Element-wise Analysis of Free-hand Line Drawings for Psychological Diagnosis

Cihan Chen

Keble College

University of Oxford

A dissertation submitted in partial fulfilment for the degree of

Master of Science in Computer Science

Trinity 2015

Acknowledgements

Firstly, I would like to express my sincere gratitude to my project supervisor Dr. Irina Voiculescu, who has been extremely supportive, responsive and mindful throughout the research progress. Secondly, I would like to thank Professor Leslie Goldberg, my academic supervisor, who shared insights and helped me develop an excellent study plan for the master's course, which I have enjoyed so much. I would also like to thank Professor Glyn Humphreys and Dr. Mihaela Duta in the Department of Experimental Psychology, for their time and advice while setting up the project and providing feedbacks. My thanks also goes to Mr. Brendan Shillingford, who provided invaluable suggestions on feasibilities of applying machine learning techniques in the project. Finally, I would like to thank Ms. Jie Lin, who helped refine written English in this dissertation.

Abstract

A drawing test is used by psychologists to diagnose apraxia, a post-stroke disease, where subjects are asked to replicate a given figure on tablets. An automatic scoring system is needed to remove the human subjectivity and to obtain a qualitative assessment method. Conventional techniques in computer vision and machine learning lack the ability to perform element-wise analysis for geometric free-hand drawings. This dissertation describes an architecture that is capable of recognizing individual elements of line drawings, by combining supervised learning and deterministic algorithms. Firstly the concept of 'symbolic element' is proposed so that certain elements can be recognized by a conventional support vector machine classifier with sliding window, and removed from the drawing. Then line segments are extracted using a dynamic algorithm depending on the given target figure, before being passed to an optimization process to determine a best mapping scheme. The implemented system is tested with real human drawings and the initial experimental results are positive.

Contents

List of Figures

Chapter 1

Introduction

This chapter briefly introduces relevant medical background in psychology, as well as the motivation, scope and objectives of the project. Related work and key challenges are discussed, to justify the general approach taken in this project.

1.1 Background and Motivation

1.1.1 Drawing Test Automation for Psychology Researches

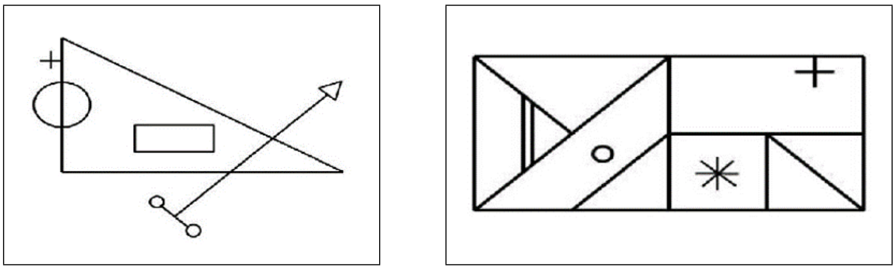
Researchers in the Department of Experimental Psychology have an ongoing study on patients with a variety of post-stroke motor difficulties, including apraxia – the inability to carry out learned motor acts with functional motor and sensory systems[11]. Such impairments are often reflected in the ability of carrying out movements on imitation, thus tasks like hand gesture imitation are widely applied in apraxia diagnosis[6].

One test used by researchers is to imitate a line drawing on a tablet. Drawing tests have been applied in neuropsychological studies for decades and proved to be helpful in assessing a wide range of cognitive impairments. With the widespread usage of touch screen devices such as smartphones and tablets, screening processes can become more quantitative and objective, and patients would be able to perform rehabilitation tasks at home.

However, the assessment and scoring of these drawing tests are still carried out by occupational therapists manually. To score a large number of subject drawings manually is time-consuming and subject to personal judgements. An automatic scoring system is therefore demanded to improve the productivity and eliminate human subjectivity from these tests.

1.1.2 Overview of Screening and Scoring Processes

The existing software used in the screening is a MATLAB GUI application run on Windows 8 tablets with touch screen and stylus. The subject being screened is given a series of tests under the guidance of an operational therapist. The test of concern is to firstly display a black and white line figure on the screen for a short amount of time, and then ask the subject to reproduce the figure from memory on the screen using stylus. Two sample figures are shown in Figure 1.1.



(a) (b)

Figure 1.1: Samples of Target Figures

Several keywords that are used repeatedly while describing this process throughout the dissertation are summarized as follows:

- **Subject**: a person given the clinical test.

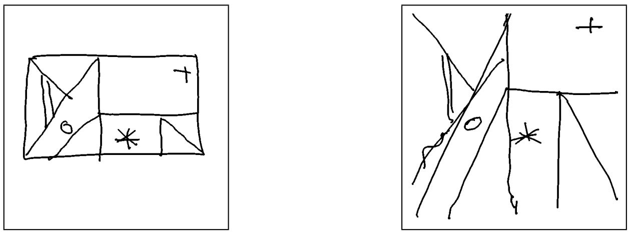
- **Target Figure**: a geometric line figure presented to a subject during the clinical test. The subject is asked to replicate the shown target figure.

- **Stroke**: a continuous mark made by a stylus on the touch screen without lifting.

- **Element**: a geometric pattern or shape defined by the user (researcher or therapist) to be considered an individual object in the target figure. An element can be a simple line segment or a more complex pattern. Its definition is indefinite and subject to user needs.

- **Drawing**: a free-hand line drawing produced by a subject intended to replicate a given target figure. A typical drawing consists of multiple strokes.

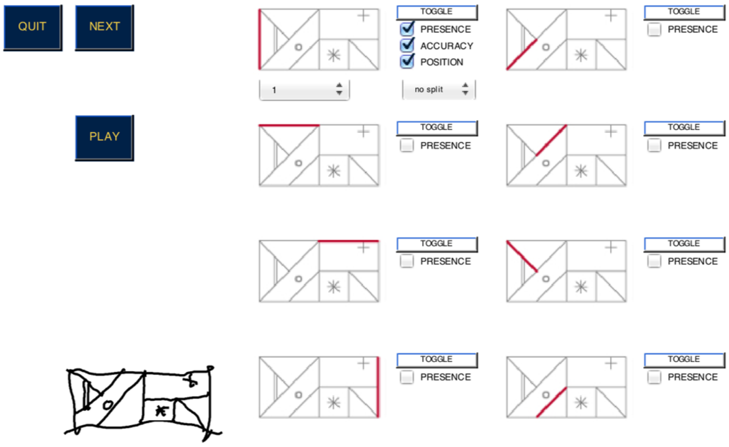
The drawing environment is unconstrained and the subject is not allowed to undo or erase any drawn strokes, because the researchers would like to capture as many subject behaviours as possible. Figure 1.2 shows two sample drawings replicating the target figure shown in Figure 1.1b.



(a) (b)

Figure 1.2: Samples of Subject Drawings

Recorded drawing data are then scored by human experts manually using a GUI application. A cropped screenshot is shown in Figure 1.3. The final drawing is shown at the lower left corner of the screenshot (centre left of the original screen). The scorer needs to tick a series of checkboxes on the right, indicating the presence or absence of all pre-defined elements in the displayed drawing. If an element is present, whether it is good in quality and accurate in position should also be specified. The 'PLAY' button triggers a playback of a video footage, showing the corresponding drawing procedure. The scorer then puts down the sequence in which all present elements are drawn.



☑ PRESENCE
☑ ACCURACY
☑ POSITION
1
no split

Figure 1.3: Cropped Screenshot of Scoring Interface

1.2 Objectives

This project is one of a few pilot projects proposed under the theme to tackle the general problem of computer aided apraxia diagnosis and rehabilitation. The ultimate goal of the bigger study is to compare the performance of different experimental target figures in terms of their effectiveness in identifying cognitive impairments, and to devise an objective assessment method for apraxia. The objectives of this project are as follows.

1. To design a generic architecture that is capable of processing drawings with relatively complex structures.
2. To develop a functional prototype software system that is able to:
   (a) report the presence or absence of all elements in a drawing;
   (b) associate a score with each drawn element, indicating its quality;
   (c) report whether elements are correctly drawn in position;
   (d) output the element-wise sequence of drawing.
3. To briefly investigate and evaluate difficulties in automating drawing replication test for psychological diagnosis.

1.3 Related Work and Major Challenges

1.3.1 Related Work

The task in this project is rarely seen in existing literatures because analyzing replicated drawings is not usually considered a quantitative subject in computer science community and many widely used drawing tests in psychology studies, such as the clock-drawing test[24], are highly abstract and require human intelligence to interpret.

The closest problem is probably the on-line unconstrained handwriting recognition, particularly the recognition for Chinese and Japanese characters, which are essentially line drawings having relatively complex structures. This is considered a challenging task in pattern recognition and statistical methods such as the Hidden Markov Models (HMM) are typically used in most solutions[20]. Recently more advanced models such as Recurrent Neural Networks (RNN) are also applied[10], due to the increasing popularity of connectionist approaches in artificial intelligence community. There are some difficulties, however, to model the task of this project as a conventional recognition problem and solve it end-to-end using statistical models. Some issues are discussed in the next sub-section.

Another related subject is line drawing recognition under the general theme of object recognition. Researches in this area usually concern engineering drawings such as scanned CAD images[4]. Some other attempts investigate methods to convert 2-D line drawings to 3-D models[3]. In both applications, printed or high quality graphs are expected, which are not available in this project. Local features that are used in object recognition[17] may be helpful. A brief investigation of applying point features is covered in Section 1.3.3.

1.3.2 Feasibility of Modelling as a Classification Problem

In theory, it is possible to model the task as a conventional classification problem. One class can be defined for each possible outcome, then a model can be trained to map a drawing to one of the classes. Nevertheless, this approach is not practical due to the following reasons.

First of all, all elements of a given target figure can be either present or absent in a drawing. If there are n elements defined in a target figure, the total number of classes is 2^n. A target figure containing 18 elements, which is the case for Figure 1.1b, would therefore require more than 260 thousands of classes, just for the presence or absence output. The number of classes grows exponentially with the number of elements defined in a target figure, therefore many conventional classifiers are impractical to use.

Secondly, many supervised machine learning techniques require a large number of labelled samples to train a statistical model. A model is typically trained before being deployed to end users. This is, however, infeasible in this project. One of the goals is to discover a good target figure, so the researchers would come up with different figures and compare their effectiveness. Therefore the model training phase, which requires considerable expertise, cannot be performed before deployment. Moreover, each experimental target figure may have only hundreds of sample drawings in total or less. Labelling a portion of them as training set may be far from sufficient, given the large number of classes needed, as mentioned in the first point. As a reference, the Large Scale Visual Recognition Challenge 2014 provides a training set of 1.2 million images for classifying 1000 categories[23].

Finally, and most importantly, it is difficult to obtain **element-wise** information of a drawing from conventional classification results. This includes two sub-tasks:

- To extract the sequence of drawing element-by-element, and
- To evaluate the quality of individual elements.

One possible approach is to classify all frames of the drawing procedure. Namely, a 'snapshot' of the drawing is captured and classified at every timestamp (A snapshot for every new stroke is not sufficient because a single stroke may represent multiple connected elements). This approach is illustrated in Figure 1.4, in which three snapshots are shown, together with corresponding classification results at their lower left corners. The system can conclude that the horizontal line segment is drawn first, followed by the vertical line segment.

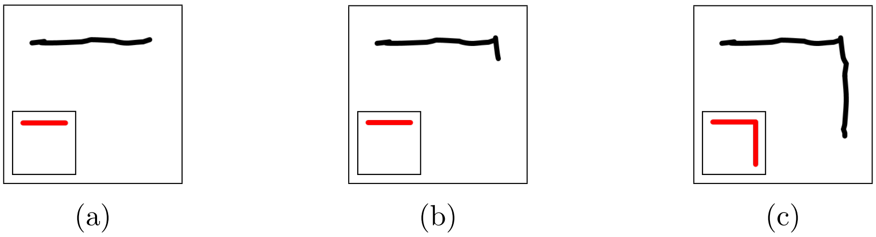
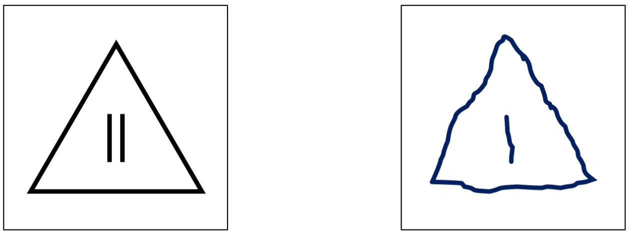


Figure 1.4: Obtain Sequence of Drawing using Conventional Classification

Even if the performance impact of classifying all frames is neglected, such a system is fragile in many cases and extra cares are needed. For instance, if a target figure contains two close vertical line segments, as shown in Figure 1.5a, and a snapshot is captured as in Figure 1.5b, it is not possible to determine which line segment the central stroke represents, until a second vertical stroke is drawn. Reversing the order of classifying the snapshots does not help since the classifier still only knows the fact that two vertical line segments become one at a certain point. Therefore some reasoning on relative positions or an optimization process is unavoidable. Moreover, it is tricky to decide when to perform such analysis or optimization, as the target figure is not hard-coded.

In addition, classifiers are not good at setting boundaries between elements, which is an essential information to evaluate the quality of individual elements. For instance, in Figure 1.4, it is hard to know in which frame the subject starts to draw the vertical line segment, because the classifier does not recognize an element until a significant portion of it is present.



(a) target figure

(b) a snapshot of drawing

Figure 1.5: Ambiguity of Element Identity when Classifying Snapshots

1.3.3 Feasibility of Using Point Features for Matching

One traditional method used in image classification and object recognition is to find matched local patterns and texture correspondences. It essentially establishes many point-to-point associations between two images. If such approaches can find good point correspondences in a drawing and its target figure, the task can be accomplished easily.

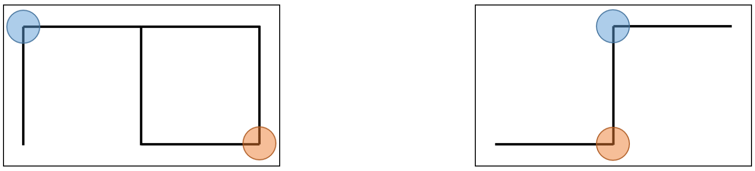
Different local features and kernels are known to be effective in recognizing real-world objects[28]. For example, Berg et al.[2] proposed the use of geometric blur point descriptors and geometric distortion between pairs of feature points to set up correspondences. Although systems relying on point features perform well on finding correlations of elements having non-repeating texture patterns, it does not work for the issues raised in this project because these descriptors typically rely on a strong similarity in local patterns and a weaker global similarity. Neither similarity can be assured in the task of drawn element recognition. Some key points are listed below.

1. Nearby information of a point in a line drawing is little. Typically there is no sufficient texture or gray-scale information to be exploited.

2. It is expected that some elements would be completely missing in a drawing, so local and global characteristics of a pair of corresponding key points can be significantly different.

3. Similarly, subjects may draw extra elements that are not in the target figure. This also affects local and global features of key points in the drawing.

Figure 1.6 illustrates the issue when a typical local descriptor is applied to fetch point correspondences. Circles of same colour indicate a pair of similar points, judged from their local patterns.

