UNIVERSITY OF CALIFORNIA, SAN DIEGO

Identification of Distortions, Erroneous Inclusions, and Omittance in Shape Sketches

A Thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science

in

Computer Science

by

Ricardo Shih

Committee in charge:

Professor Lawrence Saul, Chair
Professor Christine Alvarado
Professor David Kriegman

2014

The Thesis of Ricardo Shih is approved and is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2014

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

VITA

2012 Bachelor of Science, University of Maryland, College Park

2014 Master of Science, University of California, San Diego

ABSTRACT OF THE THESIS

Identification of Distortions, Erroneous Inclusions, and Omittance in Shape Sketches

by

Ricardo Shih

Master of Science in Computer Science

University of California, San Diego, 2014

Professor Lawrence Saul, Chair

Sketches are common in fields such as mechanical engineering and electrical engineering for drafting early designs. The ability for a computer to interpret a sketch in a domain of interest is the basis for sketch-based user interfaces. In order to analyze a sketch composition, the sketch first needs to be segmented into groups of strokes which are subsequently classified as a specific shape or component. Incorrectly grouped strokes can lead to errors during classification.

We present a novel approach to identify erroneous inclusions and omittances of strokes that can occur from the segmentation step. Once the strokes are identified as

missing or extra, a refined grouping can be constructed. Our approach utilizes a novel shape representation in order to facilitate identifying erroneous inclusions and omittances while ignoring minor distortions. We represent each shape by a list of vectors called Length Cross Section (LCS) vectors, which measure relative lengths of the cross sections of a shape. Erroneous inclusions and omittances become apparent with the aligning of LCS vectors of a query shape to the LCS vectors of a template. Our approach is rotational, translational, and affine invariant.

We evaluate our method on the domain of digital logic circuits. We find that our method is best at discovering erroneous inclusions and performs moderately well at recognizing omittances. The LCS model can also be adapted to classify shapes and results are presented against the Image Deformation Model by Ouyang and Davis.

Introduction

Sketches are common in fields such as mechanical engineering and electrical engineering for drafting early designs.

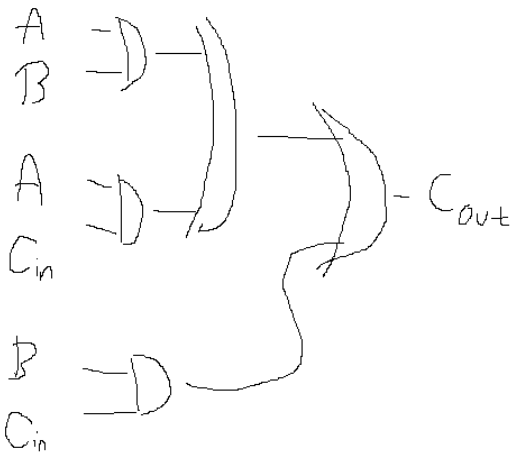

Figure 1. A digital logic circuit sketch diagram.

Sketch-based user interfaces can allow sketches, such as the one in Figure 1, to be tested as a digital prototype. While humans can fairly easily recognize the symbols and components of the sketch, computers have a harder time with these tasks. The core problems the computer must solve to interpret this diagram are:

1. To segment the diagram into groups of strokes where each group represents a single symbol (segmentation)

2. To classify each group as a symbol in the relevant domain (classification)

3. To assemble the symbols into a meaningful diagram (interpretation)

All three steps are inherently intertwined. For example, the segmentation step defines the parts that are believed to belong to a component that is to be classified while at the same time, the characteristics of a shape class can be used to help choose which parts are to belong in a shape group. There is a similar relationship between the interpretation step and the other steps. The interaction between the steps can come about by domain specifics such as differences between labels, symbols, and the interplay between the components. However in this work, we focus on the junction between segmentation and classification by refining an initial grouping before being sent to classification. The interpretation step is not pertinent to this research.

The three separate tasks are often done in a waterfall fashion where the outputs of one algorithm are used as the inputs of the next. The waterfall approach breaks the down the sketch recognition problem into tractable tasks but also allows errors to propagate to each subsequent task. In other words, the success of the classification stage depends on the accuracy of the segmentation algorithm because an incorrectly grouped symbol will likely be misclassified.

The reason most recognition algorithms fail to correctly recognize a shape for incorrectly grouped strokes is that pre-processing steps to align a query shape to a model distort the image incorrectly. The erroneous inclusion or omittance of strokes from the correct grouping can cause the pre-processing to deform a shape into an indistinguishable shape to the classification algorithms.

Figure 2 demonstrates how pre-processing incorrectly grouped strokes can offset the classification algorithm. With the inclusion of the extra piece present in the query, the query is scaled such that the part which should match to the template is smaller than the intended size. Additionally, the centering of the shape is offset. Both of these prove to be fatal to the matching algorithm. Therefore, it is necessary to have the correct strokes

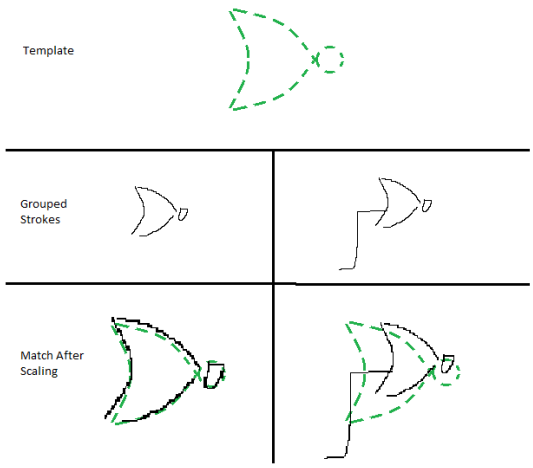

Figure 2. Template matching algorithms are very sensitive to erroneous inclusions and omittance.

grouped together before classification is done.

Unfortunately, the segmentation task is a difficult problem and is prone to error. Trying to find the correct groupings in a naive brute force way would be infeasible, as the number of total groupings is exponential, each of which would need to be run through the classification step. Other techniques such as the use of temporal or spatial information can be used but are not fail-proof. For example it can be seen in Figure 1 that the strokes composing the gates can actually be spatially farther than the wire strokes which are next to or even overlapping the gates.

Our method attempts to correct an initial grouping while avoiding a brute force search by identifying erroneous inclusions and omittances. The identification of missing or additional strokes in an initial grouping must be robust enough to be able to determine the difference between these occurrences versus minor distortions. With extra and missing pieces identified, they can be added and removed respectively and then run through a classifier. By improving the accuracy of the segmentation task, the accuracy of classification should also improve.

In order to accomplish our goal of identifying extra and missing parts of a shape, we need to avoid the problem of misalignment shown in Figure 2. If we can align the correctly grouped strokes in a query to the correct corresponding locations in the template, then the erroneous inclusions and omittances can be more easily discovered. Therefore, our algorithm attempts to align the queries to a shape.

An interesting facet of this problem is that different parts of a sketch can be considered missing or extra depending on the shape being aligned. Figure 3 shows how strokes should be identified given a particular shape to match.

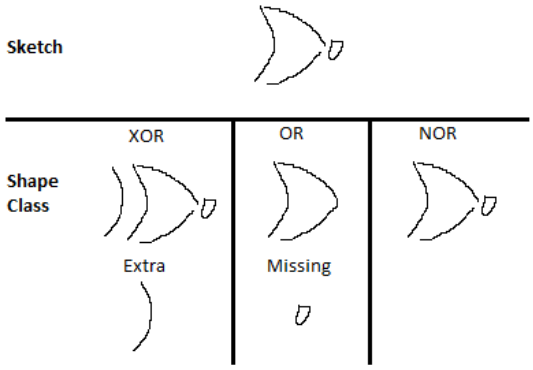

Figure 3. The shape class determines which areas of the sketch are to be considered extra or missing.

Our method identifies what is thought to be erroneous inclusions or omittances given the specific class it is currently matching. Our method computes a match value for each query to class model. Given that the best matching shape may not be the sketcher's intended shape due to the erroneous inclusions and omittances, we present results for the identification of strokes given the best match value and also given the actual shape it is supposed to represent.

The shape domain that we will use is digital logic circuits. The domain allows for a wide range of variation in the sketching styles of the shapes. Figure 4 shows various sketch styles that a NAND gate can take which the shape matching algorithm must be

able to match. Horizontal and vertical elongation and skew are all classes of affine transformations. Therefore, the allowable sketch styles exhibit the need for the shape matching to be affine, translational, and rotational invariant.

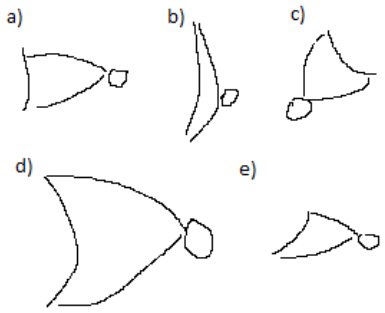

Figure 4. a) A horizontally elongated shape. b) A vertically elongated shape. c) A rotated shape. d) A scaled shape. e) A skewed shape.

Chapter 1

Literature Review

For our problem, we need an algorithm that can align a query shape to a model so that we can identify the locations of extra and missing portions of the shape. In computer vision, the majority of research in shape matching is used for shape classification. Although our method's main purpose is not the classification of shapes, the algorithms described are still relevant to understanding the underlying issues and concerns that must be addressed in our shape alignment task. The three overarching considerations are as follows:

1. We need our shape matching algorithm to be rotational, translational, and affine invariant in order to account for the various sketching styles used in our sketch domain.

