Learning New Domains

To take the idea of adaptability even further, we can imagine applying this on-line learning technique to teach our system entirely new symbols or even completely new domains. We only need to ask the user to locate and label the new symbols in each sketch, incrementally teaching our system about their visual appearance and contextual relationships. For example, if a user is trying to teach the system how to understand flow-charts, she can first draw a flow chart and then label all of the symbols and indicate their connectivity to each other. She can also define the semantic meaning of the symbols in the domain so that they can be analyzed in the same sketching interface or exported to a machine-readable format for third-party programs.

From this labeled sketch the system can learn a new recognition model that includes the visual appearance of the symbols, their likely relationships to each other, and perhaps even the rules governing their connectivity. Furthermore, as the model of the new domain becomes more refined, the system can begin to automatically recognize the symbols it has learned so far, relying on the user only for new unknown symbols. Thus, as training progresses, our approach should require less and less supervision.

### 6.3.6 Multimodal Interaction

While there has been a great deal of research on recognizing hand drawn sketches, relatively little work looks at how to combine sketching with other interaction modalities such as speech and touch (illustrated in Figure 6-5). One extremely exciting area for future work is the exploration of these multi-modal interfaces, with the vision of creating a smart notebook or whiteboard that you can draw on, gesture at (using your fingers), and speak to just like



Touch Gestures
(translation, scaling, rotation)

Gestures + Speech
("this [point with finger] is supposed to be a wedge bond")

Figure 6-5: One promising area for future work is to integrate multimodal interaction into the sketch recognition interface, such as combing touch gestures and speech recognition for editing and correction.

you would another person. This opens a number of new natural interactions, including:

- **Correction**. Correct a recognition error by pointing to the corresponding region and saying something like "Oops this is supposed to be a wedge bond".
- **Editing**. Manipulate a sketch using touch gestures similar to those on multi-touch smartphones and tablets (e.g., for dragging, stretching, rotating, etc).
- **Speech Queries**. Since the system understands the semantics of the drawing, a user can ask natural language questions such as "what happens to this circuit if this battery fails."

# Appendix A

# Drawing Notations

## A.1 Chemistry

The vocabulary of chemistry symbols that the system recognizes is outlined in Figure A-1 and Figure A-2. Currently, this consists of common non-metallic elements (e.g., N, H, O), group abbreviations (R, Ac, Me), straight bonds, hash bonds, wedge bonds, charge symbols, and number subscripts.

In order to interpret common organic chemistry diagrams our system also needs to accommodate two common chemistry notations. First, chemists frequently employ what are termed implicit elements, omitting carbon and hydrogen atoms wherever their presence can be inferred by a knowledgeable viewer. A carbon atom is implied whenever two or more bonds connect without a connecting element, or when a bond is drawn without an attachment at one end. For each implicit carbon, enough implicit hydrogen atoms are assumed to be present to fill any vacancies in its valence shell, so that it will have the requisite four connections. In Figure A-2, for example, the four bonds in the lower left box are all attached to an implicit carbon atom in the center. The second special notation involves indicating an aromatic ring by drawing a circle inside a hexagonal configuration of bonds, as in Figure A-2. These rings represent six carbon atoms connected by alternating single and double bonds.

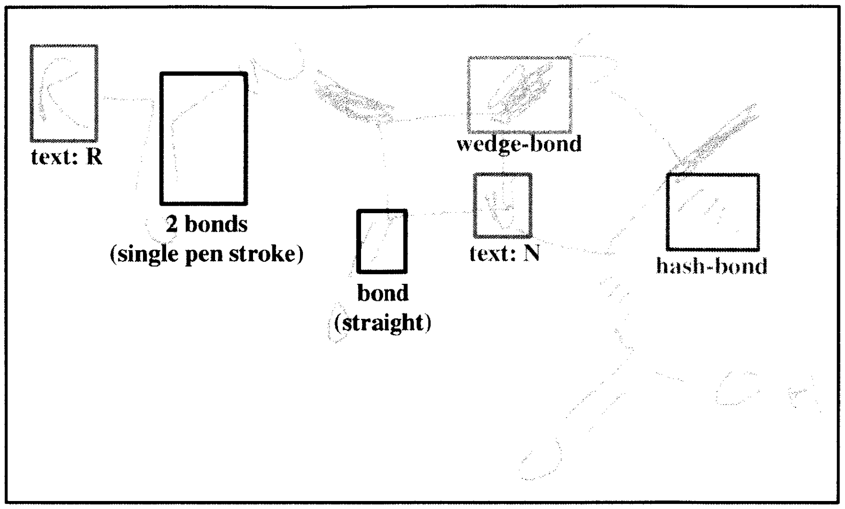


Figure A-1: Grey strokes: an example of a chemical drawing that our system is designed to recognize. The notation consists of element abbreviations (e.g., "N", "O"), group abbreviations (e.g., "R"), straight bonds, hash bonds, and wedge bonds.