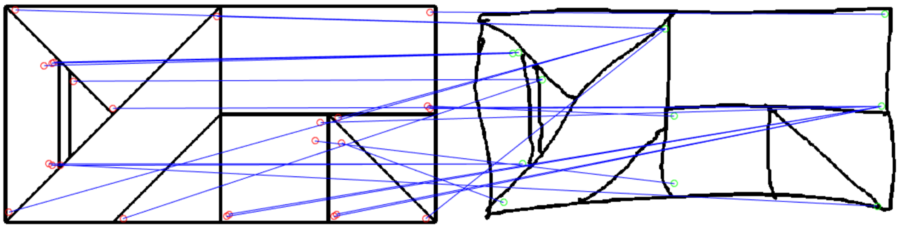


(a) target figure (b) hand drawing

Figure 1.6: Pairs of Similar Points Detected by Local Descriptors

The points highlighted in same colour are clearly not intended to be the same. The local similarities are caused by absent or misplaced line segments. One naïve fix is to iterate all possible classes of the target figure, but it can only potentially solve the problem caused by missing line segments, not extra ones. For example, the drawing in Figure 1.6b has an additional horizontal line segment at lower left, so a good pattern match cannot be found even in the 'correct' class.

The Computer Vision System Toolbox of MATLAB provides some object detection functions based on point features. Figure 1.7 illustrates feature matchings using detected Speeded Up Robust Features (SURF)[1] points, where blue lines represent point correspondences. Although many mismatches are caused by the rotation-invariance of SURF descriptor (which is not needed in this application), some are due to imperfect connections and missing or distorted strokes. For example, the central points of the two images are not associated, because the short diagonal stroke near the centre has an offset. It would be more difficult to find correct correspondences if the subject draws elements that are not defined in the target figure.



left: target figure; right: subject drawing; blue line: correspondence

Figure 1.7: Point Correspondences Detected by SURF Descriptor

To conclude, conventional means of correspondence matching using point features cannot be applied directly to map elements in this project. Nonetheless, they could be useful in detecting symbolic elements, which is described in Section 2.3.

1.3.4 Approach Taken

More advanced machine learning techniques may be able to achieve an element-wise analysis, in particular certain variant of RNN has shown promising performance in sequence labelling tasks such as on-line unconstrained handwriting recognition[10] and sequence generation[9]. These models could potentially be adapted to the drawing comparing task. Unfortunately, unlike the handwriting recognition tasks, which have a rich set of existing databases that can be used as training samples, this project lacks labelled drawings. Synthesizing training samples is a popular workaround to overcome the issue, especially for studies involving textual data. Nevertheless, generating a variety of realistic and complete drawings is a challenging task itself. Moreover, statistical models usually have underlying mechanisms that are more difficult to interpret and understand. In fact, Nguyen et al.[21] showed that deep neural networks achieving state-of-the-art performance on various pattern recognition tasks can make high confidence predictions for unrecognizable images.

On the other hand, computer vision problems are generally considered difficult, so solving the task relying on deterministic algorithms alone seems infeasible. A previous pilot project concerning the drawing test evaluates algorithmic methods for tasks ranging from extracting line segments in strokes, matching elements, to associating scores to individual strokes, for a very simple target figure[12]. Although showing some initial success, many methods proposed can hardly be extended to the more complex target figures used in clinical practise, especially those containing small symbolic elements.

Therefore, a hybrid approach of combining supervised learning and deterministic algorithm is taken in this project, in order to assure that a reasonably functional prototype can be implemented in the time frame of a master's project. Some conventional object detectors with sound mathematical models are trained by synthesized images, in order to identify small recognizable patterns in a drawing. Then an algorithmic approach is taken to extract line segments from strokes. The line segment extraction methodology adopted is inspired by the chain coding scheme proposed by Freeman[7] in 1960s, but the fundamental purpose of encoding is different. An optimization scheme is proposed to match identified line segments to elements in the target figure. Finally, results are combined and outputted.

1.4 Dissertation Structure

This chapter gives an introduction to the problem, identifying its difficulties by reviewing related work and some seemingly working solutions. Chapter 2 provides a module-by-module description of the proposed hybrid architecture that combines supervised learning and deterministic algorithms, in the order of processing. The focus is on the line segment mapping module, where a novel algorithm is proposed to dynamically extract line segments from a human drawing depending on a target figure. Chapter 3 presents sample experimental results of running the system on actual drawings in various typical categories. Then the fourth chapter discusses key choices made while designing the system. Finally Chapter 5 concludes the project and provides suggestions to further develop the system.

Chapter 2

Methodology

This chapter provides an in-depth description of the proposed methodology, with a focus on the line segment mapping technique presented in Section 2.4. The software is written in MATLAB, for compatibility with current applications used in Experimental Psychology Department. Nevertheless, since all algorithms and processes proposed are platform independent, they are presented either mathematically or in pseudo code. The number of MATLAB built-in toolbox functions mentioned has been kept to minimum.

2.1 System Overview

This section formally specifies the system input and output, setting a foundation for the remaining sections of the chapter. It also introduces the key terms used throughout the report. The high-level architecture of the system is also given.

2.1.1 Input/Output Specifications

2.1.1.1 Input One - Target Figure Definition

The first input to the system is a target figure definition specifying the figure that the subjects are asked to replicate.

It is important that a target figure is part of the input. Since the study conducted in Experimental Psychology Department is a continuing research, there does not exist an optimal or standard target figure. Therefore the system is expected to handle different target figures and hard-coding only one target figure is of little use. The system is designed to be as generic as possible, although limitations still exist.

A target figure definition consists of three properties:

1. The first property `IMG_SIZE` defines the size of target figure by $(width, height)$. The size should be reasonably large so all the details are preserved when they are converted into images in the detection phase. The trade-off of a large target figure is high computational complexity in time and space. The resolution of the raw data shown in this dissertation is 569×569, therefore setting a target figure to be much larger than that is unlikely to help.

2. The second property `TARGET_STRUCT` gives the definition of the framework or skeleton structure, by a $N \times 4$ matrix, where N is the number of target line segments. Each row defines a target line segment by (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) represent two end points. Elements defined in this property are typically the longer line segments in the figure.

3. The third property `SYMBOLIC_ELEMENTS` lists all symbolic elements in the target figure, so that an object detector can be applied to detect them. The `Type` field accepts three types of element: 'circle', 'lines', and 'polygon'. If the element is a 'circle', the `Shape` field contains (x, y, r), where x, y indicate the coordinates of the centre of circle, and r its radius. If the `Type` is 'lines', all line segments are represented in the same format as `TARGET_STRUCT`. It typically refers to shorter strokes that belong together and form a symbolic pattern.

Table 2.1 shows one way of defining the target figure illustrated in Figure 1.1b and Figure 2.1 visualizes the definition, in which all target line segments are highlighted in blue and all symbolic elements are highlighted in red squares. Some new terms appear in the definition above and are mentioned frequently throughout the dissertation. They are formally defined as follows.

- **Target Line Segment**: a line segment element defined as a row in `TARGET_STRUCT`.

- **Framework**: the structure specified by `TARGET_STRUCT`, containing all target line segments.

- **Symbolic Element**: or simply **symbol**, an element defined in `SYMBOLIC_ELEMENTS` that can be recognized by a conventional object detector regardless of its position and context. In contrast, a target line segment alone without contextual or positional information is not identifiable.

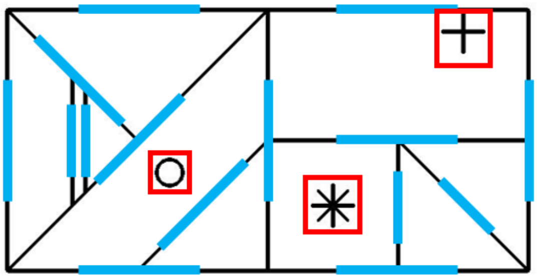
Table 2.1: Target Figure Definition

`IMG_SIZE`	$\begin{bmatrix} 650 & 325 \end{bmatrix}$
`TARGET_STRUCT`	$\begin{bmatrix} 25 & 13 & 25 & 313 \\ 25 & 313 & 325 & 313 \\ 325 & 313 & 625 & 313 \\ 625 & 313 & 625 & 13 \\ \cdots & & & \end{bmatrix}$
`SYMBOLIC_ELEMENTS`	`Type = ‘lines’` // the star `Shape =` $\begin{bmatrix} 400 & 213 & 400 & 263 \\ 375 & 238 & 425 & 238 \\ 382 & 220 & 418 & 256 \\ 382 & 256 & 418 & 220 \end{bmatrix}$
	`Type = ‘circle’` `Shape =` $\begin{bmatrix} 212 & 200 & 15 \end{bmatrix}$;
	`Type = ‘lines’` // the cross `Shape =` $\begin{bmatrix} 550 & 13 & 550 & 63 \\ 525 & 38 & 575 & 38 \end{bmatrix}$

Formally defining the concept of ‘symbolic element’ is crucial to the overall architecture design. It allows the task to be divided into two major sub-tasks: symbol detection and line segment mapping, which greatly simplifies the problem.

The definition of symbolic elements is not exact, and there are usually multiple ways to specify them for a target figure. The only requirement is being recognizable to an ordinary object detector. In the sample definition shown in Table 2.1, three symbols are defined: a star, a cross, and a circle. None of their patterns is present in a perfectly drawn framework of the given target figure, so it is valid to define them as symbols. However, it is possible to also include the two short vertical line segments as a symbol, or to exclude the cross and treat it as two line segments as part of the framework.

Similarly, there are more than one way to define target line segments. In Figure 2.1, 15 target line segments are defined. Both the top and bottom borders are divided into two sections. It is also possible to define them as two long horizontal line segments. The user would need to make the decision based on needs. In general, the more elements are defined, the more descriptive the output would be, because the basic unit of system output is element.



blue: target line segment; red square: symbolic element

Figure 2.1: Visualization of a Target Figure Definition

It is a good practice to define elements with less noisy surroundings as symbols, to achieve better results in the symbol detection phase. For instance, the two vertical line segments are connected with two diagonal line segments, so its environment is noisier than the three symbolic elements specified and therefore difficult to be detected.

In addition, the target figure given should leave some space around the edges, otherwise the symbol detection phase would be affected.

2.1.1.2 Input Two - Array of Subject Drawings

The other input to the system is an array of drawings. The system is designed to read a folder containing multiple `.mat` files, which are the output files of a MATLAB GUI application developed by Experimental Psychology Department for apraxia screening. Each `.mat` file contains the raw data of a single drawing. These files contain some information that is irrelevant to this project and only data that are used in the analysis are mentioned.

Each drawing is an array of strokes (marks that are drawn without lifting the stylus, as defined in Section 1.1.2) in temporal order, each of which is an array of points (x, y), denoting the position of the stylus at a given timestamp. Neither the number of strokes in one drawing, nor the number of points in one stroke is fixed, since the drawing process is unconstrained. In fact, many subjects tend to trace over the same target line segment several times.

In a single stroke, the time interval between two neighbouring points is constant. Therefore the data contain the information of how fast the subject draws. Previous study[12] exploited this information to detect corners, but it is not used in this system. Details of this are covered in Section 2.4.2.

2.1.1.3 System Output

For each drawing, the output of the system is an array of Boolean values indicating whether an element (as defined in Section 1.1.2, can be either a symbol or a target line segment) is present or not. The confidence scores of individual elements are also available, based on which the decisions are made. For all present elements, the sequence in which they are drawn is also given. Figure 2.2 shows a visualization of a final output. Blue lines represent target line segments that are present in the drawing. The numbers in red font starting with a hash annotate the order of drawing. The three red dots indicate the positions of symbolic elements detected.

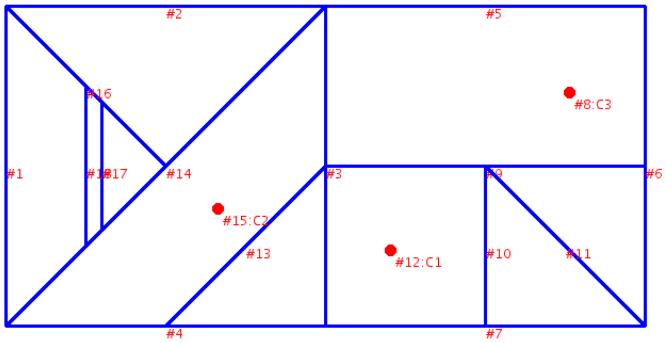


Figure 2.2: Visualization of System Output for One Drawing

2.1.2 Workflow of Processing

The drawing analysis system has a pipelined architecture with a cache mechanism to avoid re-training classifiers for symbol detection. Figure 2.3 illustrates the architecture. There are four general phases in the process: pre-processing, symbol detection, line segment mapping and output construction. The workflow can be summarized as below.

1. Pre-processing
   (a) clean up input data by removing anomalies
   (b) normalize drawings to the same size and aspect ratio as the target figure
2. Symbol Detection
   (a) detect symbolic elements with support vector machine classifiers

3. Line Segment Mapping
    (a) extract line segments from drawn strokes
    (b) map all extracted line segments to target line segments
4. Output Construction
    (a) fetch the sequence of drawing
    (b) test relational positions of detected symbols and apply adjustments

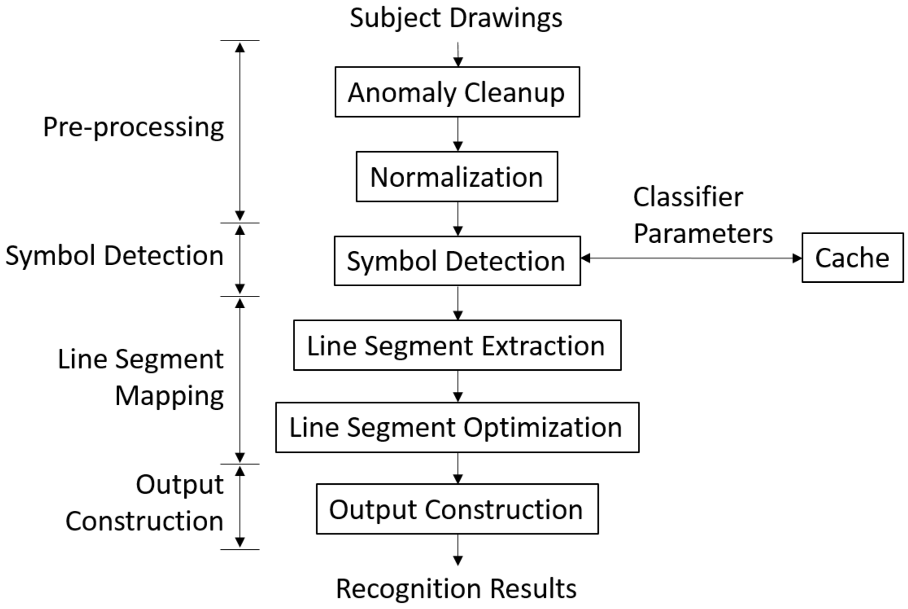


Figure 2.3: System Architecture

The symbol detection and line segment mapping phases perform the heavy lifting in the entire system. Support Vector Machine (SVM) classifiers are used with sliding window to detect symbolic elements. Optionally an additional Cascade Object Detector[25] can be applied to replace the sliding window and boost the detection speed significantly. Parameters of both types of classifier are cached when the initial training is completed. Detected symbolic elements in Step 2a are replaced by marker strokes (specified in Section 2.3.2) before the data are passed to the line segment mapping modules. Line segments are interpreted using a context-aided algorithm, and then all extracted line segments would go through an optimization process in order to produce a best mapping. The output is then constructed by restoring sequence of drawing and relational positions of detected symbols.

2.2 Pre-processing

This section introduces the pre-processing phase, consisting of anomaly clean-up and normalization.