2. We need to be able to locate the missing and extra parts of a shape.

3. We need the ability to match shapes in a sketch domain.

It is important to note that each of these considerations are independent of each other. Most of the algorithms discussed in this chapter are not able to guarantee that all three considerations are accounted for.

The two components of a shape matching algorithm that determine whether or not these considerations can be addressed are the way a shape is represented and the

method for comparing a query to a model. In general, the shape representation determines whether we can achieve rotational, translational, and affine invariance and the method for matching a query to a model will determine if we can locate the missing and extra parts of the shape. This pattern is not definite as sometimes the method for matching a query to a model can be dependent on the how the shape is represented.

A shape can be represented in a multitude of ways including point chains, polygon decomposition, curve decomposition, and syntactic analysis [29]. The choice of shape representation can largely impact the types of shapes that can be matched, the method in which matching is done, as well as the success of a classifier. For example, the shape representation used by Anderson, Bailey & Skubic in which angles are measured between consecutively sampled points will be scale invariant due to the fact that angles are invariant under scale changes [1]. This representation, however, would not be ideal for matching shapes that contain many small and disconnected strokes because the algorithm would depend on the order in which strokes are to be evaluated. As strokes are evaluated in the order that they were drawn, the end points of one stroke and the start point of the next may be very far apart. This can mean that two sketches with strokes drawn in different orders can be mismatched even if holistically the sketches are identical. Figure 1.1 shows how the algorithm will parse two sketches which only differ by the order that strokes are drawn.

The method for comparing a query to a model, which we call the matching method, is reliant on the shape representation. A comparison is made after a shape has been transformed into the algorithm's particular shape representation. Anderson, Bailey & Skubic [1] feed the sequence of sampled angles into a hidden Markov model in order to determine the most likely shape class. While this method is conducive to finding a classification, it does not provide the information about areas in the shape which may be an erroneous inclusion or omittance. The following sections will review relevant research

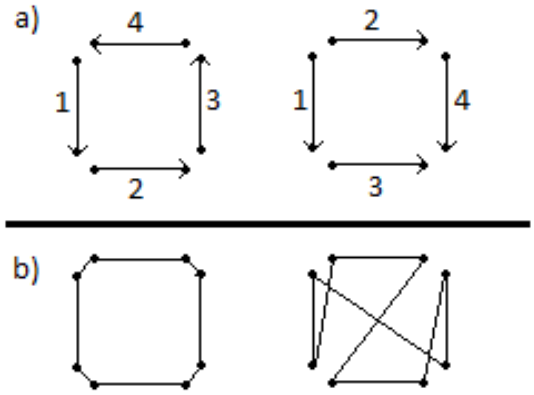

Figure 1.1. An angle classifier attempts to sequence the vertices and uses the ordered angles as a feature vector. It is highly dependent on stroke order. a) The order and direction of strokes are shown for a square shape. b) The order that vertices are processed is displayed. The angles at each vertex are the features used in classification.

in shape matching. Section 1.1 will detail how certain algorithms achieve transformation invariance. Section 1.2 will show how the matching method of various algorithms can or cannot be used to align parts of a shape which is necessary for identifying erroneous inclusions and omittances.

1.1 Transformation Invariance

We need our method to be able to account for shape transformations. The types of transformations which are important for sketched shapes are rotation, translation, scale, and affine transformations. Although our algorithm aims to be affine invariant which is more complex than scale invariance, it is still useful to observe the properties exploited in scale transformations so that we can exploit affine properties in a similar way. Section 1.1.1 discusses a few algorithms which achieve rotation, translation, and scale invariance through various means. Section 1.1.2 discusses algorithms which achieve affine invariance through various means.

1.1.1 Rotation, Translation, and Scale Invariance

The majority of shape classification algorithms are rotation, translation and scale invariant. Lewis [17] describes a fast template matching algorithm using normalized cross correlation. A template which is smoothed using a Gaussian function is scanned over the entire image, and the Euclidean distance of feature vectors is calculated. Since the template is attempting to match at any location, it is translation invariant. Scale and rotation invariance is achieved by repeating the matching using the template scaled to different sizes and oriented at different rotations.

Anderson, Bailey, & Skubic [1] sample the angle between temporally consecutive points in a sketch and feed the chain of observations into a hidden Markov model. The samples are taken in an ordered fashion and are in reference to the previous point. This means that the translation of the shape has no bearing on calculating the metric so the method is translation invariant. Since angles are invariant to scale and rotation, the method also is scale and rotation invariant. Unlike the template matching algorithm, the matching only needs to be done once, which is efficient.

Ankerst, Kastenmller, Kriegel, & Seidl [2] propose a method for shape recognition which partitions a shape into concentric bins. The bins are further divided by sectors to partition the shape into bins as shown in Figure 1.2. A histogram is constructed with each data point corresponding to a distance function that is calculated for each bin.

The method centralizes the shape by measuring the center of mass of the shape. This is how translation invariance is achieved. After centralization, the shapes are normalized so that the standard deviation of the distance of the points from the center is fixed for all examples. This achieves scale invariance. Finally rotation invariance is achieved by rotating the bins about the center and running a comparison.

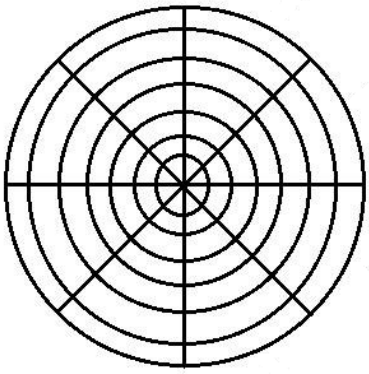

Figure 1.2. Bins used to partition a shape

1.1.2 Affine Invariance

The digital logic circuit domain allows shapes that have undergone some simple affine transformations to be classified as the same shape class. Therefore, it is important for our method to be affine invariant. This section reviews existing methods designed to account for affine transformations.

Hasegawa, & Tabbone [9] utilize the Radon transform in order to create an affine invariant shape matching algorithm. The Radon Transform is a function on an image over scale and rotation. Rotations translate the function while scaling increase the amplitude. By normalizing and translating the Radon Transform, the signals can be matched using dynamic time warping. An affine transformation affects the amplitude of the function at sparse locations, allowing affine transformations to be observed without significantly altering the match values.

Learned-Miller [16] describes how congealing can be used to generalize affine transformations between classes. Congealing is a method of applying affine transformations for each example in a class in order to make all examples as close as possible. Each example may need a different affine transformation so a class can be defined by a distribution of the affine transformations applied to the training set. Using the transformation

probability models for each class, a Bayesian classifier can be built over the space of all transformations. The classifier is constructed using expectation-maximization over unknown bias variables which define the affine transformations. The classification step would be based on the maximum likelihood of a class, given the learned models. The merit in this method was to demonstrate that previous learned distributions can be used to increase efficiency of computation.

Lamdan, Schwartz, & Wolfson [14] developed a technique to reconstruct an affine transformation using interest points. Their method makes use of the fact that an affine homography can be obtained by three matching points of interest between a query shape and template shape. The task would be to discover the best three matching points to reconstruct the affine homography and obtain the best match. An interest point detector is first used and then all permutations of point matches are checked. Mikolajczyk & Schmid [19] describe methods in which affine invariant interest points can be discovered.

1.2 Global Versus Structural Matching Methods

The matching method is the method in which a query shape is compared to a model. Shape matching algorithms can be divided into two method types: global methods and structural methods. The difference between the methods is whether the shape is represented as a whole, or separated by segments or sections [29]. Global methods produce a singular match value for each class to determine a best match. Structural methods may also do the same for a final classification, but additional match values are calculated at localized portions of a shape which makes shape alignment easier. In order align a query to a model to pinpoint the location of missing or extra pieces of a sketch, our method uses a structural matching method. There do exist a few global method algorithms which aim to create a partial shape match using a global shape similarity measure which will be described in section 1.2.1.

1.2.1 Global Methods

Global matching methods employ the entirety of a shape for the matching task. These methods sometimes utilize a single shape similarity metric in order to determine a best match. Otherwise many times a multi-dimensional feature vector is constructed and a distance metric is used to compare similarity between shapes.

An example of a global method that uses a single shape similarity metric is the Hausdorff distance used by Kara & Stahovich [13]. The Hausdorff distance is calculated between two sets of points A,B by

$$H(A,B) = max(h(A,B),h(B,A)) \tag{1.1}$$

where

$$h(A,B) = \max_{a\in A} \min_{b\in B} \|a-b\| \tag{1.2}$$

The Hausdorff distance is a singular number that is computed for each match. Kara & Stahovich use this metric with some modifications for shape classification.

Global methods many times compare multi-dimensional feature vectors. You & Jain [27] describe a shape by calculating the n^2 chord lengths/distances from each point to each point and constructing a histogram of the lengths. Using this histogram, the matching task is easily done by calculating a distance metric. You & Jain opt to use the Euclidean distance.