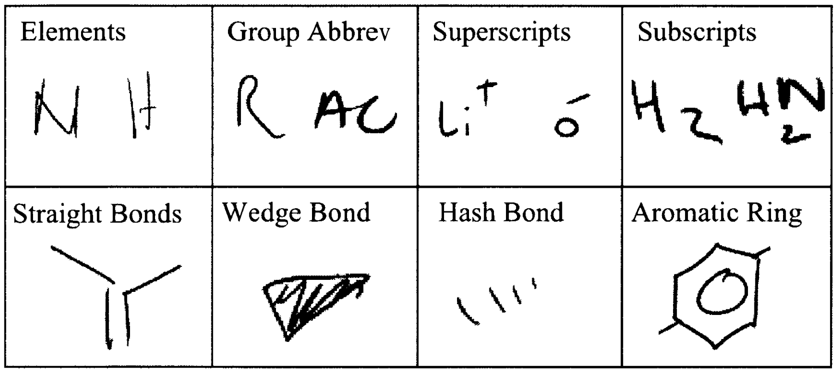


Figure A-2: Notations used in chemical diagrams. Wedge and hash bonds show the 3-D structure of a molecule: hash bonds angle down beneath the plane, wedge bonds angle up.

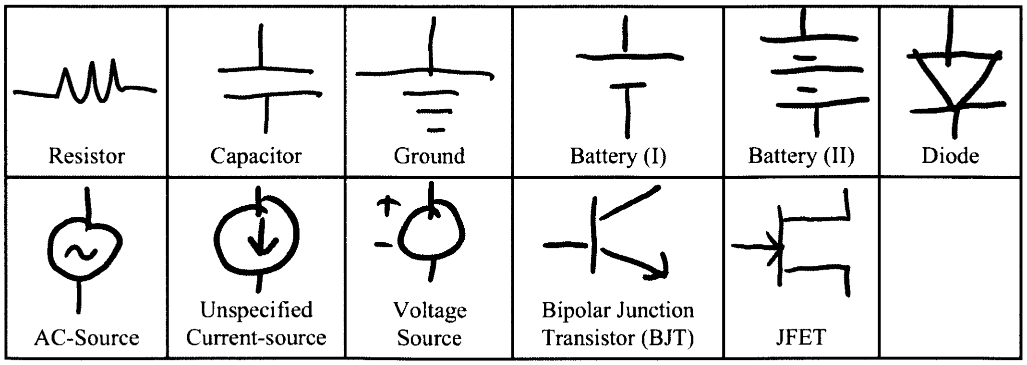


Figure A-3: Notations used in analog electrical circuit diagrams (wires not shown).

## A.2 Analog Circuits

The vocabulary of analog circuit symbols that our system supports is outlined in Figure A-3. These include 10 different circuit components, which are connected by wires in a complete sketch. Note that Battery (I) and Battery (II) are considered the same label during recognition.

# Bibliography

[1] The ncbi pubchem structure database. http://pubchem.ncbi.nlm.nih.gov/.

[2] F. Alimoglu and E. Alpaydin. Combining multiple representations and classifiers for pen-based handwritten digit recognition. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, 1997.

[3] Christine Alvarado and Randall Davis. Sketchread: A multi-domain sketch recognition engine. In *Proceedings of UIST*, 2004.

[4] C. Bahlmann, B. Haasdonk, and H. Burkhardt. Online handwriting recognition with support vector machines - a kernel approach. *Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition, 2002.*, pages 49–54, 2002.

[5] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges. Lerec: A nn/hmm hybrid for on-line handwriting recognition. *Neural Computation*, 7(6):1289–1303, 1995.

[6] C. Calhoun, T.F. Stahovich, T. Kurtoglu, and L.B. Kara. Recognizing multi-stroke symbols. *AAAI Spring Symposium on Sketch Understanding*, pages 15–23, 2002.