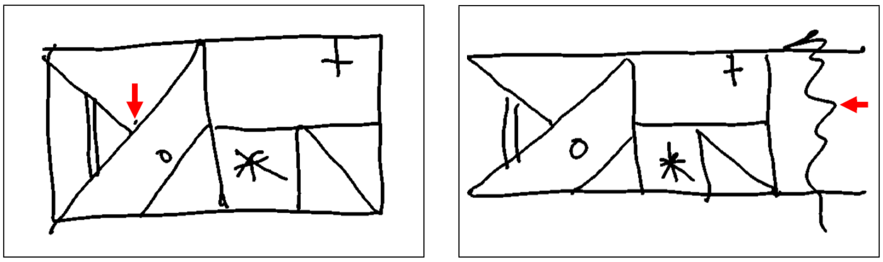
2.2.1 Anomaly Clean-up

Since the raw data are obtained in a completely unconstrained situation, abnormal strokes are very common. There are two main scenarios when abnormal strokes are introduced:

1. The subject accidentally hits the touch screen with stylus, typically resulting an extremely short stroke or a single point.

2. The subject tries to cross out some unwanted sections from previously drawn strokes.

Figure 2.4 shows both types of anomalies annotated by red arrows. Extremely short strokes like the one shown in Figure 2.4a are unlikely to contribute to the drawing and a single point is completely useless. It is therefore safe to remove such strokes without much reasoning and analysis. The default cut-off length is 10 pixels, and any stroke whose length is less than the cut-off length would be removed. The cut-off threshold should be set to be much shorter than the shortest target line segment defined. Removing these strokes makes little visual difference to the drawing.



(a) extremely short stroke

(b) intentional amending stroke

Figure 2.4: Samples of Abnormal Strokes

In Figure 2.4b the subject tries to cross out unwanted sections of two previously drawn strokes. This behaviour is common among free-hand line drawings. Nevertheless, a deterministic algorithm to identify all such strokes is hard to invent. Current system does not handle these extreme cases.

2.2.2 Normalization

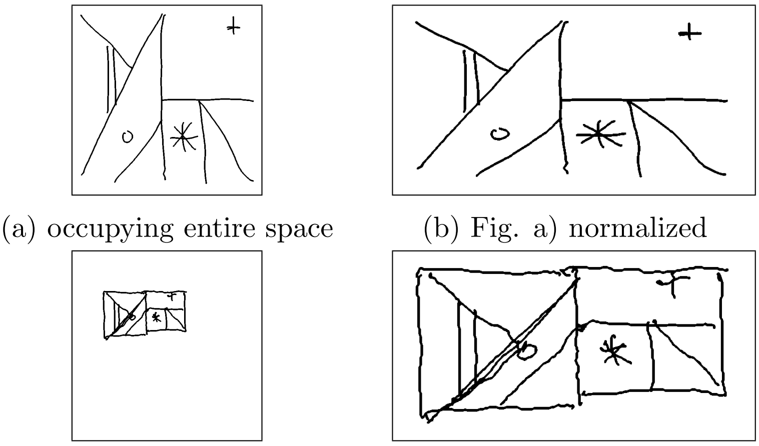
The normalization module zooms and stretches the effective region of drawings linearly on both dimensions so they have the same size and aspect ratio as the defined target figure. This process improves the accuracy of subsequent phases because most of the strokes would be moved or stretched to be closer towards the corresponding elements.

Each point (x, y) in a drawing is updated to (x', y') according to the formulas:

$$x' = \frac{(x - x_{min})(1 - 2R)W}{(x_{max} - x_{min})} + R \cdot W \tag{2.1}$$

$$y' = \frac{(y - y_{min})(1 - 2R)H}{(y_{max} - y_{min})} + R \cdot H \tag{2.2}$$

where W and H are the width and height of the target figure, correspondingly. R is the edge ratio, whose default is 0.05, meaning 5% of marginal space is left empty in each dimension. This is to allow a better rendering result in the symbol detection phase, which is also the reason to have marginal space in target figure definition. Figure 2.5 shows the effect of normalization for two samples.



(a) occupying entire space

(b) Fig. a) normalized

(c) drawing at a corner

(d) Fig. c) normalized

Figure 2.5: Samples of Bilinear Normalization

There exists more advanced techniques of normalization in the field of handwritten characters recognition, especially those for Chinese and Japanese characters, which are more similar to drawings than most alphabetical letters. Suggestions on future development are briefly discussed in Section 5.2.

2.3 Detection of Symbolic Elements

Methods of detecting small symbolic elements are covered in this section. The key challenges are to train statistical classifiers using synthesized images, and to eliminate false positive using geometric reasoning. Symbols detected are replaced by marker strokes before the drawing is passed to the next step.

2.3.1 Training Support Vector Machine Classifiers

One linear SVM classifier is trained for each symbolic element. The major difficulty is to come up with sufficient training data. Similar problems in handwriting recognition and object detection usually have a large number of real labelled training data, which are not available in this project, as mentioned in Section 1.3. Fortunately, although generating complete drawings as training set to accomplish the general element-wise recognition is very difficult, for the simplified sub-task of detecting symbolic elements, classifiers trained by synthesized samples can actually achieve a satisfactory performance.

2.3.1.1 Generation of Positive Training Data

To train the classifiers, labelled images are synthesized automatically according to the symbol definitions. Figure 2.6 shows several sample positive training images generated for star, circle and triangle (defined as a polygon) elements. A random polygon image is produced by introducing different degrees of randomness to coordinates of its vertices. Circles are drawn as curved line segments, with 7 turning points. They are essentially octagons, with the possibility of having an imperfect closing, to imitate how most people draw a circle. The 4th and 5th images in the second row in Figure 2.6 illustrate the imperfect closing. Elements constructed by line segments, such as the star and the cross in Figure 2.1, are handled similarly, by introducing randomness to all end points. It is worth noting that synthesized line segment symbols may not preserve their shapes well because intersections of line segments are not maintained. The last two samples of star in Figure 2.7 demonstrate this phenomenon.

In Figure 2.6, the thickness of lines increases gradually from samples on the left to samples on the right. It is important to have a variety of thickness in the training set, in addition to the morphological randomness. The size of symbols in human drawings varies but will be stretched to the same size before being passed to the SVM classifier. As a result, the line thickness of images fed into the classifiers would be different. The training samples have to reflect this, in order to obtain robust classifiers.

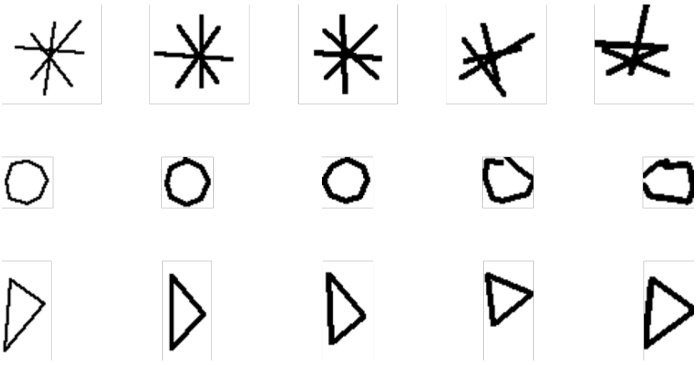
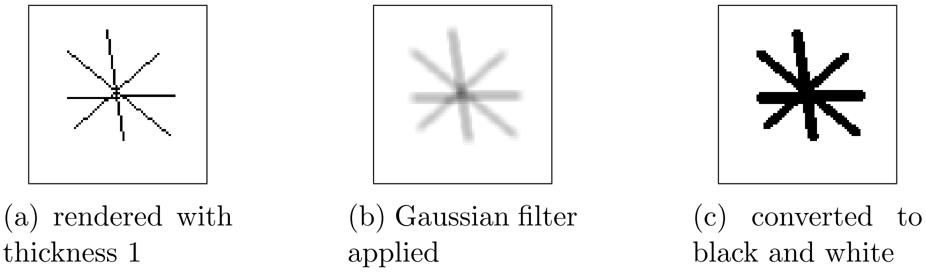


Figure 2.6: Samples of Generated Positive Training Images

The process of line thickening is illustrated in Figure 2.7. Figure 2.7a is a star rendered by line segments of thickness 1; Figure 2.7b is the result of applying a 5 × 5 2-D Gaussian filter provided by the Image Processing Toolbox in MATLAB. Then the image is converted into black and white using a threshold of 0.9, meaning all pixels whose darkness are greater than 10% of the darkest color in the image are turned black. The threshold can be computed dynamically using Otsu's method[22], which minimizes intraclass variance of the black and white pixels, but a constant 0.9 turns out to be sufficient for the purpose here. This process is better than directly setting the line width while rendering, because it gives smoother edges that make the generated drawings more natural, thus gaining similarities to real drawings.



(a) rendered with thickness 1

(b) Gaussian filter applied

(c) converted to black and white

Figure 2.7: Thickening of a Generated Star Element

2.3.1.2 Generation of Negative Training Data

Part of negative samples are generated using the `TARGET_STRUCT` property of target figure definition. These generated images are shared as a common source of negative training set for all symbols, because they do not contain any of the symbolic elements. Some samples are shown in Figure 2.8. End points of line segments are shifted by random values and the shifting is key points based. Namely, if two or more segments share a common end point (called a key point), the generated images would preserve this property. This prevents the system from generating positive samples in local areas by accident. For instance, if complete randomness is allowed to apply to all end points, it is unavoidable that shapes like the star and cross would naturally appear. Preserving connections on non-end points are more difficult, so it is not implemented.

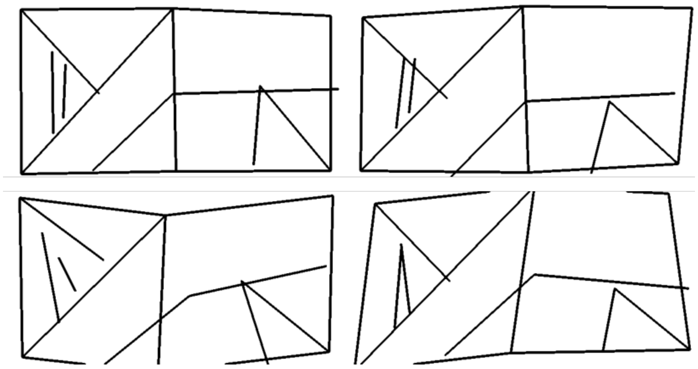


Figure 2.8: Samples of Generated Negative Images

Since the generated images are of the same size as the target figure, they are not used directly as negative samples in SVM training. Actual training images are obtained by cropping the images to the size of sliding window applied later, excluding empty images. A single SVM classifier is used to separate two classes, so each classifier can only be used to detect one symbolic element.

For each symbol, positive examples for other symbolic elements are added into its negative training set. For instance, the negative training set of the star classifier contains all positive samples of circle and cross for a target figure defined in Table 2.1.

2.3.1.3 Training and Tuning Linear SVM Classifier

Positive and Negative images obtained are saved in the cache folder. They are converted to feature vectors using Histogram of Oriented Gradients (HOG) feature extractor, with a default cell size of 8 × 8. Discussions on choosing a proper feature extractor and setting parameters are covered in Section 4.2.

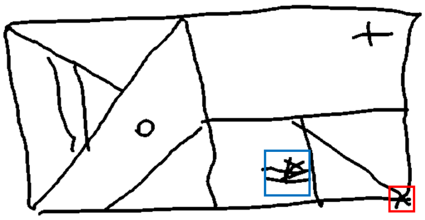
Classifiers are trained using the `fitcsvm` function provided by the Statistics and Machine Learning Toolbox of MATLAB, following the recommended routines[18] to cross-validate and tune the parameters. The cross-validation and tuning are not performed by the system as part of the workflow. A stand-alone subroutine is responsible for selecting the best performing parameters, using the target figure in Figure 2.1. Determined parameters are applied as constants. All potential target figures used in the study are likely to be very similar in terms of visual characteristics, so retuning parameters of SVM classifier for individual target figures is not necessary.

Since training is a time-consuming task, trained classifier objects are saved in the cache folder so they can be reused for future drawings of the same target figure.

2.3.2 Symbol Detection and Removal

A sliding window is used to scan the drawing and classify the image within the window using the trained SVM classifiers. By default, three different window sizes are used: 150%, 100%, and 70% of defined symbol size, to cope with the variations of human drawings. Although the drawings have been normalized to the same size and aspect ratio, the relative size of elements still varies. The variation is not significant due to the normalization, so three scales can handle most cases. The default step size of sliding window is 10% of the window size along each dimension.

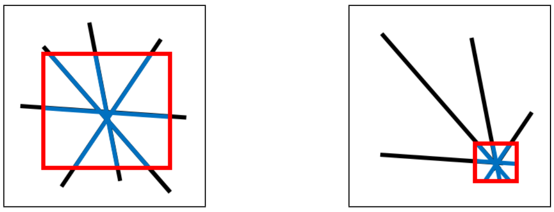
One issue of using sliding window is its lack of high-level understanding. A sliding window might detect a perfectly matched pattern which is in fact a false positive. In Figure 2.9, the false positive for the star element in the red window would obtain a much higher score than the true positive in the blue window.



red window: false positive; blue window: true positive

Figure 2.9: True and False Positives Detected by Sliding Window

In order to address the problem, an additional subroutine is used to filter out samples that are unlikely to be symbolic elements. Such a filter is designed based on an intuitive observation: true positive samples are likely to be composed by dedicated strokes; while local patterns that look like symbols are typically just small portions of parent strokes. The observation is illustrated in Figure 2.10. In typical true positives, sliding window would contain the majority portions of contributing strokes, as shown in Figure 2.10a. In contrast, false positives such as the one illustrated in Figure 2.10b would only capture a small fraction of contributing strokes. Therefore, by calculating the portions of strokes that contribute to a potential match, obvious false positives can be eliminated.



(a) true positive (b) false positive

Figure 2.10: Contributing Strokes of True and False Positives

The algorithm used to detect symbolic elements for a single drawing is summarized as pseudo-code in Algorithm 1. In the pseudo-code, *symbols* is an array of symbolic elements defined, *windowSizes* is a configurable array of window sizes that are used. `STEP_SIZE`, `RATIO_THRESHOLD_1`, and `RATIO_THRESHOLD_2` are configurable constants used as step size along both dimensions, threshold to determine whether a stroke contributes to the detected element, and threshold to determine whether the detected symbol is a false positive of the kind described previously, respectively. Both ratio thresholds have defaults of 40%.

For each symbolic element, the algorithm runs a full sliding window scan for each defined window size. So the time complexity increases linearly with the number of symbolic element defined, as well as the number of different window sizes specified. Such a design helps to solve ambiguity of visually similar symbols by removing the first detected symbol before detecting the second. Therefore, it is helpful to define the symbolic elements in the order of confidence to detect, in descending order.

The method applied in Algorithm 1 to filter out false positive is summarized below.

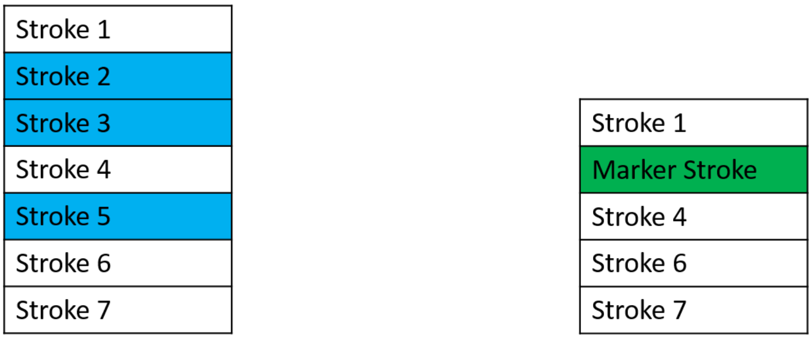
- Step 1. Find all positive samples and store them in a set;
- Step 2. Fetch the sample with the highest score, record the inside-window portions of all intersected strokes, and store all strokes that contribute to the element detected;
- Step 3. Test whether it is a true positive, by the method detailed below;
- Step 4. If it is a true positive, remove all strokes found in Step 1 from the drawing and insert a marker stroke. If it is a false positive, remove the sample from the set and repeat Step 2-4 until the set is empty, in which case report the element being absent.