Osada, Funkhouser, Chazelle, & Dobkin [21] propose a method of creating a feature vector containing multiple single global shape descriptors. Their method finds the averages of these descriptors over random points sampled throughout a shape. The shape descriptors they use are properties such as the distance between two random points along the contour, the angle between three random points, the distance between a fixed point

and a random single point on the contour, the square root area of a triangle constructed out of three random points, and the root cube of the volume of random points in a 3-D shape.

One of the most successful shape matching algorithms uses Zernike moments. This method was developed by Hse & Newton [11]. The feature vector is made up of the set of Zernike moments calculated for a shape. They use multiple metrics for shape comparisons including support vector machines, minimum mean distance, and nearest neighbor.

Our method intends to be able to identify erroneous inclusions and omittances. For this we look at shape matching algorithms that can match partial shapes. The majority of algorithms which can match a partial shape use structural matching methods. The rest of this section describes two partial shape matching algorithms which utilize a global matching method.

The algorithm proposed by Latecki, Lakaemper, & Wolter [15] is able to find a partial shape match using a global shape similarity metric. It requires that the query and template be a continuous polyline. The query polyline is matched consecutively at different starting points along the template polyline, meanwhile calculating a single shape similarity metric. The template polyline is modified in their algorithm to attempt to find the best match. Unfortunately, this method would be difficult to translate to the sketch domain as shapes are not guaranteed to be continuous polylines.

The other algorithm is designed by Hirata & Kato [10]. They compare a partial sketch to a scene in a database. The database scene uses edge detection to create an image consisting of lines to match to the sketch. Unfortunately, this method is not translational invariant, as it requires the sketcher to approximately sketch the scene in place.

1.2.2 Structural Methods

The structural method for shape matching partitions a shape into segments or sections which can be matched independently. The types of structural methods are very diverse as some can be region matching, point matching, or stroke matching.

A region matching algorithm is presented by Ouyang and Davis [20]. After shape normalization, the shape is partitioned into a 12 by 12 grid. For each of these grid regions, separate features are measured and compared. The features measure how nearly horizontal, vertical, or diagonal the stroke is at each point in the region. The 3 by 3 grid centered at each region is allowed to move in any direction by one region in order to find the best overall match value for the center region. A very similar method is used by Bau [3] in which the features are measured by Gabor filters instead.

Liang, Li, Baciu, Chan, & Zheng [18] developed a partial shape matching algorithm which matches bi-segments (2-connected segments) in a query to bi-segments in a model. The algorithm is in three parts. The first is to smooth the lines, the second is to separate the sketch into segments (strokes). The third uses a bi-segment model in order for local features to be measured and compared. The method is aimed at matching queries with missing pieces, but does not account for extra strokes. Additionally the method requires that a shape be represented as a contour.

Donoser, Riemenschneider, & Bischof [6] created a shape matching algorithm which is unique in that it matches the largest partial portion of a shape. The method requires the shape to have a closed contour and is implemented in a domain consisting of silhouettes with occlusions. For each point along the contour, an angle is calculated with that point, a point δ points away, and every other point along the contour, creating an n by n matrix. By matching the largest portion of this matrix to another example, the largest partial shape match can be found which is resistant to missing and extra pieces.

Unfortunately, as mentioned in section 1.3.2, a closed contour cannot be guaranteed in the sketch domain.

Algorithms mentioned earlier, like Belongie, Malik, & Puzicha's shape contexts from section 1.1.1 and Lamdan, Schwartz, & Wolfson's method from section 1.1.2 are structural matching methods. The shape contexts match each point to one other point. Lamdam, Schwartz, & Wolfson match each interest point to one other interest point.

1.3 Issues Regarding Matching Sketches

While the previous algorithms reviewed in this chapter are all operable in the sketch domain, for completeness we present shape matching algorithms which are not fit to be used in the sketch domain. The problem of matching a shape is a vague problem. In particular, the algorithms for matching a shape depend on how a shape is to be defined. Some shape matching algorithms are able to achieve fast results for matching by making certain assumptions about a shape which cannot be made in the sketch domain. For example, we cannot assume that shapes will have rigid structures or that a shape will be closed.

1.3.1 Resistance to Distortions

The sketch domain assumes the existence of some amount of distortion in the shapes. There are shape matching algorithms which are tailored for matching exact shapes that are not applicable in the sketch domain. Stevens [25] uses a method called contour projection to discern the borders of a shape in which signals are propagated inwards towards a shape until borders are found. In this way, the contours of a shape can be discovered. It then uses localized patterns of signal propagation in order to create a geometric shape pattern that is invariant to rotation, scale, and location. However this method is used to identify rigid shapes that are exactly scalable from a known shape

pattern. The sketch domain will not have a test set example which is exactly scalable to another separate sketch. So while in the sketch domain, only methods that are resistant to distortions can be considered.

1.3.2 Shape Properties

There are specific classes of shapes that are defined by particular properties. A shape can be open or closed, regular or irregular, concave or convex, and may even consist of multiple sub shapes. Zeljkovic, Vincelette, & Savic [28] present shape recognition algorithms utilizing polygon boundaries that is tolerant to severe noise. Their method calculates the area of the shape, rotates examples about the center of mass and calculates the highest area overlap. While their algorithms are efficient, we cannot make the same assumptions that shapes will consist of closed polygonal lines in the sketch domain. Sketched shapes may not have a clearly identified boundaries or borders and can be open and even disconnected. While there may be means in which to extrapolate a boundary for a sketch, it is likely that important features can be lost in the process.

1.4 Discussion

The major properties that we are looking for in a shape matching algorithm are affine invariance, and the ability to align shapes in the presence of missing or extra pieces. Other desirable properties are that the methods should be resistant to distortions and should be able to match the very generic types of shapes that arise out of the sketch domain.

Based on the algorithms reviewed in section 1.1, we see the need for the shape representation to be able to exploit properties of affine transformations. The shape representation that we use in our algorithm exploits the particular property of affine transformations that parallel lines stay parallel and the ratio of lengths of parallel lines

remains invariant after transformation.

Nearly all methods that consider shapes with missing portions or occlusions are structural matching methods. The global matching methods reviewed in section 1.2.1 which consider partial shape matches are not applicable to this domain. Latecki, Lakaemper, & Wolter's method requires unbroken polylines which is not guaranteed in a sketch domain and Hirata & Kato's method is not translation invariant.

The matching method can also determine whether a shape match with missing or extra pieces is possible. A very common pre-processing technique is to center and normalize a shape before matching can be done. This pre-processing is present in all of the algorithms by Kara & Stahovich; Bau; Hasegawa & Tabbone; Learned-Miller and more. Figure 2 demonstrates the problems that arise for methods that use this technique. The missing or extra strokes would interfere with the process which would result in an incorrect centralization and normalization. As a result, the classification task's accuracy drops for these examples.

Of the methods reviewed in this chapter, only one method is both affine invariant and utilizes a structural method. Lamdan & Wolfson's method described in section 1.1.2 matches interest points and reconstructs an affine transformation from the points. The difficulty in applying this method to our problem lies with the task in finding interest points. Lamdan & Wolfson tested their algorithm on the domain of silhouettes in which it is easier to find interest points than in the sketch domain. Additionally an 'And' gate would have exactly three interest points. If one stroke was missing, then there would be insufficient interest points to construct an affine transformation.

Additional things to think about are that some methods utilize temporal properties or drawing styles to gain valuable information about the sketches. Stroke order, number, direction, and drawing speed can be used as features for matching. It is debatable that these features should be used, because if any particular user deviates from the drawing

styles of the majority, their sketches may be misclassified with very high likelihood even if a human would have no difficulties identifying the shape class.

Our proposed algorithm is both affine invariant and utilizes a structural matching method in order to align the query to a model. Our model is operable in the sketch domain, and does not utilize temporal data which can be highly variable between different sketchers.

Chapter 2

Approach

2.1 Overview

The goal of our approach is to identify extra and missing pieces given an initial grouping of strokes. The approach will be applied in the digital logic circuit domain where the possible shape categories are the AND, OR, NAND, NOR, NOT, and XOR gates shown in Figure 2.1.

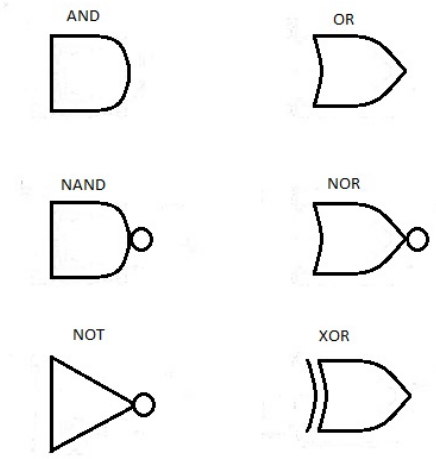

Figure 2.1. Digital logic circuit gate categories

Given a candidate set of strokes that may contain missing or extra strokes, our algorithm transforms the candidate strokes to best align with a model, without allowing

missing strokes or extra strokes to incorrectly skew the alignment between the candidates and the model. The erroneous inclusions and omittances are to become apparent in relation to the alignment.

Our approach uses a unique shape representation which achieves rotational, translational, and affine invariance in order to facilitate the alignment of a grouping of strokes to a shape model. Section 2.2 will explain how and why we chose to represent shapes for our approach. Our approach then generates and stores a single model for each shape classification in our domain (Figure 2.1) that will be used for alignment. The model will be representative of the training set. Section 2.2.1 describes the process of generating these models. The procedure for aligning each example with a model is described in section 2.3. Sections 2.4 and 2.5 go into the finer details of how a best alignment is determined. Once an alignment is chosen for a model, the extra and missing strokes can then be identified. Section 2.6 explains the stroke identification process.