[7] R. Casey, S. Boyer, P. Healey, A. Miller, B. Oudot, and K. Zilles. Optical recognition of chemical graphics. *Document Analysis and Recognition*, pages 627–631, 1993.

[8] S.D. Connell and A.K. Jain. Template-based online character recognition. *Pattern Recognition*, 34(1):1–14, 2001.

[9] M.P. Dubuisson and AK Jain. A modified hausdorff distance for object matching. In *Proceedings of the 12th International Conference on Image Processing*, 1994.

[10] M. Everingham, L. Van Gool, CKI Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge 2008 results, 2008.

[11] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[12] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings*, volume 2, 2003.

[13] L. Gennari, L.B. Kara, T.F. Stahovich, and K. Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, 2005.

[14] M.D. Gross. The electronic cocktail napkina computational environment for working with design diagrams. *Design Studies*, 17(1):53–69, 1996.

[15] T. Hammond and R. Davis. Ladder: a language to describe drawing, display, and editing in sketch recognition. In *International Conference on Computer Graphics and Interactive Techniques*, 2006.

[16] H. Hse and AR Newton. Sketched symbol recognition using zernike moments. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 2004.

[17] L.B. Kara and T.F. Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. *AAAI Fall Symposium: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[18] D. Keysers, C. Gollan, and H. Ney. Local context in non-linear deformation models for handwritten character recognition. In *International Conference on Pattern Recognition*, 2004.

[19] T. Kurtoglu and T.F. Stahovich. Interpreting schematic sketches using physical reasoning. In *AAAI Spring Symposium on Sketch Understanding*, pages 78–85, 2002.

[20] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, pages 282–289, 2001.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] A.C. Long, J.A. Landay, L.A. Rowe, and J. Michiels. Visual similarity of pen gestures. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2000.

[23] D.G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, volume 2, pages 1150–1157. Corfu, Greece, 1999.

[24] H. Mitoma, S. Uchida, and H. Sakoe. Online character recognition using eigen-deformations. *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 3–8, 2004.

[25] K. Murphy, Y. Weiss, and M.I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of UAI*, pages 467–475, 1999.

[26] J. Nocedal and S.J. Wright. *Numerical optimization*. Springer, 2000.

[27] Michael Oltmans. *Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, May 2007.

[28] Tom Ouyang and Randall Davis. Learning from neighboring strokes: Combining appearance and context for multi-domain sketch recognition. In *Advances in Neural Information Processing Systems 22*, pages 1401–1409, 2009.

[29] Tom Y. Ouyang and Randall Davis. A visual approach to sketched symbol recognition. In *Proceedings of the 2009 International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[30] Sharon Oviatt and Adrienne Cohen. Toward high-performance communications interfaces for science problem solving. *Journal of Science Education and Technology*, 19:515–531, 2010. 10.1007/s10956-010-9218-7.

[31] Brandon Paulson and Tracy Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *Proc. IUI*, pages 1–10, 2008.

[32] D. Rubine. Specifying gestures by example. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 329–337, 1991.

[33] Eric Saund. Logic and mrf circuitry for labeling occluding and thinline visual contours. In *In NIPS. 2006*, 2005.

[34] T.M. Sezgin and R. Davis. Sketch based interfaces: Early processing for sketch understanding. In *International Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 2006.

[35] TM Sezgin and R. Davis. Sketch recognition in interspersed drawings using time-based graphical models. *Computers & Graphics*, 32(5):500–510, 2008.

[36] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. ACL*, pages 134–141, 2003.

[37] M. Shilman, H. Pasula, S. Russell, and R. Newton. Statistical visual language models for ink parsing. *AAAI Spring Symposium on Sketch Understanding*, 2002.

[38] M. Shilman, P. Viola, and K. Chellapilla. Recognition and grouping of handwritten text in diagrams and equations. In *Frontiers in Handwriting Recognition*, 2004.

[39] Ivan E. Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, DAC '64, pages 6.329–6.346, New York, NY, USA, 1964. ACM.

[40] M. Szummer. Learning diagram parts with hidden random fields. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 1188–1193, 2005.

[41] Dana Tenneson and Sascha Becker. *Interpretation of Molecule Conformations from Drawn Diagrams*. PhD thesis, Brown University, 2008.

[42] A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[43] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *UIST-07*, pages 159–168, New York, NY, USA, 2007. ACM.

[44] Y. Xiong and J.J. LaViola Jr. Revisiting shortstraw: improving corner finding in sketch-based interfaces. In *Proc. Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 101–108, 2009.