In Step 2, for all strokes that intersect with the window, a ratio is calculated as the length of stroke inside the window divided by the length of the stroke. If the ratio exceeds `RATIO_THRESHOLD_1`, the stroke is considered a contributing stroke.

To test whether a sample is true positive in the third step, the average value of all ratios (not just ratios of contributing strokes) obtained in Step 2 is compared with `RATIO_THRESHOLD_2`. If the average is larger, the sample is considered a true positive.

Finally, contributing strokes of detected symbols identified in Step 2 are removed from the drawing data to simplify the subsequent steps. A **marker stroke**, which specifies the type and centre point of detected symbol, is inserted to the position of the first stroke among all removed ones, in order to reconstruct the sequence of drawing in the final stage of processing.

A marker stroke is a fake stroke used to represent a detected symbolic element. The x coordinate of the first point of a marker stroke is set to infinity, to signal that it is a marker stroke. The y coordinate of the first point specifies the internal index of the detected symbol. The second point records the centre position of the window in which the symbol is recognized. Figure 2.11 illustrates the replacement process. In Figure 2.11a, contributing strokes of the detected symbol are highlighted in blue. All such strokes are removed from the drawing, and a marker stroke is inserted to the position of Stroke 2, which is the first contributing stroke of the detected symbolic element.



(a) drawing before replacement

(b) drawing after replacement

Figure 2.11: Marker Stroke Replacement

Algorithm 1 Symbol Detection and Removal

```
procedure SymbolDetection
    for each symbol in symbols do
        classifier = fetch/train SVM classifier for symbol
        resultArray = ()                          ▷ empty array of item(score, box)

        for each winsize in windowSizes do
            scan and crop image to winsize for each STEP_SIZE
            obtain HOG feature vectors for all cropped window images
            classify all vectors
            append score (score) and window box (box) of positives to resultArray
        end for

        while resultArray is not empty do
            maxItem = item with highest score
            strokes = ()                          ▷ strokes that contribute to symbol
            portions = ()          ▷ portions of all strokes that intersect with box
            for each stroke in drawing do
                if stroke intersects with maxItem.box then
                    ratio = portion of stroke inside box
                    append ratio to portions
                    if ratio >RATIO_THRESHOLD_1 then
                        append stroke to strokes
                    end if
                end if
            end for

            if Average(portions) <RATIO_THRESHOLD_2 then
                remove maxItem from resultArray
            else
                replace earliest stroke in strokes with a marker stroke in drawing
                remove all the other strokes in strokes from drawing
                break
            end if
        end while
    end for
end procedure
```

2.3.3 Use of Additional Cascade Object Detectors

The performance of SVM classifiers used with sliding window is the bottleneck of the general system performance, where a large number of cropped images need to be classified. The system therefore provides an option that allows the user to replace the sliding window of SVM classifier with a Cascade Object Detector to significantly reduce the number of images passed to the SVM classifier.

The implementation of Cascade Object Detector in Computer Vision Toolbox of MATLAB is used, which cannot be applied alone to detect symbolic elements, because the detector does not return a score associated with a decision. Therefore the detector is used as a replacement of sliding window.

Although the detector itself uses a sliding window as well, it performs multi-stage processing procedures, where negative samples are rejected as early as possible, resulting in a rapid detection speed[25]. Stages of the cascade detector are set to have low false negative rates and relatively high false positive rates. In this application, the vast majority of images considered will be obviously negative, so the cascade detector would filter out a great number of true negatives rapidly, and pass a limited number of potential positive images to SVM for more refined classifications.

2.4 Line Segment Mapping

A novel algorithm, the context-aided line segment extraction method described in Section 2.4.3, is proposed in this section to dynamically extract line segments from strokes, in order to tackle the element-wise mapping problem. The proposed method is able to handle absent, extra, or repeated line segments in a drawing, which is the major challenge of the project.

2.4.1 Overview of Line Segment Mapping

Once the symbolic elements are successfully detected and replaced by marker strokes, the original problem is reduced to mapping drawn strokes to target line segments that construct the framework (as defined in Section 2.1.1.1) of target figure. Nonetheless, unlike detecting the symbolic elements in preceding steps, mapping line segments requires more relational information. This task consists of two steps:

1. **Line Segments Extraction**: A single stroke does not necessarily correspond to a single target line segment. It is very common for humans to draw connected line segments in one stroke, like the outline borders of Figure 2.1. It is therefore important to simplify all strokes into sets of line segments first.

2. **Line Segments Mapping**: After all line segments in the drawing are extracted, they are compared with all target line segments. There are usually multiple ways to map all line segments so an optimization is required to search for a 'best' mapping.

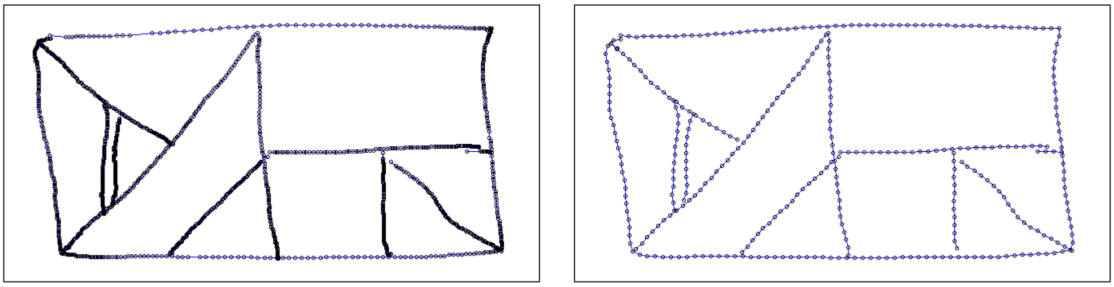
Line segments that are extracted in the first step are refereed to as **extracted line segments** throughout the report, to be distinguished from the **target line segments** specified in the target figure definition. An extracted line segment is obtained from a subject drawn stroke, which may intentionally represent one target line segment, or part of one.

The term '**mapping**' is used to represent a scheme of associating every extracted line segment to a target line segment. The association between a single extracted line segment and a target line segment is referred to as an **assignment**.

2.4.2 Stroke Sampling for Uniform Point Density

The raw data of a stroke is a sequential series of points with constant temporal interval. Information of how fast the stroke has been drawn can be deduced. Some initial attempts[12] to split strokes into line segments exploit this information to detect corners, because it is typical that a subject would slow down while drawing corners. Although the observation can be helpful in some cases, the proposed system in this project does not leverage this assumption since it is not universally applicable. For example, subjects with better sketching skills may not slow down while drawing corners, and people who are used to write ideographs that contain square-like characters may also draw corners quickly. It is risky to apply the observation obtained from a small number of biased samples, which may end up with a system that does not generalize well.

Therefore, all strokes are processed based on geometric information alone. They are sampled at a constant translational interval. Namely, a stroke is fragmented into a series of very short connected line segments having the same length. Such short line segments are referred to as **fragments**. This process essentially makes the information density along all strokes uniform, and flattens wavy strokes to some extent. Figure 2.12 shows the effect of such sampling using an interval of 10 pixels, where small circles represent points in strokes. In Figure 2.12a, strokes have non-uniform point densities. In particular, the subject tends to draw horizontal strokes quickly and slows down when drawing vertical and diagonal strokes. In Figure 2.12b, such speed information is removed, and the total number of points is greatly reduced, without distorting the original structure of the drawing.



(a) before stroke sampling

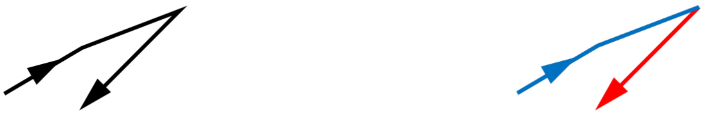
(b) after stroke sampling

Figure 2.12: Visualization of Stroke Sampling

It may be attempting to apply the stroke sampling at the earlier stage of pre-processing. This is risky because symbolic elements are usually small in size and tend to have curved or circular structures. Sampling strokes before detecting these elements may distort the symbols and therefore has an impact on detection accuracy.

Algorithm 2 describes the sampling procedure for a single drawing in pseudo-code. Its time complexity is roughly linear to the number of points in a drawing. The default translational interval, `INTERVAL`, is 10 pixels. It should be set to a value that achieves a good balance by trading some pattern details for reduction of the total number of points. The process would preserve the drawing perfectly while smoothing wiggling strokes, if most of the removed information is either redundancy or noise.

The code in line 16 to 25 of Algorithm 2 splits a stroke into two when the turning angle exceeds `ANGLE_THRESHOLD`, which is 130° by default. The value should be set between 90° and 180°, to represent a significant turning in direction. The effects of this procedure are illustrated in Figure 2.13. This step is important for retaining the original patterns of strokes, when the method of line segment extraction described in the following section is applied.



(a) before: a single stroke

(b) after: two separate strokes

Figure 2.13: Splitting Stroke at a Turning Point

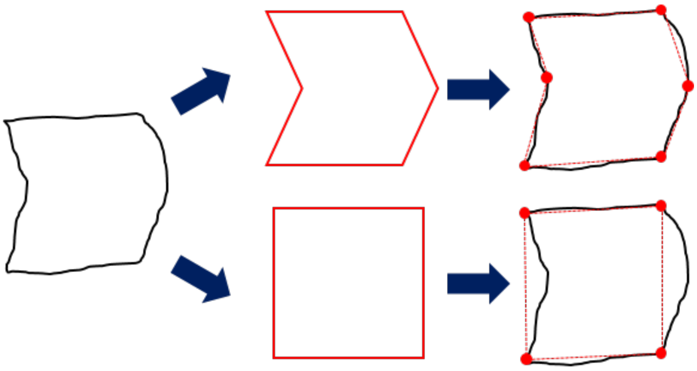
2.4.3 Context-aided Line Segment Extraction

A key idea of line segment extraction is that, it cannot be done context-free. Since all the drawings are attempts to replicate a given target figure, it is ambiguous to identify intentional line segments in free-hand drawings without any information of the target figure. The phenomenon is illustrated in Figure 2.14. The same free-hand drawing can be interpreted as a relatively good drawing of an arrow-like polygon, or a poor quality square, depending on the target figure provided.

Therefore, to extract meaningful line segments from strokes, the target figure has to be supplied, hence the name 'Context-aided Line Segment Extraction'. Notice that the purpose of this stage is not to map drawn strokes to target line segments, but to extract line segments from the strokes. For each fragment in a stroke, the closest match to any target line segment is found by searching for the minimum cost

Algorithm 2 Stroke Sampling

```
procedure StrokeSampling
    newDrawing = ()                    ▷ initialize empty array for fragmented drawing
    for each stroke in drawing do
        if stroke is a marker stroke then              ▷ skip detected symbols
            continue
        end if

        newStroke = ()                        ▷ empty array for fragmented stroke
        index = 2                                  ▷ index for scanning strokes
        prePoint = 1                               ▷ index for preceding point

        while index <= size(stroke) do
            length = distance(stroke[index], stroke[prePoint]) ▷ euclidean distance

            if length >INTERVAL then
                if size(newStroke) >= 2 then
                    segment1 = segment(stroke[index], newStroke[end])
                    segment2 = segment(newStroke[end], newStroke[end - 1])
                    turningAngle = angle(segment1, segment2)

                    if turningAngle >ANGLE_THREDSHOLD then
                        newDrawing.append(newStroke)
                        newStroke = (newStroke[end])             ▷ reset array
                    end if
                end if

                unitVector = (stroke[index] - stroke[prePoint]) / length
                stroke[index] = stroke[prePoint] + unitVector * INTERVAL
                newStroke.append(stroke[index])
                prePoint = index
            end if
            index++
        end while
        newDrawing.append(newStroke)
    end for
end procedure
```



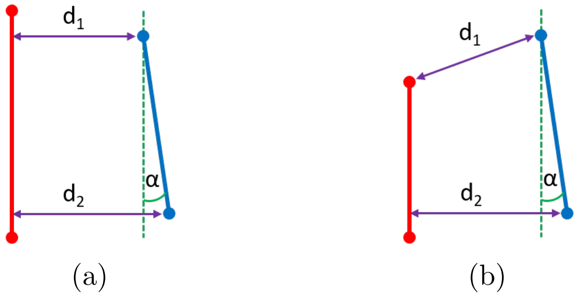
left: free-hand drawing; middle: target figures; right: line segment interpretations

Figure 2.14: Ambiguity of Line Segment Interpretation

of matching. The cost for a fragment i to be matched to a target line segment j is defined as:

$$C_{ij} = \begin{cases} d_{ij1} + d_{ij2} & \text{for } \alpha_{ij} < \min_j \alpha_{ij} + \varepsilon \\ +\infty & \text{otherwise} \end{cases} \tag{2.3}$$

where d_{ij1} and d_{ij2} are euclidean (point-to-line) distances from end points of fragment i to target line segment j, and α_{ij} is the separation angle between fragment i and target line segment j. ε is a constant defining tolerance. Figure 2.15 illustrates the calculation in two cases.



(a)

(b)

red: target line segment; blue: extracted line segment

Figure 2.15: Calculating Cost for a Matching

The term $\min_j \alpha_{ij}$ in Equation 2.3 represents the smallest angle formed by fragment i and any target line segment j, which is guaranteed to be less or equal to 90°. Essentially, the cost is meaningful if and only if the slopes of fragment and target line segment are the closest, allowing a tolerance of ε, whose default value is 0.2 rad, roughly 10°.

The index of target line segment that yields the lowest cost is considerd the best match J_i, namely:

$$J_i = \arg\min_j C_{ij} \tag{2.4}$$

Then every fragment is associated with an index J_i, representing its matching target line segment. A visualization of a concrete example of matching is shown in Figure 2.16, in which blue line segments connect fragments with their best matching target line segments.

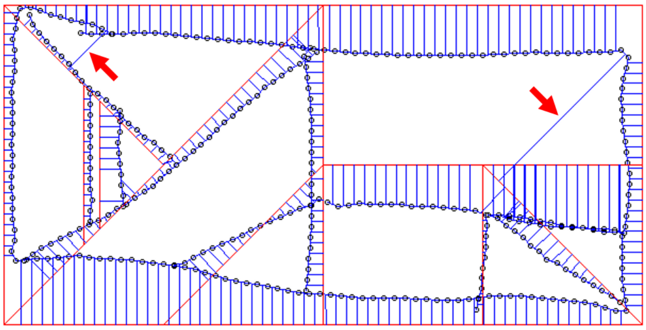
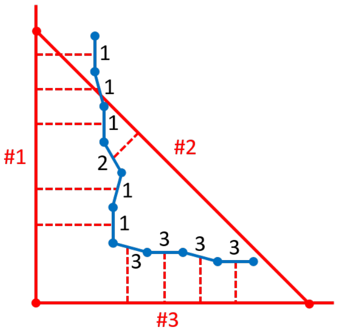


Figure 2.16: Visualization of Fragment Matching

Each stroke is encoded by a matched index array containing all J values of its fragments. A sample stroke encoded [1, 1, 1, 2, 1, 1, 3, 3, 3, 3] is demonstrated in Figure 2.17. Ideally, such a stroke should be interpreted as two line segments, with one matched to '1' and the other matched to '3'. The '2' in the array is a typical noise caused by local wiggling, which can also be seen in Figure 2.16, indicated by red arrows.



blue line: fragment; red solid line: target line segment;
#1, #2 and #3: indices of target line segments;
1, 2 and 3: elements in index matching array.