2.2 Shape Representation

The most basic shape representation is a set of strokes which contain (x,y) pixel coordinates. We must transform this shape representation into a different shape representation that allows the identification of missing and extra pieces and also guarantees affine invariance. Our representation exploits the properties of affine transformations that parallel lines stay parallel and that the ratio of lengths of parallel lines stay constant.

Given the (x,y) pixel representation of a shape, we construct a smaller 2-D matrix to represent the shape. The new matrix will be constructed such that at equidistant points along an arbitrary reference axis, there is an associated 1-dimensional vector. The 1-dimensional vector represents the ordered length of segments, or cross sections, that are perpendicular to the chosen reference axis. The 1-dimensional vector will be referred to as a length cross section (LCS) vector.

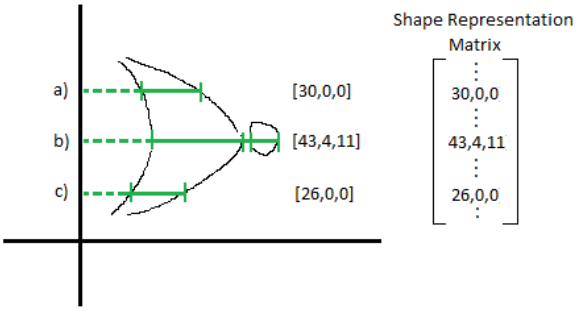

Figure 2.2. Length Cross Section vectors at various points along the y-axis

Figure 2.2 shows the construction of the LCS vectors on a NAND gate at arbitrary points. Choosing the y-axis as the reference axis, we define a LCS vector at consecutive intervals along the y-axis. Each segment in a LCS vector is the length between consecutive pixels that lie on the axis perpendicular to the reference axis. A point in the sketch serves as a separator of sections. Note that the length of the LCS vectors will always be equal to the longest LCS vector, and that any non-numerical value will be set to 0. This allows us to construct a 2-dimensional matrix out of all of the LCS vectors.

2.2.1 Model

Each shape class needs a model to compare against. The model for each class will be a single 2-D LCS vector representation. The model is constructed for each class by taking a training set of sketched gates and creating one shape descriptor. For each example in the training set, it is first converted into an LCS vector representation. The examples are rotated to be oriented in the same way. The model is then created by combining all of the training example LCS vector representations into a single representation through averaging the LCS vector matrices.

Since it is common for an LCS vector to contain very small length sections due to sketch distortions, taking the mean of the LCS vectors results in smaller than intended

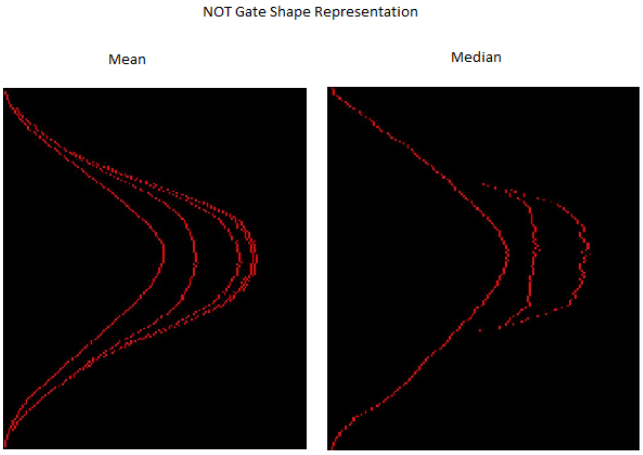

Figure 2.3. Pixel representation of the NOT gate models using mean versus median

length sections for the model LCS vectors and also creates many sections at the ends of the model LCS vectors which have near zero values. In order to ignore these distortions, the median value is used instead. Figure 2.3 transforms the shape representation back into pixel coordinates to illustrate why using the median creates a better model.

2.2.2 Pre-processing

This section describes the details of transforming (x,y) pixel oriented examples into the LCS vector format. The pre-processing techniques used also help to clean up the inputs in order to minimize errors and speed up the algorithms. The examples used for training and the examples used for queries are pre-processed in the same way with two differences.

The first difference is that the query examples will only be processed for a single orientation whereas the training examples for the model are processed for each rotation. To achieve rotational invariance, either the query or the model needs to be rotated to find the best matching rotation. We choose to process the model for each rotation since the model has more examples to use and distortions would be less likely to be manifested in

the model LCS vectors. The second difference is that the training examples record LCS vectors for each pixel along the reference axis (y axis), whereas the query examples will only record N LCS vectors spaced at equidistant intervals.

To ensure consistent behavior, the examples are centered at the average (x,y) coordinates at each rotation. The x and y coordinates are then normalized by the standard deviation of the distance of points from the origin. A standard deviation of 50 was chosen so that the size of the example is about 200 pixels. This means that the trained model would have approximately 200 LCS vectors.

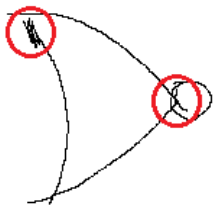

Figure 2.4. Overlapping strokes and sketch corrections create many small distortions.

A common occurrence is that LCS vectors have many very small sections due to overlapping strokes or the person drawing the sketch attempting to correct or highlight a section of the sketch. Figure 2.4 highlights locations with many small distortions. The existence of many small sections can greatly affect the computation speed of the vector matching problem described in section 2.4. Therefore, if a section is less than 2% of the length of the sketch, we combine it together with the previous section. In our algorithm, we normally use the combining of sections to determine erroneous inclusions and omittances as described in the stroke classification section 2.6. We combine these smaller sections in the pre-processing step under the assumption that these are considered minor distortions so should be ignored in stroke classification.

2.3 Matching Method

2.3.1 Affine Invariance and Alignment

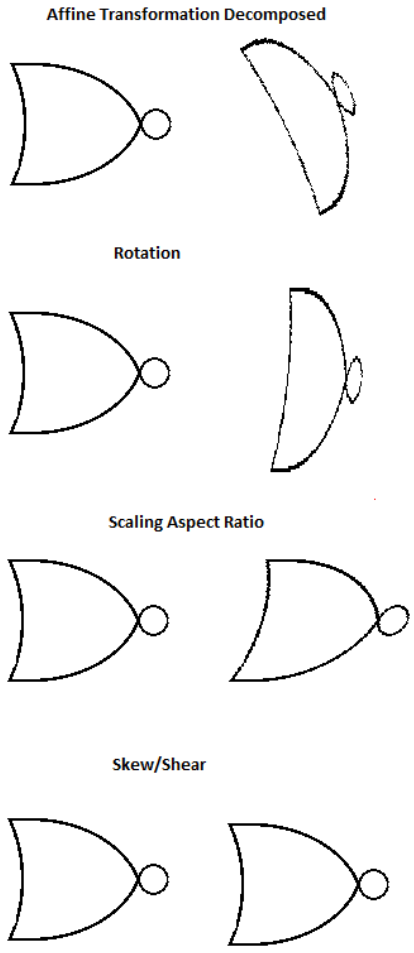

Figure 2.5. Individual transformations of an affine transformed NOR gate.

This section will give a general illustration of what steps are needed to align a potentially affine transformed query to a model. The details of the alignment algorithm will be explained in section 2.3.2. Figure 2.5 shows a NOR gate that has undergone an affine transformation. The transformation can be decomposed into a rotation, followed by resizing the aspect ratio, followed by a skew/shear. The order of transformations is

not strict, but a different order will have different decomposed transformations.

With the knowledge that a shape has undergone an affine transformation, the alignment task is broken down into discovering the transformations. The first step is to discover the reference axis which results from the rotational transformation. The second step is to discover the aspect ratio about the new reference axis. The last step is to discover the skew.

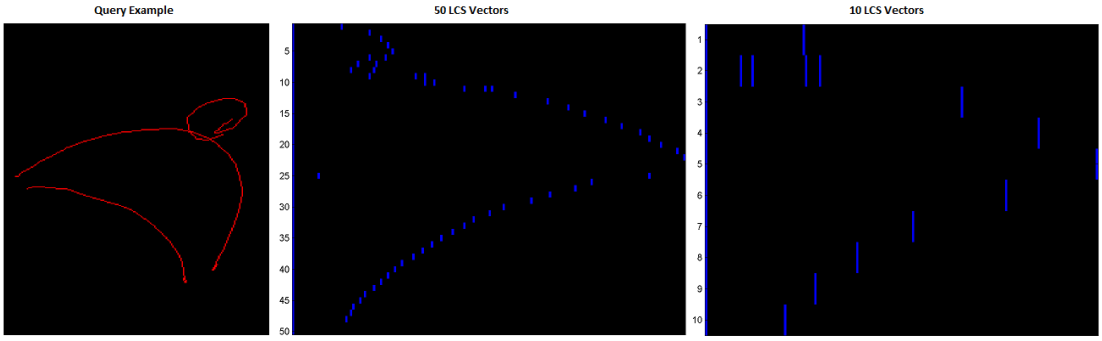

Figure 2.6. LCS vector representation of a NOR gate example.

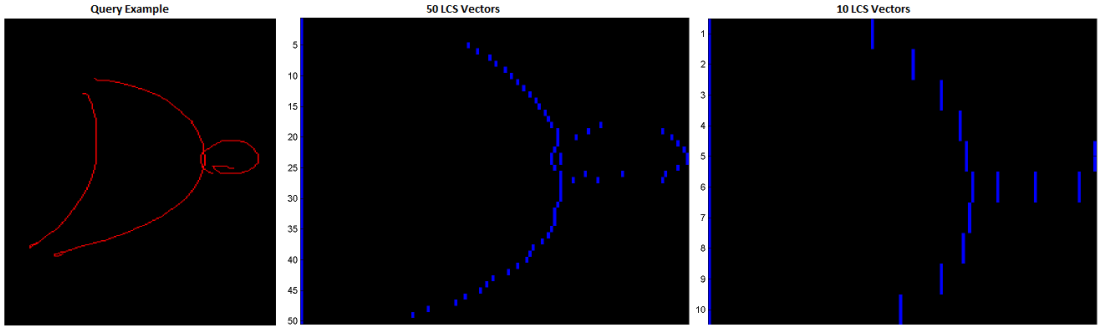

Figure 2.7. LCS vector representation of a rotated NOR gate example.

To show how the LCS vector representation is useful, we examine how it behaves on a shape that has undergone an affine transformation. As described in section 2.2.2, a query example will consist of N LCS vectors. Figures 2.6 and 2.7 shows a visualization of the LCS vector representation with 50 and 10 LCS vectors taken at equidistant points

along the reference axis (y axis).

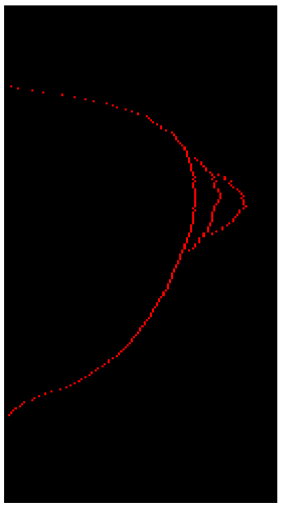

Figure 2.8. NOR gate LCS model.

Figure 2.8 shows the model that we wish to compare our NOR query against. The first step is to discover the correct rotational transformation. Figure 2.7 shows the correct orientation that we want to use to continue the alignment process.

After the rotational transformation is found, the next step is to find the correct aspect ratio. Figure 2.9 shows an incorrect and correct alignment of a NOR gate example to a model. The aspect ratio is manifested by the scaling of LCS vectors and the distance between the LCS vectors. These are the two dimensions of the aspect ratio. The LCS vector representation scales all of the LCS vectors by the same amount. This guarantees that the LCS vectors, which are parallel, preserve the ratio of lengths and guarantee affine invariance. The LCS vectors are also always equidistant. The logic is the same, as the ratio of lengths between the LCS vectors should stay 1:1 to guarantee affine invariance.

These figures illustrate several points. The first point is that that the alignment process needs to be able to find the correct starting placement of the LCS vectors. The incorrect alignment in Figure 2.9 starts the alignment outside of the boundaries of the

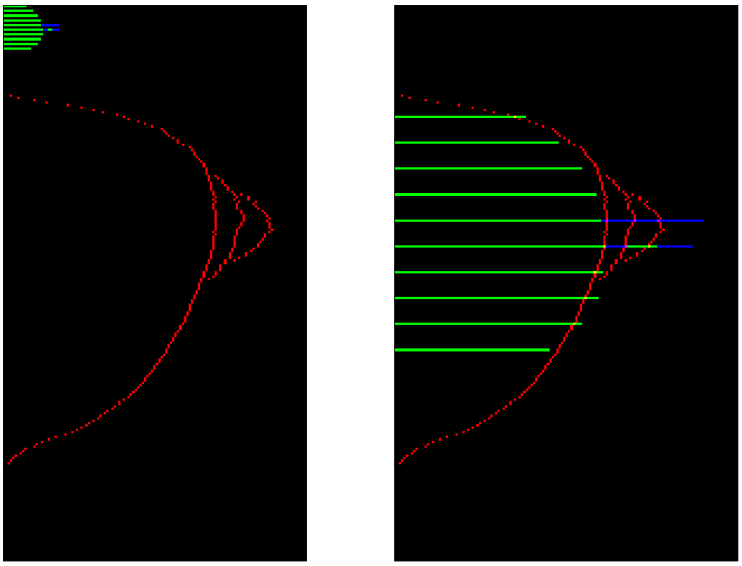

Figure 2.9. Incorrect and correct alignments of a NOR gate example to a model.

model whereas the correct alignment starts the alignment just inside the boundary of the model. The second point is a minor point in that smaller N values chosen for the number of LCS vectors to represent a shape determines how small we can scale the query along the reference axis.

The last task in order to recover a full affine transformation is to find the skew. Skew translates points on the axis perpendicular to the reference axis. Since we only consider the lengths of sections and not their relative starting points in the Cartesian coordinate system, the LCS model represents a shape such that the skew is ignored. No matter how an example is skewed at this point in the alignment process, the LCS representation will be the same.

2.3.2 Algorithm Details

Given a model and a query, the matching algorithm will align the query to the model. The variables that must be discovered are the new reference axis, the aspect ratio, and the starting placement of the LCS vectors. A match value can be computed for many

different alignments to find the best alignment. We use a brute force search for most of the parameters. We calculate match values for each new reference axis for a discrete set of rotations about the center of the query. The higher number of discrete rotations R, the higher the chance that the alignment is as close as possible to the model.

The components of the aspect ratio are the distance between the equidistant length between sample LCS vectors, and the ratio of the model LCS vectors and the query LCS vectors. We can brute force search both the equidistant length between LCS vectors and the starting placements.

Matchings of LCS vectors between the model and query are defined by starting point and the equidistant length between the query LCS vectors. We test a discrete number of possible equidistant lengths Y. Y is chosen such that the smallest distance between LCS vectors is 1 pixel, and the largest allows the entire query to span 1.5 times the size of the model. Therefore, if H is the height of the model in pixels and N is the number of sampled LCS vectors in the query, the set of values used are Y=$[1, 1.5 * H/N]$. Since there are N samples, the match will span $N * Y/H * 100\%$ of the entire model. Therefore, the choice of values of Y mean that we will be looking at matches where the query can be scaled vertically from $N/H * 100\%$ to 150% of the model. We allow the starting and ending LCS vectors of the query to overflow past the original model in order to account for the query containing extra strokes above and below the model.

At this point in the algorithm, an arbitrary rotation has been chosen from the discrete set of rotations and an arbitrary length has been chosen from the discrete set of values Y. This is still not enough to align the model and query. There needs to be a starting point to match the first LCS vector of the query. This starting point can exist outside the bounds of the model as shown in Figure 2.9, meaning LCS vectors in the query could be matched with nothing. The starting points are chosen so that at least $2/3$ of the query are matched within the bounds of the model. If this number is smaller, then

the algorithm simply creates more matches to be calculated which have a low probability of being the best alignment since non-matched LCS vectors will not contribute to the match value. Once a starting point, rotation, and $y \in Y$ length are defined, the query and model have an alignment. The first LCS vector of the query is matched at the start point on the model, and each subsequent piece is matched y pixels apart. The last step is to find the ratio of the model LCS vectors and the query LCS vectors and compute a match value. The method for both of these steps are described in sections 2.4 and 2.5

2.4 Vector matching

In order to determine a good match in the matching algorithm, the LCS vectors must be compared. Given two 1-dimensional vectors of equal length, a distance or error metric can be computed. In our matching problem, an error metric must be calculated for the matching of a query LCS vector L to a model LCS vector L'. The lengths of L and L' may be different as the query can contain more or less sections than the model. We must operate on the LCS vectors to make L and L' the same length to run an error metric. If an LCS vector is 0, it signifies that there was no section and therefore should be ignored in the matching.

Intuitively, matching the LCS vectors is determining how close the cross sections from the query match the cross sections of the model. First the query vector is scaled by an α so that the total sum of the model and query vectors is the same. This can be done since we assume the ratio of lengths stays constant so multiplying by a constant factor does not change the ratio between cross sections. Ensuring that the model and query sum to the same value makes the model and query LCS vectors have a 1:1 ratio, where α should correspond to the aspect ratio about the axis perpendicular to the reference axis.

Our method allows sections in both the query and the model LCS vectors to be combined and/or have the end sections pruned. Note that pruning sections will generate

different α values and the total sum of the vectors will change. Combining and pruning sections achieves two results. First, it will allow the L and L' vectors to be modified to be the same length so that an error metric can be computed. Secondly, the combining and pruning of sections in each LCS vector is the key to finding extra and missing pieces. If a section in the query L is combined and matched to a section in L', this signifies that the joining point of the sections was present in the query, but should be removed to create a better matching. If the query has a pruned section, it signifies that the end points are to be removed as well. These signify erroneous inclusions in the query. Likewise, if a section in the model L' is combined and matched to a section in L, it signifies that the query is missing a point where the template sections are joined. Pruned sections in the model designate missing points which signify omittance in the query.