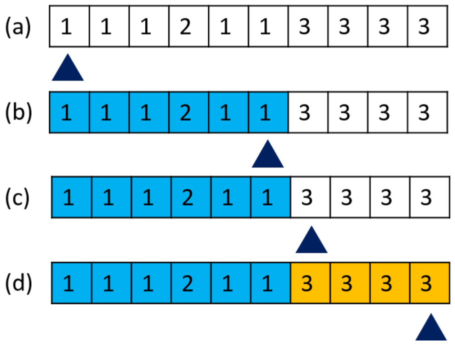
Figure 2.17: Matched Index Array of a Stroke

The process to assign indices to fragments may seem like the chain code shape boundary representation proposed by Freeman[8], but has some fundamental differences, which are further investigated in Section 4.1.1.

The system applies a simple grouping algorithm to split a stroke into multiple line segments, or interpret it as a single line segment. The pseudo-code to process one stroke is shown in Algorithm 3. The time complexity is linear to the number of fragments in a stroke.

Algorithm 3 Matched Index Grouping

```
procedure FragmentGrouping
    $newStroke = (stroke[1])$                                        ▷ set first element
    $index = 1$
    while $index <=$ length($matchedIndexArr$) do
        $targetLength$ = length of target line segment $matchedIndexArr[index]$
        $upperLimit$ = min(($index$ + STOP_RATE * $targetLength$ / INTERVAL), length($matchedIndexArr$))
        $endPoint$ = find index of the last occurrence of $matchedIndexArr[index]$ in the range of $matchedIndexArr[index:upperLimit]$
        $newStroke$.append($stroke[endPoint]$)
        $index = endPoint$
    end while
end procedure
```



(a) step 1: fetch first element ‘1’; (b) step 2: search the last occurrence of ‘1’;

(c) step 3: fetch next element ‘3’; (d) step 4: search the last occurrence of ‘3’.

Figure 2.18: Steps of Grouping Fragments

Figure 2.18 shows the steps of grouping the stroke in Figure 2.17, using the matched index array. The constant STOP_RATE in Algorithm 3 is used to define an upper limit for the range to search, which is 1.4 by default. It tells the system not to search beyond 1.4 times the length of the matching target line segment as defined in the target figure, which makes sense since the drawing has been normalized. This is a simple mechanism to avoid errors when a stroke is a loop. The outline frame of Figure 2.16, whose order of drawing is shown in Figure 2.19, is a good example of loop stroke. Such a circular stroke would have a matched index array like [1, 1, 1, 1, 2, 2, ... 6, 6, 1, 1, 1]. If there is no limitation applied on ranges to search, the entire stroke would be recognized as a single line segment, which is obviously incorrect.

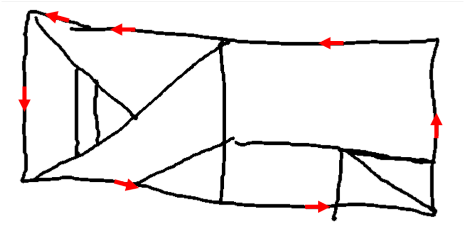


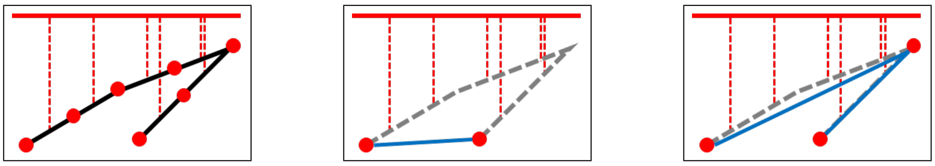
Figure 2.19: Outline Drawn with a Single Circular Stroke

Once the grouping of fragments is determined, all fragments belonging to the same group are replaced by a line segment, whose end points are simply the first and last points of the group. More sophisticated methods such as linear regression can be used to fit a line to all the points within a group, but it may change some important relationships (e.g. how line segments are connected) between line segments by altering positions of end points.

The algorithm is simple and efficient in grouping fragments, and robust enough to neglect some infrequent noise. Line segments extracted by this method are relatively correct. However, without the stroke splitting process described in Line 16-25 of Algorithm 2 and Figure 2.13, the correctness can be significantly affected.

It is typical for a subject to draw a short, sharp, hook-like shape at the ends of a stroke, similar to the ones shown in Figure 2.13a. Fragments in a 'hook' are likely to be grouped together, due to their similarities in slope and position. The effect is demonstrated in Figure 2.20. Without the process of splitting such a stroke into two, extracted line segments will not preserve the original shape of the stroke. Having multiple extracted line segments matched to a single target line segment is taken care of in the optimization process, so there is no need to remove any duplicate or repeated line segments that are close to each other at this stage.

Figure 2.21 shows the line segment extraction process presented by far, applied to a good drawing and a significantly distorted sample, in separate rows. These samples illustrate how the algorithms extract line segments from the original drawings while reducing the amount of data to process and preserving the patterns well. The target



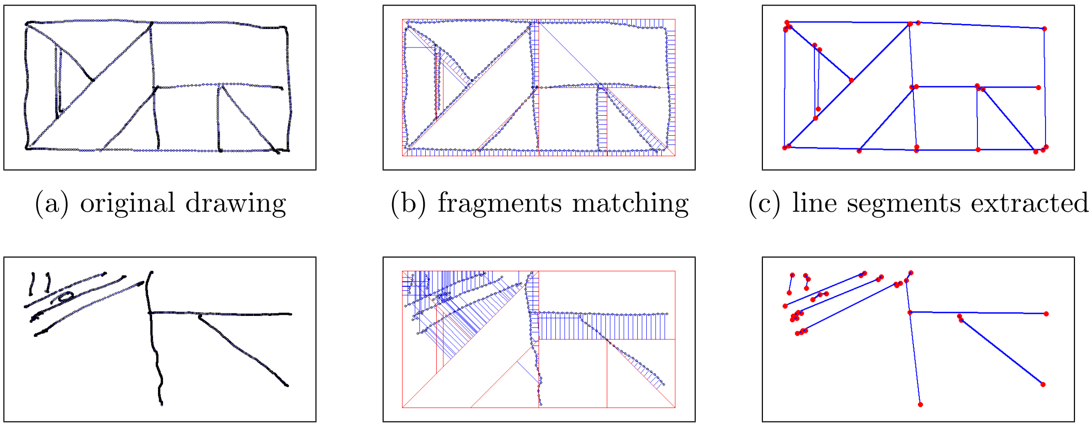
(a) result of stroke sampling at a hook-like position

(b) line segment extracted without stroke splitting

(c) line segment extracted with stroke splitting

Figure 2.20: Line Segment Extraction with and without Stroke Splitting

figure in the processes is used as a context to assist the line segment extraction. No stroke versus target line segment mapping is conducted. It can be seen in the second phase of both examples that some fragments are 'misclassified', which does not affect the final extraction results. The circle symbol in the distorted drawing fails to be detected in preceding detection phase and its shape is not retained. This is because the algorithm described in this section assumes all structures in the drawings consist of straight line segments only. The circle can actually be detected successfully, as shown in Figure 3.8 in Chapter 3, but a failed case is given here to illustrate what would happen if a symbol fails to be detected.



(a) original drawing

(b) fragments matching

(c) line segments extracted

(d) original drawing

(e) fragments matching

(f) line segments extracted

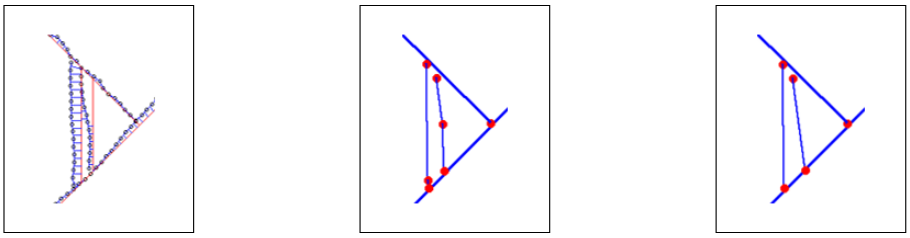
first row: process for a normal drawing; second row: process for a distorted drawing

Figure 2.21: Line Segments Extracted from Normal and Distorted Samples

2.4.4 Further Simplification of Extracted Line Segments

The final touches of line segment extraction utilize some human-crafted reasoning. The key observation is that, a stroke cannot have contingent segments that map to disconnected target line segments.

This additional process is helpful when several target line segments are close to each other. An example would be the two short vertical line segments at the left of Figure 2.1. Strokes representing these line segments are sensitive to translational errors when the context-aided line segment extraction method is applied. For instance, the right vertical stroke in Figure 2.22a has half of its fragments matched to one target line segment and half to the other, which results in an extraction outcome illustrated in Figure 2.22b. Since the two matched target line segments do not intersect, the system can deduce that such an association is illegal, and simplify the drawing further. The result is shown in Figure 2.22c, where the right vertical line segment is further simplified.



(a) stroke sampling (b) initial extraction (c) further simplification

Figure 2.22: Line Segment Extraction and Simplification

The refinement is done by extending the dominant segment up to the projected point of the end point of removed segment. In the case of Figure 2.22b, the upper half segment is longer; therefore it is dominant. The upper segment is then extended to the projected position of the bottom point along that direction. The procedure is illustrated in Figure 2.23.

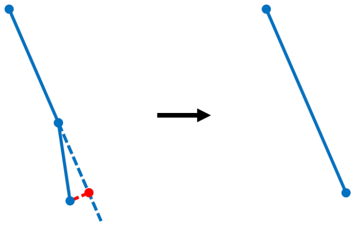


Figure 2.23: Simplification of Extracted Line Segments by Geometric Reasoning

2.4.5 Line Segment Mapping Optimization

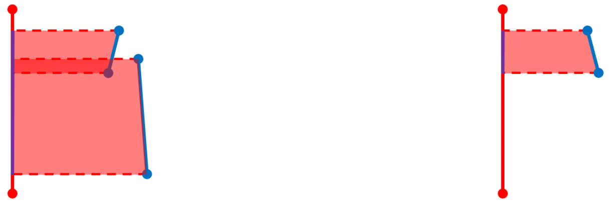
2.4.5.1 Score and Cost

The input to this module is expected to be drawings containing only essential information. The task is to map extracted line segments to target line segments. This can be done with an optimization process. A possible mapping scheme M is the set of ordered pairs of an extracted line segments i and its assigned target line segment j, for all i. Namely, an M defines a many-to-one relation between extracted line segments and target line segments. Every mapping M is associated with a score S_M and cost C_M. Then the mapping having the highest score is considered the most probable mapping, and the cost is used to break ties.

The score S_M of a mapping M is defined as:

$$S_M = \sum_j I_\Theta(r_{jM}) \tag{2.5}$$

where j is the index of target line segment. r_{jM} is calculated for every target line segment j, representing the ratio of coverage of the target line segment in the mapping scheme M. The 'covered' portion is calculated as the accumulated projected ratio by projecting all matched extracted line segments to the target line segment j. Figure 2.24, in which the extracted line segments are blue, and target line segments are red, illustrates how the ratio is obtained. Portions in target line segments that are 'covered' by projections are highlighted in purple. r_{jM} is then obtained by dividing the length of covered portion by the length of target line segment. Overlapped sections are not double counted so repeated strokes do not increase the score.



(a) majority covered (b) only small fraction covered

blue: extracted line segment; red: target line segment; purple: 'covered' portion

Figure 2.24: Calculation of Covered Portions

$I_\Theta(x)$ in Equation 2.5 is an indicator function defined as:

$$I_\Theta(x) = \begin{cases} 1 & \text{if } x > \Theta \\ 0 & \text{if } x \le \Theta \end{cases} \tag{2.6}$$

where Θ is a threshold whose default is 0.5. Therefore, any target line segment j is considered present if it has a coverage of more than 50%. The cost C_M is measured by energy, which is frequently used as objective function in optimization problems:

$$C_M = \sum_{(i,j)\in M} C_{ij}^2 \tag{2.7}$$

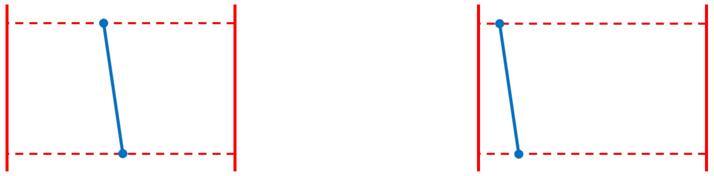
where C_{ij} is the cost of matching extracted line segment i to target line segment j, as defined in Equation 2.3.

2.4.5.2 Reducing State Space of Optimization

The number of all possible mappings is huge. Let the number of all extracted line segments in a drawing to be E and the number of target line segments to be N. There are N^E possible mappings because each extracted line segment can potentially be assigned to any target line segments. The number of cases to calculate is unmanageable except for the simplest situations.

It is therefore necessary to introduce some constraints so only a reasonable number of scenarios are considered. The system restricts the number of potential assignments of a single extracted line segment to no more than two, so the worst case upper bound becomes 2^E. It also tries to only keep the best assignment for each extracted line segment, if the secondary assignment is much worse than the primary choice.

A threshold DISCARD_THRESHOLD, whose default is 70 pixels, is specified to determine whether the secondary assignment is kept. Assuming a target line segment j has two potential matches i_1 and i_2, then i_2 will be discarded if $C_{i_2 j} - C_{i_1 j} \geq$ DISCARD_THRESHOLD. Both cases are shown in Figure 2.25, where extracted line segments from drawings are blue and target line segments are red.



(a) both assignments are considered (b) only best assignment is kept

Figure 2.25: Primary and Secondary Line Segment Assignments

2.4.5.3 Optimization

Each extracted line segment now has one or two potential assignments. The system then constructs all possible mappings and calculates their scores and costs. The best mapping is obtained by maximizing the score S as defined in Equation 2.5. If there are multiple mappings having the highest score, the cost C, as defined in Equation 2.7, is used to break ties. A smaller cost is preferred.

To improve the performance, all singly-matched extracted line segments are considered first, forming a base state. Then the state is updated by iterating all possible assignments of doubly-matched line segments. This avoids repeated examinations of extracted line segments that have only one assignment, minimizing the state space. The method is summarized in Algorithm 4. The time complexity of the algorithm is exponential to the number of doubly-matched line segments, but in most cases the majority of extracted line segments are matched to a single target line segment.

Algorithm 4 Mapping Optimization

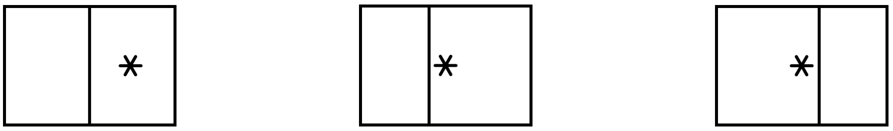
```
procedure GetBestMapping
    initialize baseState                              ▷ set up base state of calculation
    for each line in drawing do
        if line has only one matched target then
            baseState.update(line)
            remove line from drawing
        end if
    end for

    bestAssignment = null
    allOptions = array of all possible assignments for remaining line segments
    for each assignment in allOptions do
        score = calculateScore(baseState, assignment)
        cost = calculateCost(baseState, assignment)
        if score >bestAssignment.score (break ties using cost) then
            bestAssignment = assignment
        end if
    end for
end procedure
```

2.5 Output Construction

This section summarizes the final steps before outputting. Positions of symbolic elements are tested and adjusted; then the sequence in which the elements are replicated is restored.