If we denote the resulting combined query vector as CL and the combined model vector as CL', the total error is computed by the average error per section

$$\sum_i \frac{|CL_i - CL'_i|}{CL'_i} / |L| \tag{2.1}$$

In order to avoid simply combining all sections in both L and L', we constrain the problem by only allowing combined sections of L to be matched with an uncombined section of L' and vice versa. One way to imagine the vector matching problem is a path traversal of a NxM grid, where N is the length of L and M is the length of L'. A horizontal movement through the path signifies combining a model section, a vertical movement signifies combining a query section, and a forward diagonal movement signifies that the algorithm starts looking at the next sections in the model and query. The constraint mentioned earlier means that the path cannot move vertically if the previous movement was horizontal and vice versa. Figure 2.10 shows a valid path through a grid representing the combination of LCS vector sections. Figure 2.11 demonstrates the

constraint violations of the path through a grid.

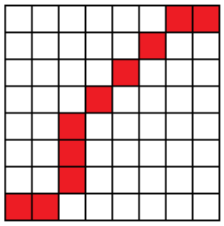

Figure 2.10. The path through a grid represents the combining of query and model LCS vectors

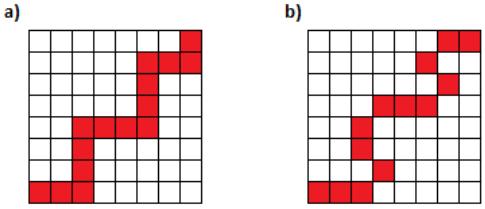

Figure 2.11. Invalid paths through a grid for combining query and model LCS vectors. a) has vertical movements followed by horizontal movements and vice versa. b) has backward diagonal movements

To brute force search every path is $\binom{M+N}{N}$ which is $O((M+N)^N)$. Since this can get large very fast, we limit $N, M \leq 5$ so that it is O(252). This is only possible because the domain has vector lengths generally around 1-4. If there are more, it is usually caused by the small distortions, which are automatically combined into their largest neighbor in pre-processing to improve computation speed. This is reasonable to do given that distortions should be ignored in matching. We can also use Dynamic Time Warping (DTW) to search a pruned space which is $O(MN)$.

DTW requires a cost function d to be defined for every action. This function

needs to have a positive cost in order for the algorithm to work. In the experiments run, we design a cost function such that the cost function is 0 if a section is combined (a horizontal or vertical movement through the grid) and is smaller than its matched section in the other LCS vector. Otherwise it will return the cost of matching the two sections $\frac{|CL_i - CL'_i|}{CL_i}$. If a diagonal movement through the grid is taken when the combined sections are smaller than the larger section, then no cost will have been recorded. In this case, we make the cost function for this action to be the cost of matching the sections.

2.5 Voting on a Scale Factor

Voting is the mechanism for finding the ratio between the LCS vectors of the model and the query. We want to minimize $\sum_n error(\alpha' LCSvector_n, LCSvector'_n)$ for some α' value. In the vector matching described in section 2.4, we assume that $\alpha_n LCSvector_n = LCSvector'_n$ for arbitrary query and model LCS vector matches. However, the α_n scale factor is only relevant in the comparison for a single LCS vector pair. We want to find an α' value that is best for all of the N LCS vector pairings.

The α' scale is a scale correction of a query after pre-processing to match to a model. The need for scale correction comes from the existence of missing and additional strokes affecting the pre-processing normalization. Since α' is a real number it is necessary to have a discrete set of α' scalings that we look for. We choose to set the boundaries of α' to [0.01,100]. We believe this range to be more than sufficient as the missing and extra strokes should not be so many as to scale the part of the query we wish to align more than 100 times the actual size. In our experiments, we consider 250 discrete samples from the range [0.01,100].

It is important to note that for matching a single LCS vector as described in section 2.4, the matching procedure is done with a specific precomputed α value. If the α value were to change, the vector matching could have a different output. Therefore, it

is computationally expensive to calculate a vector match for each and every potential α' value. The solution is to issue votes based on the vector matching results for each pairing of LCS vectors.

For each LCS vector pairing, the vector matching algorithm computes an α_n and error value. First the error value must be converted into a vote. It is desirable that the lower the error, the more weight a vote should carry. The weight of the vote is calculated using the Gaussian function

$$V(x) = e^{-(x-\mu)^2 / 2\sigma^2} \tag{2.2}$$

with μ=0.0 and σ=0.5. This particular Gaussian is useful since it is maximized at 0.0 error, and caps the maximum weight of each vote to 1.0. The σ value was chosen so that the vote tapers towards a 13% vote if the error is around 1.0. The error calculation from equation 2.1 shows that an error of 1.0 means that on average the query sections are twice or half the size of the model sections.

The second aspect of the voting algorithm is to choose how to place the votes. The goal is to use one α_n and vote calculation to estimate the vote that other α' values would have generated using the vector matching. Given the α_n and error pair, it is converted into α' and vote pairs. Instead of directly calculating the vote for each α' value, an estimation of the vote is done instead. The idea is that α' values close by α_n should have similar votes, but with slightly more error and that α' values that are farther away should have errors too high to be considered. Therefore, votes are estimated for the α' values that are up to approximately 26% larger or smaller than α_n. It was mentioned earlier that the α' values being considered were from the range [.01,100]. The 250 samples in the range are taken from the log linear range [ln(.001),ln(100)] in even increments so that the voting algorithm can vote on α' values which are of a certain scale larger or smaller than its neighbors. Another Gaussian function of the form described in equation 2.2 is used to

modify votes for α' values that are farther away from α_n with μ and σ chosen such that the farthest α' value with a scaling of about 26% will have its vote adjusted to 13% of the original vote.

The last consideration is that each query to model LCS vector pairing can have multiple (α_{ni},error) pairs due to LCS vector pruning described in section 2.4, we do not want to double count votes for α' when α_{ni} values are close in value. Therefore, for each LCS vector pairing, we only count the maximum vote that any α' value generates.

Finally, the votes are counted and the α' value with the highest total votes is chosen. Since we can keep track for each query and model LCS vector pairing which (α_{ni},error) pair contributed the most to a particular α' scale, the entire LCS vector matching can be extracted.

2.5.1 Voting Modifications

There are three modifications to the voting algorithm described in order to improve its performance. All of the modifications adjust the magnitude of the votes. It was observed that many times, the votes would tend towards matching a small portion of the entire model. Figure 2.12 shows a query being matched to a model where it would find a very good match within a small portion of the model.

The motivation for the first modification is to assume that sketches are as close to the model as possible. It is more likely that the sketcher was trying to draw 100% of the shape rather than 10% of the shape. The first modification is dependent on equidistant value between the chosen LCS vectors. The Gaussian function described in equation 2.2 is used again to lower the magnitude of matches that are close together giving the largest possible vote to matchings that spanned more of the model.

The second modification is to counteract stretching the model too large. We allow the query to extend beyond the boundaries of the model in order to account for

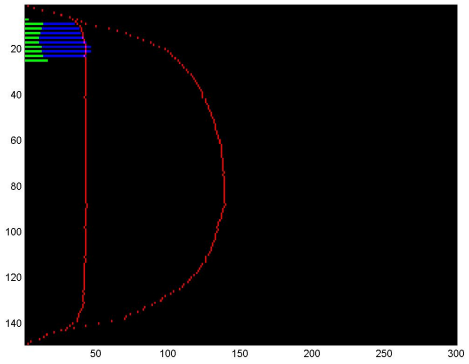

Figure 2.12. XOR gate query to model matching without Gaussian weighting over the span of the sketch

extra pieces. However, this could cause a very good match for just matching part of the shape and disregarding the pieces that are matched to nothing in the model. Therefore, another Gaussian is used to modify the votes, lowering the magnitude of matches that have unmatched LCS vectors.

The motivation for the third modification is to assume that it is less likely that the sketcher made a mistake. The modification gives a lower vote to particular LCS vector matches that needed to combine or prune more sections in the LCS vector matching. If no pruning or combinations were made then it would garner a full vote. Otherwise the vote would be modified using the Gaussian function from equation 2.2 with the number of combined sections divided by the total number of possible combined sections as input. Figure 2.13 shows the matching of an XOR query after these modifications.

2.6 Stroke Identification

Once a best match has been found, it is very easy to classify the missing and extra strokes. The combinations and pruning of each matching of the LCS vectors of the model and query can be saved. For each connection point of combined query sections, it

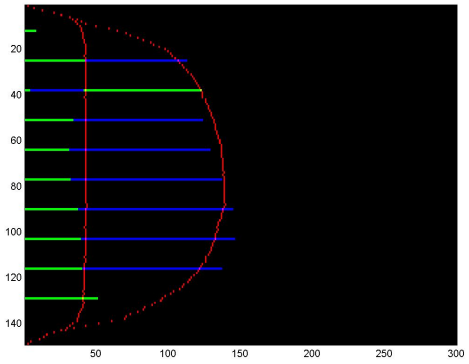

Figure 2.13. XOR gate query to model matching with Gaussian weighting over the span of the sketch

can be voted as an extra stroke. Each connection point of combined model sections is considered a missing point. The original sketch can be searched in the general vicinity of the denoted missing point to see if a stroke point can be found.

We define the general vicinity differently based upon where the missing point is located. If the point is supposed to occur in between a section, we look at an area that is 30% of the section size centered around where the missing point is designated. If the missing point is designated to occur beyond the boundary of the shape, we use the distance from the boundary point to the missing point as the supposed section size. We then look at points that are 40% of that section size centered around where the missing point is designated.