2.5.1 Positioning of Symbolic Elements

Unlike target line segments, whose presence implies the correctness of position, symbolic elements need extra processing in order to determine their correctness. The position information of detected symbolic elements stored in marker strokes cannot be used directly to derive the positional correctness. The coordinates stored are their raw locations in the original drawings; nevertheless a relational analysis is needed in order to conclude whether the symbol detected is at the right place. The issue is illustrated in Figure 2.26. Figure 2.26a is a target figure having one symbolic element. The drawn symbols in Figure 2.26b and Figure 2.26c have exactly the same coordinates, however only 2.26b is considered a correct drawing (although not good in quality).



(a) target figure (b) correct drawing (c) incorrect drawing

Figure 2.26: Judging Correctness based on Relational Position

A four-directional probing method is applied to derive the relative positions of detected symbols. The algorithm searches for the nearest extracted line segments in four directions: up, down, left and right of the detected symbol, and the distances along these directions are recorded as d_{up}, d_{down}, d_{left}, and d_{right}, respectively. Searching is necessary because it is not guaranteed that all target line segments are present. If there is no extracted line segment towards certain direction, the distance to the edge of drawing area is used. Indices indicating extracted line segments encountered in four directions are also recorded, denoted as i_{up}, i_{down}, i_{left}, i_{right}. These indices are obtained in the line segment mapping phase, representing the assigned target line segments in the best mapping. Indices are not necessarily unique because a diagonal line segment can be encountered in two directions. Figure 2.27 illustrates the four-directional probing method, in which case the i_{up}, i_{left} would be the same.

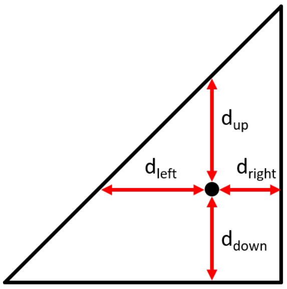


Figure 2.27: Four Directional Probing

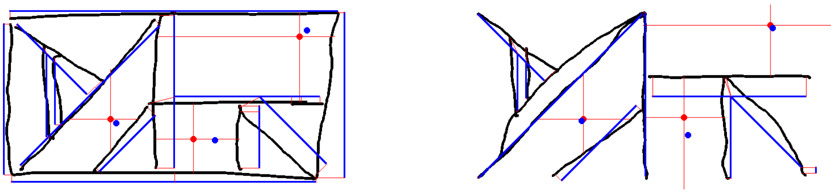
A similar procedure is then applied to calculate the distances, d'_{up}, d'_{down}, d'_{left}, d'_{right}, of the given symbol in the target figure. The key difference is that, no search is performed. The indices obtained above are used directly to calculate the distances. For instance, if i_{up} is '1', it indicates the symbolic element encounters the stroke representing the first target line segment in the 'up' direction. Therefore, d'_{up} is assigned the upward distance between the symbol and target line segment '1', both as defined in the target figure.

If there is any target line segment not in the expected direction, the system concludes that the position of the detected symbol is incorrect. Otherwise, the system considers the position to be correct, in which case an offset $(\Delta x, \Delta y)$ is calculated as below, to better illustrate the relational location of the detected symbol in the final output.

$$\Delta x = \frac{1}{2}\left(\frac{d_{\text{left}}}{d_{\text{left}} + d_{\text{right}}} - \frac{d'_{\text{left}}}{d'_{\text{left}} + d'_{\text{right}}}\right) \cdot (d'_{\text{left}} + d'_{\text{right}}) \tag{2.8}$$

$$\Delta y = \frac{1}{2}\left(\frac{d_{\text{up}}}{d_{\text{up}} + d_{\text{down}}} - \frac{d'_{\text{up}}}{d'_{\text{up}} + d'_{\text{down}}}\right) \cdot (d'_{\text{up}} + d'_{\text{down}}) \tag{2.9}$$

The formula calculates the relative positional difference along both dimensions. The constant factor $\frac{1}{2}$ is introduced as a simple means to prevent the adjusted location from overshoots when some of the nearby line segments are diagonal or skew. The offset $(\Delta x, \Delta y)$ is applied to the position of the given symbolic element as defined in the target figure (not its position in the drawing). The adjusted position is used to roughly indicate where the subject draws the symbol. Two examples are given in Figure 2.28, where the original strokes are shown in black, and extracted line segments in blue. The red dots annotate positions of detected symbols in drawings, while the blue dots represent adjusted positions. It can be seen that the algorithm retains the relational position well. For instance, the middle symbol detected in Figure 2.28a is drawn at the centre of the small square. The adjusted position preserves this property among target line segments. It also handles cases when some nearby line segments are missing, as shown in Figure 2.28b.



(a) a complete drawing

(b) an incomplete drawing

red dot: raw position; blue dot: adjusted position

Figure 2.28: Adjusted Positioning of Detected Symbolic Elements

It should be pointed out that the calculated offset does not preserve the relative location perfectly, because it is only used for a better illustration when outputting the results. If a precise assessment on the positional accuracy is required, more sophisticated methods should be applied.

2.5.2 Restoring Sequence of Drawing

Finally, the order in which the elements are replicated is restored. If every target line segment has only one contributing stroke, the process is straightforward. However some subjects would draw multiple strokes for a target line segment, as illustrated in Figure 2.24a, which is either a habit of drawing or intentional amendment or fix. In the latter cases, it is common for contributing strokes of a target line segment to be drawn in a non-sequential order, because some subjects tend to amend their drawings after having made an initial attempt.

For all target line segments that are determined to be present, the system takes the primary contributing strokes as reference for reconstructing drawing sequence. The contribution is determined by the portion of projection, not the length of extracted line segment or stroke. Then the system outputs the recognition results. Sample outputs are given in the following chapter.

Chapter 3

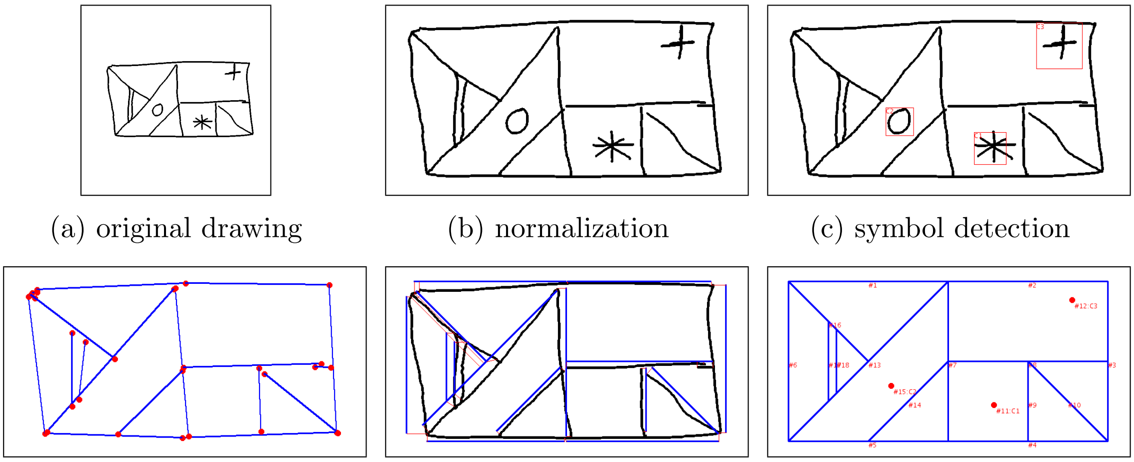
Results

This chapter presents sample results produced by the system using drawings made by healthy subjects. Statistics of different phases are also provided.

3.1 Samples of Typical Results

The input used to test the system is 64 drawings from a control group collected by the Department of Experimental Psychology with the target figure defined in Figure 2.1 and Table 2.1. Sample results from different categories of drawings are shown in this section.

3.1.1 Drawing in Good Quality



(a) original drawing (b) normalization (c) symbol detection

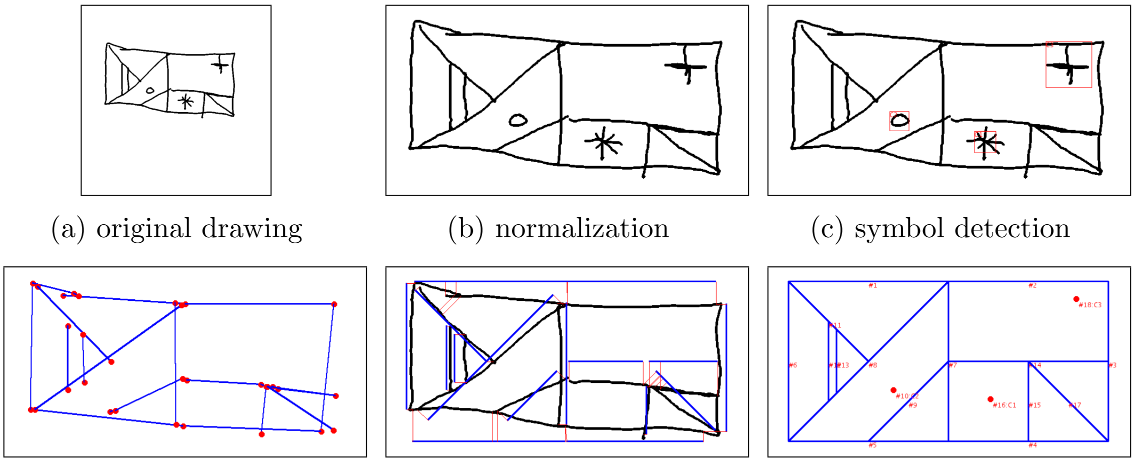
(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.1: Processes of a Good Quality Drawing

Figure 3.1 shows a number of key phases of processing for a good quality freehand drawing being correctly analyzed. Red squares in step (c) highlight positions of detected symbolic elements. Blue line segments with red dots at their ends in (d) represent extracted line segments obtained after the process described in Section 2.4.5. Step (e) shows the best mapping produced by the line segment optimization, where blue line segments represent 'covered' portions of target line segments (as illustrated in Figure 2.24 of Section 2.4.5) and red lines connect contributing strokes with matched portions. A visualization of final output is illustrated in (f), where all present line segments are shown, with the sequence of drawing labelled in red font. Red dots represent detected symbols at their adjusted positions (by the process described in Section 2.5.1). These conventions of annotation are followed by all figures shown in this chapter.

3.1.2 Skew Drawing

Figure 3.2 shows the processing sequence for a skew drawing. Results in all steps are correct and the skewness is handled well. In particular its outline borders are drawn with a single stroke, which are correctly recognized.



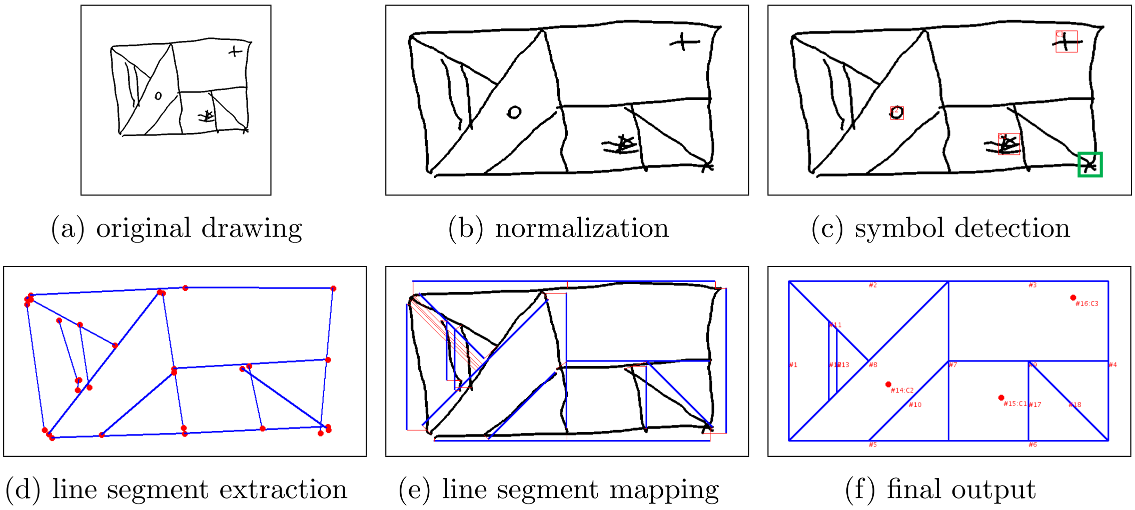
(a) original drawing (b) normalization (c) symbol detection

(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.2: Processes of a Skew Drawing

3.1.3 Poor Quality Symbolic Element

Figure 3.3 illustrates the robustness of the proposed symbol detection method. The drawing has a poorly drawn star element, and suffers from the false positive issue described in Section 2.3.2. The potential false positive of a star element is highlighted in step (c) with a green square, which is not part of the results outputted by the procedure. The proposed algorithm successfully eliminates this false positive and reports the correct one.

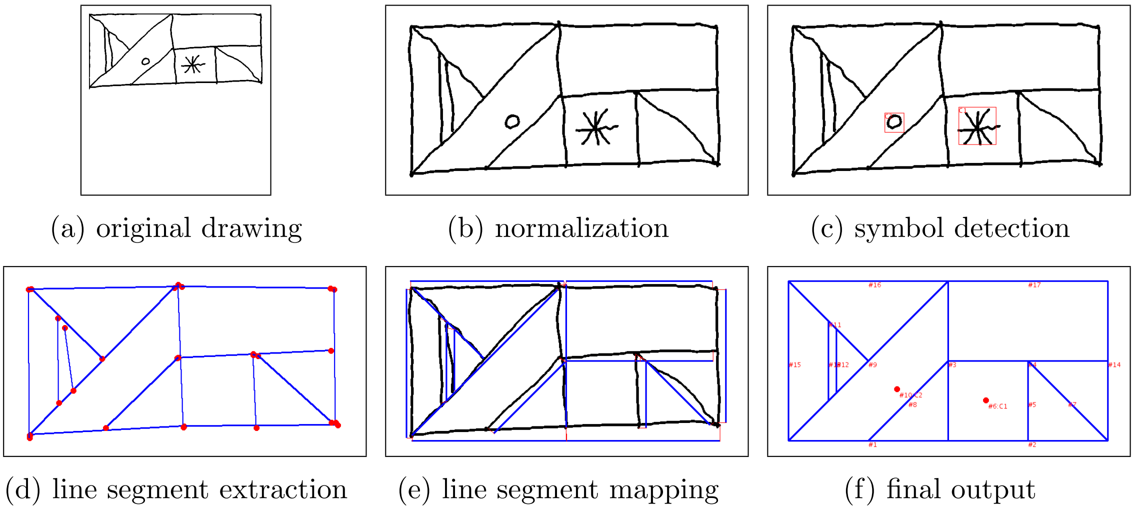


(a) original drawing (b) normalization (c) symbol detection

(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.3: Processes of a Drawing with a Poorly Drawn Symbol

3.1.4 Missing Symbolic Element



(a) original drawing (b) normalization (c) symbol detection

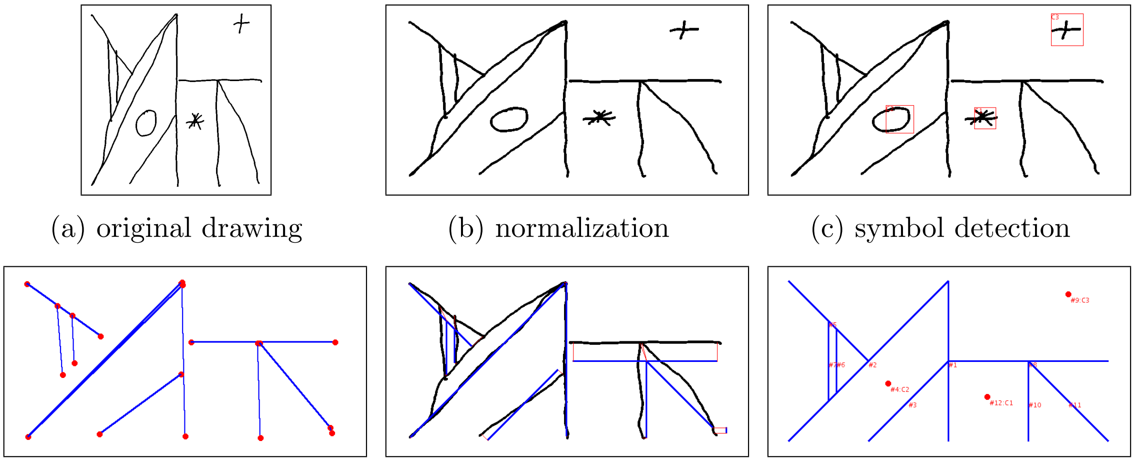
(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.4: Processes of a Drawing with a Missing Symbol

Figure 3.4 illustrates the situation when a symbolic element (the cross) is missing. The output is correct and no false positive is produced in the detection phase. Unfortunately this may not always work for the current system implementation. The synthesized training samples for the cross usually contain images close to T-shapes due to the randomness applied, and the detector may classify some intersection regions as crosses. The algorithm described in Section 2.3.2 eliminates such false positives only if the subject draws all line segments using continuous strokes (which is the case in Figure 3.4). This mechanism may fail if the drawing is composed by many short, discrete strokes. Further details are investigated in Section 4.2.3.