One issue that occurs is that if an example has a lot of LCS vectors that are unmatched, the scale factor could be incorrect. This can happen if the LCS vectors are paired outside of the template boundary, or if the LCS vector is created from a single point and has no values. For example, if an AND gate composed of two strokes is missing a stroke, it could just be a line. The LCS vectors would all be empty and the scale factor is wrong. Therefore, we keep track of how many unmatched LCS vectors there are. The

more unmatched LCS vectors, the less confident we are in the scale factor. Therefore, when looking for missing strokes, we expand the search of missing points outside the shape boundary by

$$1/V([Number\ of\ matched\ LCS\ vectors]/[Total\ number\ of\ LCSvectors]). \quad (2.3)$$

Where V is the Gaussian (equation 2.2) with $\mu = 1$ and $\sigma = 3/5$.

Since a single stroke's points can be present in many different LCS vectors, each LCS vector pairing designates whether the point in a stroke is an extra point, missing point, or part of the sketch. The classified points in a stroke are then counted with the stroke being identified as the majority classification. In the case of a tie, it is denoted as an extra stroke. In the case that a stroke is not present in any of the LCS vectors it will remain as originally classified. This is possible if a stroke lies in the interval between samples.

Chapter 3

Experimental Results

In this section we evaluate the accuracy of identifying extra and missing strokes. The data is taken from the segmentation stage of a sketch recognition program called LogiSketch. The grouping process is described in "Grouping Strokes into Shapes in Hand-Drawn Diagrams" [22]. During their segmentation process, groups are formed using distance and temporal data. Therefore incorrect groupings can happen if gates are drawn too close together, or in quick succession. For their experiments, the found groupings contain all of the correct strokes around 85% of the time [22]. This means that the other 15% contain erroneous inclusions or omittances. We present the accuracy in identifying extra and missing strokes for incorrectly grouped examples. Although our algorithm does not specialize in classification, this chapter will also show the performance of the algorithm when used for classification.

3.1 Stroke Identification Accuracy

Table 3.1. Stroke identification accuracy using brute force vector matching

Omitted Stroke	Extra Included Stroke	Correctly Grouped
50%	93%	94.4%

Table 3.2. Stroke identification accuracy using DTW vector matching

Omitted Stroke	Extra Included Stroke	Correctly Grouped
50%	90.7%	91%

The grouping stage segments the strokes into separate groups. The data is labeled so that the true classifications for each stroke can be known. If a stroke's true classification is associated with a particular gate, but is not in the grouping, it is an omitted stroke. If a stroke's true classification is not associated with a particular gate, but is in the grouping, it is an erroneous inclusion. All other strokes are considered to be correctly grouped. Our method only uses the grouped strokes as inputs. After running our procedure, accuracy is measured by each stroke type (omittance, erroneous inclusion, correctly grouped). Tables 3.1 and 3.2 show the accuracy of identifying omittance and erroneous inclusions and correctly grouped strokes using the brute force and DTW vector matching methods described in section 2.4. Given a grouping with extra strokes grouped, our algorithm is able to correctly identify the extra strokes 93% of the time using brute force vector matching with accuracy slightly reduced when using DTW.

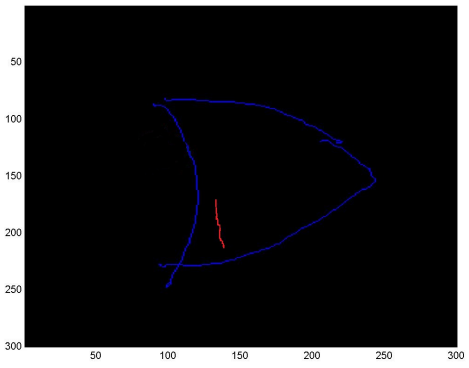

Figure 3.1. OR gate example with missing strokes. Blue strokes indicates correctly grouped strokes. Red strokes indicate missing strokes.

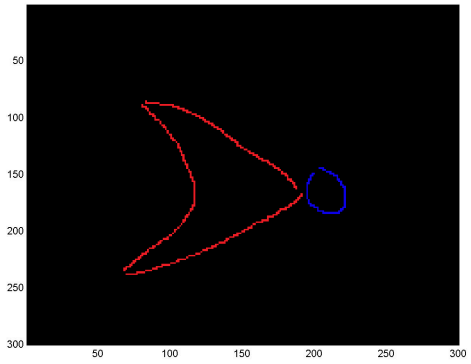

Figure 3.2. NOR gate example with missing strokes. Blue strokes indicates correctly grouped strokes. Red strokes indicate missing strokes.

For the omitted strokes, there was a lack of sufficient data. Out of the approximately 5000 examples gathered, only 14 contained any missing strokes at all. Figures 3.1 and 3.2 show examples of when the missing strokes were misclassified. Figure 3.1 shows an example of a missing stroke that was labeled as part of the gate but clearly is not necessary for classification. Figure 3.2 only uses the bubble as input. Since the grouped stroke contains very minimal data, the algorithm has trouble aligning the bubble to a gate. The bubble would most likely be aligned to the larger part of a gate due to the voting algorithm as described in section 2.5.

In order to gain meaningful results, we synthesized examples to have missing pieces and then ran our algorithm on the synthesized data. We ran two experiments: checking the accuracy when one stroke is randomly removed and checking the accuracy when two strokes are randomly removed from perfectly grouped examples. Tables 3.3 and 3.4 show the results with the synthesized groups.

The tables compare the results of the stroke identification for when the example is matched with its best matching class versus its correct class. This distinction is made to illustrate the example of how an NAND gate without a bubble may look more like an

Table 3.3. Stroke identification accuracy for synthesized groups with one random stroke removed.

	Omitted Stroke	Correctly Grouped
Best Matched Class	60.3%	98.17%
Correct Class	68.84%	98.63%

Table 3.4. Stroke identification accuracy for synthesized groups with two random strokes removed.

	Omitted Stroke	Correctly Grouped
Best Matched Class	63.15%	99%
Correct Class	74.94%	99.06%

AND gate. The identification of missing and extra strokes changes based upon the class that is being matched.

We see that the accuracy is somewhat improved in the synthesized groupings. Examples like Figure 3.2 are still likely to occur in the synthesized groups, in which a bubble would be aligned incorrectly. The increase in accuracy for two missing stroke can be explained by the fact that there are more missing strokes to classify. If an example is likely to find one missing stroke, it is probably also likely to find the second missing stroke as well.

3.2 Classification Accuracy

The matching algorithm outputs a match value for the best alignment of each class. We can also use the match value of a model for classification. We run a classification experiment using our LCS model and Ouyang and Davis' Image Deformation Model (IDM) [20]. The classification experiment is run on data with correct groupings as the IDM would not accurately pre-process examples with erroneous inclusions or omittances.

Figure 3.3 shows for each class, the percent of examples that are correctly classified within the top one and top three best matches. We compare our method with the IDM.

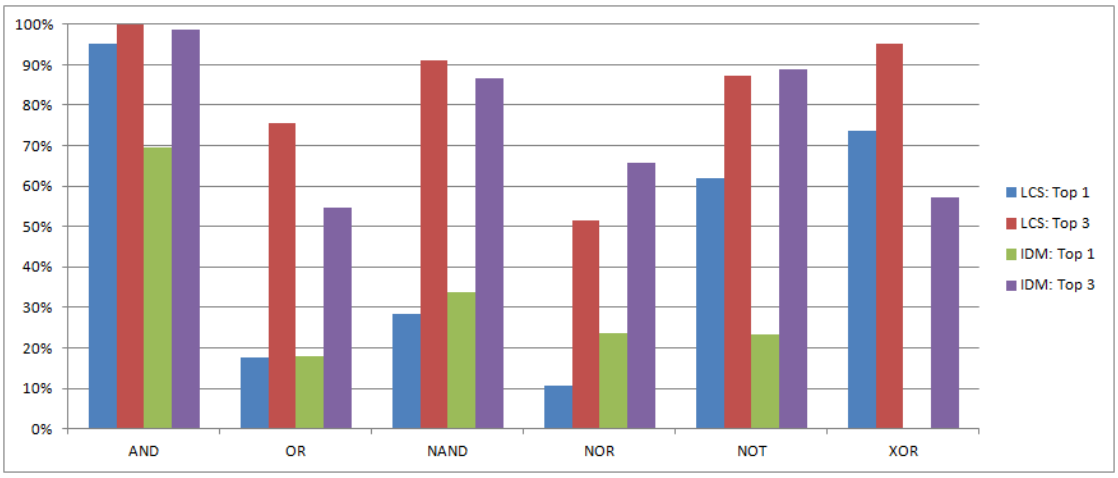

Figure 3.3. LCS vs IDM: Classification Accuracy

Just looking at our method, we can see that for almost every class, over 85% have their actual class within the top three closest models. The exceptions are the OR class and the NOR class. Figure 3.5 shows the confusion matrix for classification of the best matching model. In this table we can observe how the OR and NOR examples are classified. The OR class which gets classified as AND most of the time and the NOR class which is fairly widespread among all the classes. The reason the OR examples are classified as AND is because the shapes are very similar. The reason the NOR class is widespread is because it is the class that shares the most similarities with the other classes. Figure 3 in the Introduction shows how a NOR class could be aligned to 3 different classes.