3.1.5 Missing Strokes

Since the target figure applied has a rectangular frame, it is actually common for subjects to interpret the outline borders of the given target figure as the edges of drawing area, and omit the drawing of them completely. This phenomenon reflects the importance of designing a good target figure. Such ambiguities would influence the effectiveness of the clinical screening. Figure 3.5 illustrates one drawing sample with missing borders. The system successfully reports the absence of all missing target line segments. In addition, the adjusted positions of the star and the cross well preserve the relational locations of the drawn elements.



(a) original drawing

(b) normalization

(c) symbol detection

(d) line segment extraction

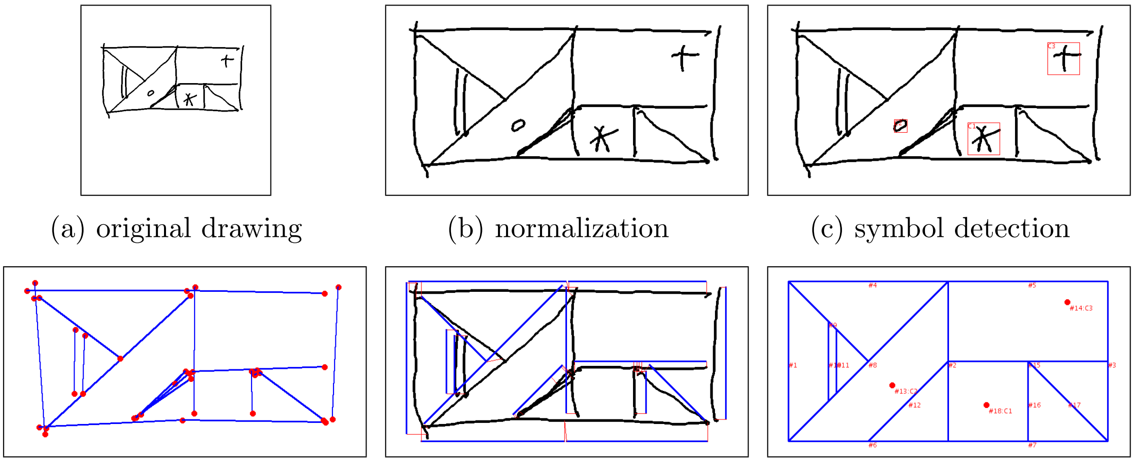
(e) line segment mapping

(f) final output

Figure 3.5: Processes of a Drawing with Missing Strokes

3.1.6 Repeated Strokes

Many subjects tend to amend their drawings by producing extra strokes, as illustrated in Figure 3.6, as well as in Figure 3.5. Repeated strokes are handled well in both samples. In the case of Figure 3.5, there is a risk of introducing false positive line segments because the borders are not drawn. The algorithm that suppresses unlikely matches, as described in Section 2.4.5, ensures that a redundant stroke will not be matched to a remote absent target line segment.

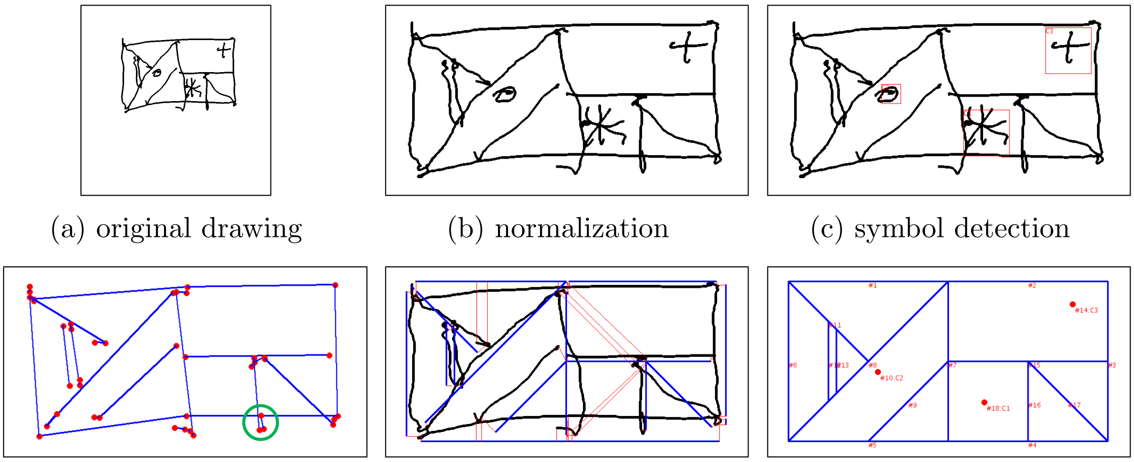


(a) original drawing (b) normalization (c) symbol detection

(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.6: Processes of a Drawing with Repeated Strokes

3.1.7 Wiggling Strokes and Hooks



(a) original drawing (b) normalization (c) symbol detection

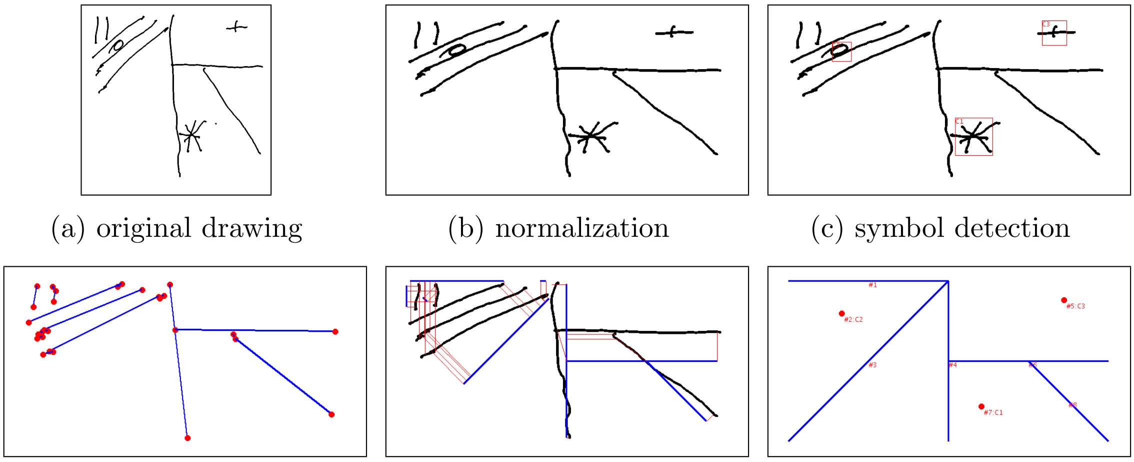
(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.7: Processes of a Drawing with Wiggling Strokes

Elderly people tend to produce wiggling drawings. Since stroke (brain attack) risk increases with age[26], post-stroke subjects being screened are likely to be senior. It is therefore important to correctly process wiggling strokes. Figure 3.7 shows the processing of such a drawing. Step (d) shows a good interpretation of all strokes. The 'hook' structure discussed in Section 2.4.3 is also present and highlighted in a green circle. The stroke splitting method preserves the patterns well.

3.1.8 Highly Distorted Drawing

Figure 3.8 shows a failed case, where the drawing is highly distorted. The right half of the drawing is relatively good but the left half suffers from severe distortion; therefore the bilinear normalization does not help. The two short vertical line segments are too far from the expected positions and fail to be recognized. The uppermost diagonal stroke is so close to the upper left edge and is recognized as a border stroke. This example illustrates some limitations of the proposed methodology, which are further discussed in Section 4.1.3.

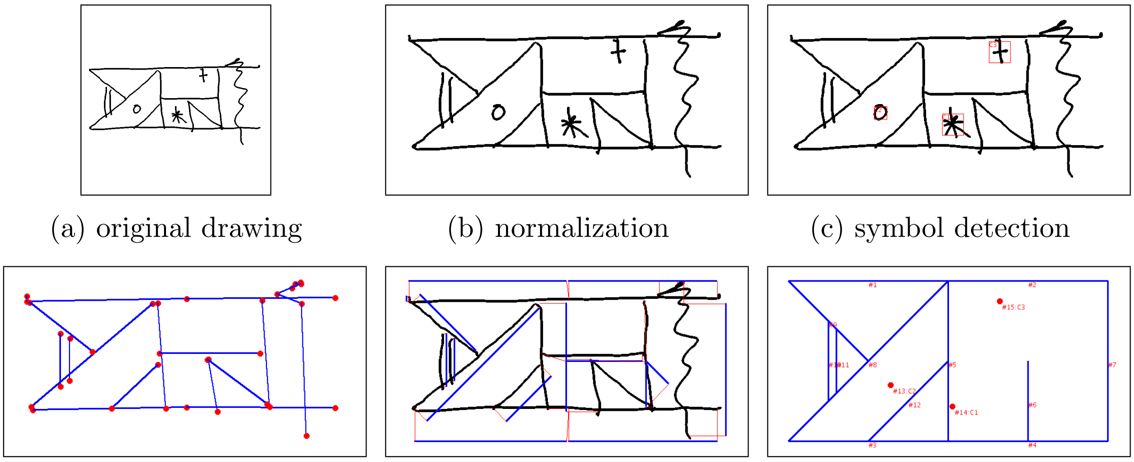


(a) original drawing (b) normalization (c) symbol detection

(d) line segment extraction (e) line segment mapping (f) final output

Figure 3.8: Processes of a Highly Distorted Drawing

3.1.9 Intentional Amending Strokes



(a) original drawing

(b) normalization

(c) symbol detection

(d) line segment extraction

(e) line segment mapping

(f) final output

Figure 3.9: Processes of a Drawing with Amending Strokes

Figure 3.9 shows another failed case, where the subject tries to cross out unwanted sections of upper and lower borders by introducing a wiggling stroke. This is a common behaviour when people make sketches. The unwanted section introduces some translational offset horizontally, which fails the line segment matching. Extracted line segments are either matched to incorrect target line segments, or resulting in insufficient coverage ratios.

3.2 Statistics

Table 3.1 shows the recognition results of symbolic elements, using SVM classifier only and speeded up with the help of additional cascade detectors. There are 64 stars, 65 circles and 63 crosses present in the 64 drawings. The recognition results of using SVM alone are promising, where the majority of drawn elements are successfully recognized. One of the drawings has two circles but the system is not able to report the fact. This is discussed in Section 4.2.3. Using additional cascade detectors to accelerate the processing has a significant impact on circle detection. A possible reason could be that the cascade detector provided by MATLAB has a wide range of window sizes, which tend to capture loop structures in frameworks rather than actual circles. However, the false positive rate of using cascade detectors is lower than using the SVM alone.

Table 3.1: Symbolic Elements Recognition Results

Symbol	Classifier	SVM Only	SVM + Cascade
Star	True Positive	64	61
	False Positive	0	0
	False Negative	0	3
	Sensitivity	100%	95%
Circle	True Positive	62	45
	False Positive	2	0
	False Negative	3	20
	Sensitivity	95%	69%
Cross	True Positive	62	60
	False Positive	1	1
	False Negative	1	3
	Sensitivity	98%	95%

Table 3.2 shows the recognition results for target line segments. Whether the cascade detector is used does not affect the target line segment recognition, so only one version is presented. The unit used is number of drawings rather than individual strokes or line segments, so there are 53 out of 64 drawings have all target line segments correctly recognized. Among the failures, 7 out of 11 incorrect sequence interpretations are caused by mismatches of the two short vertical line segments due to their sensitivity to distortion and limited number of options considered in optimization. One failure is caused by failing to remove a detected symbol cleanly. Remaining incorrect results are caused by highly distorted drawings and anomalies,

as shown in Figure 3.8 and Figure 3.9. Limitations of the line segment mapping method are further discussed in Section 4.2.3

Table 3.2: Target Line Segment Recognition Results

	Presence or Absence	**Sequence**	**Both**
Count of All Correct	57	53	53
Accuracy	89%	83%	83%

In addition, the system with SVM alone has two occasions of incorrect symbol position results. All combined, the system using SVM alone produces 50 outputs with everything correct, so the general accuracy is 78%. On the other hand, the system accelerated by cascade detectors yields 37 all-correct outputs, resulting in an accuracy of 58%.

The bottleneck of the system is to match the two short vertical line segments correctly; and the major problem of using additional cascade detectors is its unsatisfying recognition rate for circles. It is easy to achieve a much higher accuracy by playing with some threshold settings, but it would risk the generalization of the system. More diverse drawing samples have to be tested in order to devise better settings.

Chapter 4

Analysis

This chapter discusses major decisions made while designing the two core modules of the system: symbol detection and line segment mapping. Benchmarks and experimental analysis are provided, as well as scenarios when these modules may fail. A brief analysis for running time is also included.

4.1 Line Segment Mapping Method

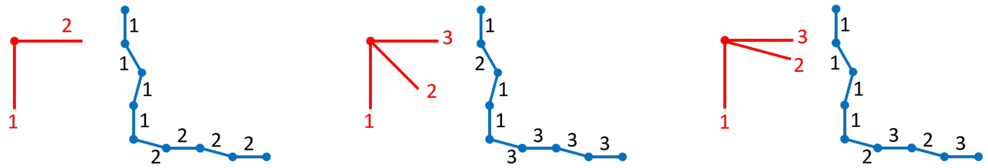
The line segment mapping method proposed is considered the core contribution of the dissertation. This section discusses its flexibility and provides a simple benchmark algorithm to compare the results. Scenarios in which the proposed method fails are also covered.

4.1.1 Dynamic Decomposition of Direction and Space

The context-aided line segment extraction method proposed shares some similarity with the chaincode shape approximation[8]. In chaincode approximation, a shape is encoded by an array of codes, which represents a connected sequence of straight line segments having constant length and limited choices of direction, using 4- or 8- connectivity[19]. Similarly, the line segment extraction method also computes an index array for a stroke.