When comparing our method with the IDM, and looking at the top three closest matches, our method generally has a higher accuracy apart from the NOR gates. Both methods have difficulty with the OR and NOR classifications. Our method has a much higher accuracy with the XOR gate.

Table 3.5. Confusion Matrix

LCS Model

		Predicted Class					
		AND	OR	NAND	NOR	NOT	XOR
Actual Class	AND	95.1%	4.4%	0.3%	0%	0%	0.2%
	OR	65.7%	17.6%	0.1%	0.1%	0%	16.5%
	NAND	64.4%	2.4%	28.4%	2.9%	1.1%	0.8%
	NOR	35.9%	16.8%	19.4%	10.6%	1.1%	16.2%
	NOT	7.9%	4.6%	12.5%	7.1%	62%	5.9%
	XOR	14.9%	11.3%	0.2%	0%	0%	73.6%

IDM

		Predicted Class					
		AND	OR	NAND	NOR	NOT	XOR
Actual Class	AND	69.6%	28.4%	1.3%	0.3%	0.7%	0.3%
	OR	76.1%	18%	2%	1.6%	1.3%	1%
	NAND	19.6%	6.6%	33.9%	37.2%	2.7%	0%
	NOR	23.7%	6.6%	43.1%	23.7%	1.3%	0.7%
	NOT	12.6%	4.3%	45.7%	13.6%	23.2%	0.7%
	XOR	77.4%	18.4%	16.4%	1.3%	1.3%	0%

Even though shape classification is not the main objective of our algorithm, it still performs as well as other shape classification algorithms. This shows that our algorithm could potentially be useful for shape classification in further research.

3.3 Dataset

The complete methodology for acquiring the sketch data is described in "The Effect of Task on Classication Accuracy: Using Gesture Recognition Techniques in Free-Sketch Recognition" [7]. The data used in this study was collected in a laboratory. The data simulates natural sketching styles as participants were asked to produce their own designs [7]. All sketches were done on computer tablets using a program called LogiSketch. LogiSketch is a tool designed at Harvey Mudd College which allows users

to draw a digital logic circuit diagram and label the gate types. The explicit labeling is used to keep track of the true classifications for every stroke. The program distinguishes individual strokes from pen down to pen up and also is able to measure temporal data in addition to the pen pressure at each point. The LogiSketch program currently has a stroke classification algorithm which first identifies strokes as wires or gates, then groups strokes together for symbol classification, followed by the symbol classification. The grouped examples used in this research are from the grouped data using LogiSketch before symbol classification.

Chapter 4

Conclusion

We have demonstrated a working affine invariant group correction algorithm. It is able to very accurately determine erroneous inclusions given a set of pre-grouped strokes, while ignoring minor distortions that are common in sketched symbols. It is also able to identify most missing strokes. Extra strokes are easier to identify since they are part of the inputs while the location of missing strokes are estimated. Future work could focus on improving the identification of missing strokes.

The ability to modify a grouping should improve classification accuracy since classification algorithms are run on the outputs of the grouping steps. Incorrect groupings can cause classification pre-processing steps to misrepresent a symbol. Reducing the frequency of incorrect groupings inherently improves classification.

Our algorithm is designed for group correction, however our algorithm is also adept at classification of symbols in an affine environment. We compared our algorithm's classification accuracy against Ouyang and Davis' IDM model. Our model outperformed the IDM in most cases. This result is probably due to the fact that our method is more sensitive to symbols with minor differences. Our algorithm uses cross section counts in its alignment and match calculation, whereas other algorithms may only use distance metrics in which strokes like the bubble only create a very small distance offset. This is shown for the differences between an AND and NAND gate in which the bubble in front

of the NAND gate could be considered a distortion in the IDM whereas our model would treat it as its own cross section.

Further research for this algorithm could include performance optimizations and parameter tuning. The main bottleneck for computation is the vector matching. The research would include other signal matching methods which could be used for more efficient approximate matches. Individual vector matches are also independent of each other opening the possibility for parallelization.

Bibliography

[1] Anderson, D., Bailey, C., & Skubic, M. (2004). Hidden Markov Model symbol recognition for sketch-based interfaces. In AAAI Fall Symposium (pp. 15-21).

[2] Ankerst, M., Kastenmller, G., Kriegel, H. P., & Seidl, T. (1999, January). 3D shape histograms for similarity search and classification in spatial databases. In Advances in Spatial Databases (pp. 207-226). Springer Berlin Heidelberg.

[3] Bau, T. C. Using Two-Dimensional Gabor Filters for Handwritten Digit Recognition (Doctoral dissertation, M. Sc. thesis, University of California, Irvine).

[4] Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 24(4), 509-522.

[5] Daugman, J. G. (1988). Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression. Acoustics, Speech and Signal Processing, IEEE Transactions on, 36(7), 1169-1179.

[6] Donoser, M., Riemenschneider, H., & Bischof, H. (2010). Efficient partial shape matching of outer contours. In Computer VisionACCV 2009 (pp. 281-292). Springer Berlin Heidelberg.

[7] Field, M., Gordon, S., Peterson, E., Robinson, R., Stahovich, T., & Alvarado, C. (2010). The effect of task on classification accuracy: Using gesture recognition techniques in free-sketch recognition. Computers & Graphics, 34(5), 499-512.

[8] Fu, A. W. C., Keogh, E., Lau, L. Y., Ratanamahatana, C. A., & Wong, R. C. W. (2008). Scaling and time warping in time series querying. The VLDB JournalThe International Journal on Very Large Data Bases, 17(4), 899-921.

[9] Hasegawa, M., & Tabbone, S. (2012, March). Affine invariant shape matching using radon transform and dynamic time warping distance. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (pp. 777-781). ACM.

[10] Hirata, K., & Kato, T. (1992, January). Query by visual example. In Advances in Database TechnologyEDBT'92 (pp. 56-71). Springer Berlin Heidelberg.

[11] Hse, H., & Newton, A. R. (2004, August). Sketched symbol recognition using Zernike moments. In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on (Vol. 1, pp. 367-370). IEEE.

[12] Hse, H., Shilman, M., & Newton, A. R. (2004, January). Robust sketched symbol fragmentation using templates. In Proceedings of the 9th international conference on Intelligent user interfaces (pp. 156-160). ACM.

[13] Kara, L. B., & Stahovich, T. F. (2005). An image-based, trainable symbol recognizer for hand-drawn sketches. Computers & Graphics, 29(4), 501-517.

[14] Lamdan, Y., Schwartz, J. T., & Wolfson, H. J. (1988, June). Object recognition by affine invariant matching. In Computer Vision and Pattern Recognition, 1988. Proceedings CVPR'88., Computer Society Conference on (pp. 335-344). IEEE.

[15] Latecki, L. J., Lakaemper, R., & Wolter, D. (2005). Optimal partial shape similarity. Image and vision computing, 23(2), 227-236.

[16] Learned-Miller, E. G. (2006). Data driven image models through continuous joint alignment. Journal IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(2), 236-250.

[17] Lewis, J. (1995, May). Fast template matching. In Vision interface (Vol. 95, No. 120123, pp. 15-19).

[18] Liang, S., Li, R. H., Baciu, G., Chan, E. C., & Zheng, D. (2010, April). Partial matching of garment panel shapes with dynamic sketching design. In Proceedings of the 1st Augmented Human International Conference (p. 16). ACM.

[19] Mikolajczyk, K., & Schmid, C. (2004). Scale & affine invariant interest point detectors. International journal of computer vision, 60(1), 63-86.

[20] Ouyang, T. Y., & Davis, R. (2009, July). A Visual Approach to Sketched Symbol Recognition. In IJCAI (Vol. 9, pp. 1463-1468).

[21] Osada, R., Funkhouser, T., Chazelle, B., & Dobkin, D. (2002). Shape distributions. ACM Transactions on Graphics (TOG), 21(4), 807-832.

[22] Peterson, E. J., Stahovich, T. F., Doi, E., & Alvarado, C. (2010, April). Grouping Strokes into Shapes in Hand-Drawn Diagrams. In AAAI (Vol. 10, p. 14).

[23] Saund, E., & Lank, E. (2003, November). Stylus input and editing without prior selection of mode. In Proceedings of the 16th annual ACM symposium on User interface software and technology (pp. 213-216). ACM.

[24] Schiele, B., & Crowley, J. L. (2000). Recognition without correspondence using multidimensional receptive field histograms. International Journal of Computer Vision, 36(1), 31-50.

[25] Stevens, M. E. (1961, December). Abstract shape recognition by machine. In Proceedings of the December 12-14, 1961, eastern joint computer conference: computers-key to total systems control (pp. 332-351). ACM.

[26] Sun, Z., Wang, C., Zhang, L., & Zhang, L. (2012, October). Query-adaptive shape topic mining for hand-drawn sketch recognition. In Proceedings of the 20th ACM international conference on Multimedia (pp. 519-528). ACM.

[27] You, Z., & Jain, A. K. (1984). Performance evaluation of shape matching via chord length distribution. Computer vision, graphics, and image processing, 28(2), 185-198.

[28] Zeljkovic, V., Vincelette, R. B., & Savic, M. (2006, September). Efficient shape recognition method using novel metric for complex polygonal shapes. In Proceedings of the 2nd international conference on Mobile multimedia communications (p. 23). ACM.

[29] Zhang, D., & Lu, G. (2004). Review of shape representation and description techniques. Pattern recognition, 37(1), 1-19.