Despite the similarity, there is one fundamental difference. The decomposition of direction in chaincode shape approximation is pre-defined. In contrast, the context-aided line segment extraction method can be approximately seen as decomposing stroke orientations depending on their context, namely the provided target figure. Therefore, the decomposition is dynamic.

Figure 4.1 shows different ways of assigning indices to fragments along a stroke. If a directional decomposition scheme is pre-defined, there is only one possible encoding given a stroke. The robustness of the proposed line segment extraction method is due to the use of target figure in the decomposition. For instance, if the target figure contains only vertical and horizontal line segments, scheme in Figure 4.1a is used. If diagonal lines are also present, the scheme in Figure 4.1b may be applied instead. Moreover, the method permits an uneven decomposition scheme, such as the one shown in Figure 4.1c, where the index '2' does not evenly divide the angle between '3' and '1', which is usually the case in a pre-defined scheme.

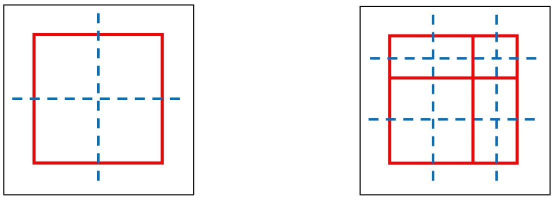


(a) (b) (c)

Figure 4.1: Different Decomposition Schemes for Direction

Basically, the complexity of approximated decomposition scheme increases with the complexity of the given target figure. This makes the system robust and generic when handling figures with different degrees of complexity.

Similarly, the approximated space decomposition is also dynamic, which again depends on the given target figure. Figure 4.2 illustrates the equivalent space decomposition for two target figures.



(a) (b)

red: target figure; blue dashed: space decomposition

Figure 4.2: Different Decomposition Schemes for Space

As a result, the context-aided line segment extraction method is highly flexible, capable of handling different target figures. Notice that the decomposition schemes described above are approximated, because the cost calculation defined depends on both the directional and translational errors. An exact decomposition scheme is actually a combination of them.

4.1.2 Benchmarking with Stroke Tracing Approach

A straightforward approach to map line segments is implemented as an initial attempt to tackle the challenges mentioned in Section 1.3. The method is briefly described below as a benchmark, to illustrate the effectiveness of the proposed algorithm.

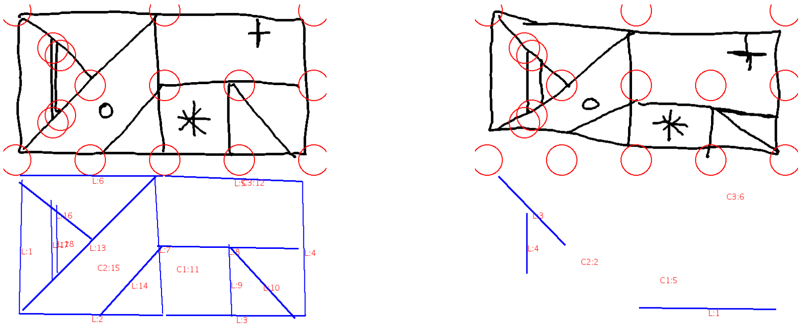
Assuming all strokes are meant to represent individual line segments or multiple connected line segments, circular regions centred at key points in the target figure can be used to detect the start and end points of line segments. The simple algorithm is summarized in Algorithm 5, where *regions* is an array of pre-defined circular detecting regions having a constant radius. Figure 4.3 shows two recognition results. Red circles indicate detecting regions and blue line segments on the lower half sections show recognized line segments.

Algorithm 5 Stroke Tracing

```
procedure TraceStrokes
    for each stroke in drawing do
        while stroke is not empty do
            if the first point of stroke is in any region1 of regions then
                for each region2 that defines a target line segment with region1 do
                    if there exists a point p of stroke in region2 then
                        mark the target line defined by region1 and region2 as found
                        remove all points preceding p from stroke
                        break
                    end if
                end for
            else
                remove the first point from stroke
            end if
        end while
    end for
end procedure
```

The drawing in Figure 4.3a is of high quality so the output is all correct. However, if a drawing suffers from a moderate degree of translational or rotational errors, the recognition fails easily, as illustrated in Figure 4.3b.

The presence or absence recognition rate for target line segments using stroke tracing is 63% when radius of detecting regions is 60 pixels, and the accuracy drops to only 6% when the radius is set to 30 pixels. In contrast, the proposed method can reach an accuracy of almost 90% .



(a) (b)

Figure 4.3: Results of Stroke Tracing Line Segment Mapping

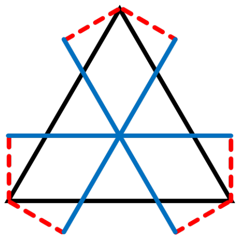
The fundamental issue of the naïve stroke tracing method is the lack of flexibility. Although attempts are made to allow the detecting regions to be more dynamic using various techniques, e.g. Harris corner detectors, the results are not satisfying. Another problem is that assumptions made on the way people draw things do not hold. A non-typical drawing sequence can hardly be recognized by tracing-based methods. In contrast, the proposed algorithm is able to adapt to drawings made by any unusual order, as long as the final product has a good visual similarity compared with the target figure.

4.1.3 Limitations of Line Segment Mapping Method

The proposed method of line segment mapping relies on the assumption that the subject tries to reproduce the given target figure. The final drawing may lack a few strokes or have some extra or misplaced ones, but the majority of strokes are expected to be located near the correct positions and a general shape should be maintained. When these assumptions no longer hold, the method may fail. Some scenarios are discussed as follows.

4.1.3.1 Broken General Structure

Since the optimization process picks the mapping with the highest possible score, and there is no upper limit applied to the overall mapping cost, it is possible to cheat the system by producing a drawing in which all strokes have similar slopes to those of target line segments but with significant translational offsets. This phenomenon is illustrated in Figure 4.4, where the target figure is a triangle but the subject draws an asterisk. In the drawing, the general structure defined in the target figure (a triangle) is completely broken, but all strokes are considered present.



black: target figure; blue: subject drawing; red: line segment mapping

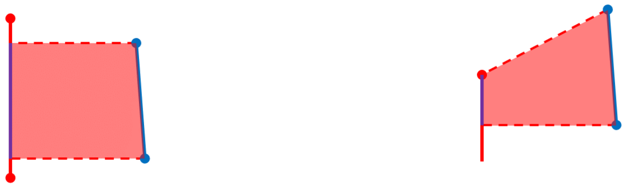
Figure 4.4: Line Segment Mapping of a Drawing with Broken Structure

Such extreme situations are unlikely to happen in real clinical screening processes, nevertheless it does indicate a possible scenario of failing and may be observed in local areas of a drawing.

To address the problem, either the cost calculation should be modified or an upper limit of total cost should be specified. However, more real data need to be collected and studied to see how frequently such problem may appear. If it is unlikely even for apraxia patients, the problem can be reasonably neglected.

4.1.3.2 Translational Errors in Parallel of Matched Target Line Segment

The ratio of coverage used in score calculation described in Section 2.4.5 is sensitive to translational errors in parallel of the matched target line segment. Figure 4.5 illustrates the issue. Both extracted line segments have the same length and slope, but the score of Figure 4.5b is much lower due to the translational error. Apparently, shorter line segments are much more sensitive to such translational errors. The majority of mismatches in results presented in previous chapter are caused by the two short vertical line segments suffering from this issue.



(a) majority covered

(b) only little fraction covered

blue: extracted line segment; red: target line segment; purple: 'covered' portion

Figure 4.5: Ratio of Coverage Calculation Sensitive to Translational Errors

As a result, not all distortions can be overcome, although most moderate distortions can be handled well. Figure 4.6 shows two actual failed cases. In both situations the system fails to recognize the two short vertical line segments due to the severe translational errors.

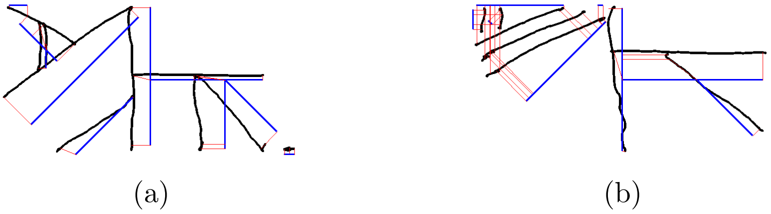
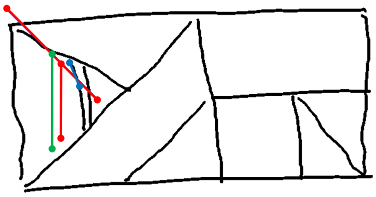


Figure 4.6: Failed Mappings due to Translational Errors

A possible solution is to loosen the coverage calculation. For instance, the coverage ratio can be calculated using the length of extracted line segment or the projected length along the direction of the matched target line. Alternatively, a looser threshold can be applied. Such solutions come with the risks of a higher false positive rate. For instance, some 'hook' strokes may accumulate and contribute to a line segment that is not actually drawn. Another possible solution is to apply more sophisticated normalization methods in the pre-processing phase, which is discussed in Section 5.2.

4.1.3.3 Limited Potential Matches

To reduce the number of mappings considered, only up to two matches of every extracted line segment are considered in the optimization process, as described in Section 2.4.5.2. This works in most cases, but may occasionally fail. Figure 4.7 shows an actual mapping failure. The extracted line segment is skew so it can be matched to either the diagonal or the nearby vertical target line segment. The correct match, the left vertical line segment, is excluded because only two options can be considered.



blue: extracted line segment; red: potential matches; green: correct match

Figure 4.7: Mismatches Caused by Limited Number of Potential Matches

The problem can be addressed by allowing more potential matches to be considered in the optimization, or by introducing more geometric reasoning. The geometric reasoning already applied in the system (Section 2.4.4) does not prevent the error from happening because the diagonal target line segment is connected with the two short vertical line segments so the system cannot conclude that matching to the diagonal line segment is impossible (in theory, a subject can draw part of the diagonal line segment and a vertical line segment in a single stroke).

4.2 Symbol Detection Method

Although symbol detection is an important phase in the proposed architecture, the goal of this project is to design an overall end-to-end prototype, focusing on achieving the capability of element-wise analysis. Detecting unconstrained handwritten symbols is an active research topic, and choosing and adapting a proper method to the application concern here deserves its own project. Some potential improvements in this direction are discussed in Section 5.2.

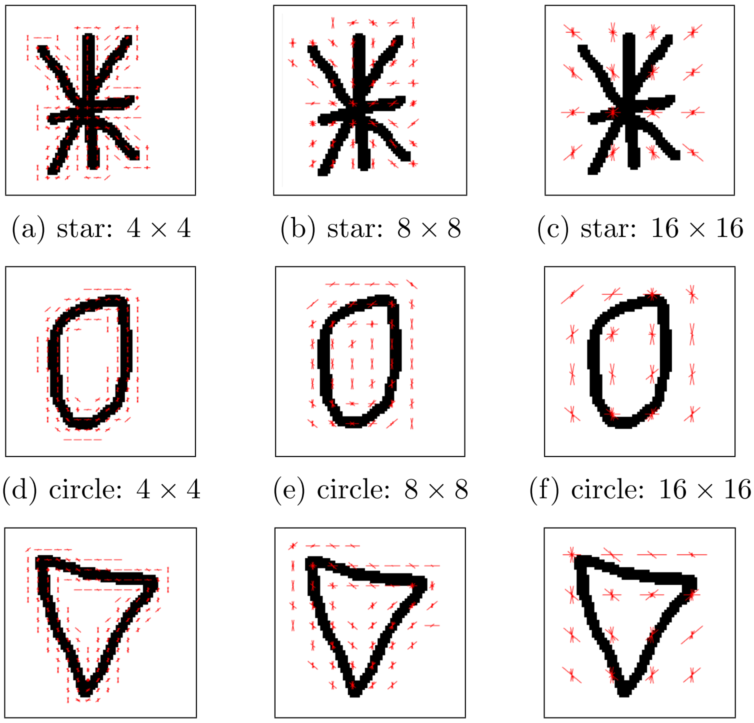
Therefore the target of the module is not to accomplish a state-of-the-art accuracy, but to have a reasonably functional symbol detector. A commonly used design approach is applied and built-in functions of MATLAB are used whenever possible for convenience. This section briefly discusses the feature and classifier of choice.

4.2.1 Feature Extractor Selection

4.2.1.1 Histogram of Oriented Gradients

Oriented gradients and directions are frequently used building blocks of character recognition systems[15]. A HOG feature extractor is applied in the system implementation because it is relatively easy to interpret and visualize, with a reasonably good performance in detection tasks[5]. Moreover, the Computer Vision System Toolbox of MATLAB provides a good implementation of it. The HOG feature extractor splits an image into cells, and calculates a histogram of gradient directions for each cell. More small-scale details are lost when the cell size gets larger, but a very small cell size may produce large feature vectors that carry too much useless information. Figure 4.8 shows visualized features for three types of symbolic elements: star, circle, and triangle.

It can be seen that a cell size of 16×16 seems to capture too little information of a symbol drawing, while 4×4 cell seems to encode too many unnecessary details. A cell size of 8×8 would be a good choice.



(a) star: 4 × 4 (b) star: 8 × 8 (c) star: 16 × 16

(d) circle: 4 × 4 (e) circle: 8 × 8 (f) circle: 16 × 16

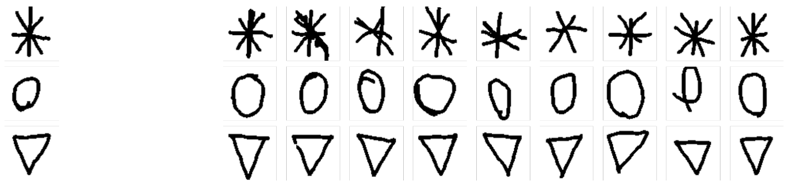
(g) triangle: 4 × 4 (h) triangle: 8 × 8 (i) triangle: 16 × 16

Figure 4.8: Visualization of HOG Features with Different Cell Sizes

4.2.1.2 Pre-trained Convolutional Neural Network as Feature Extractor

Convolutional Neural Network (ConvNet) is a very popular machine learning architecture used in object recognition. It forms the foundation of many state-of-the-art classifiers. Although these models are typically tricky to train, there are pre-trained models available on the Internet. If the features learned in these models are effective in recognising line drawings, they can be directly used as feature extractors. This section evaluates this option by comparing features extracted by two pre-trained ConvNets with the HOG feature described above.

Since the purpose is to roughly evaluate the features extracted, the test method is extremely simple and no classifier is involved. 30 drawings of star, circle and triangle are used, 10 samples for each symbol. One sample is used as training image for each category, and their feature vectors are recorded. The others are considered test images. Training and test samples are shown in Figure 4.9. Feature vectors of test samples are compared with the recorded training vectors using cosine similarity, and the highest similarity is taken as the classification result.



(a) training samples

(b) test samples

Figure 4.9: Drawings for Feature Evaluation

Two models are evaluated: CaffeNet and GoogLeNet, both trained by UC Berkeley and available on the model zoo of Caffe[13]. The features are extracted from the 'conv5' layer of CaffeNet (the last convolution layer before the final pooling), and the 'pool5/7x7_s1' layer of GoogLeNet (final average pool). These layers are believed to be able to distil high-level descriptive features from images. The classification results are shown in Table 4.1.

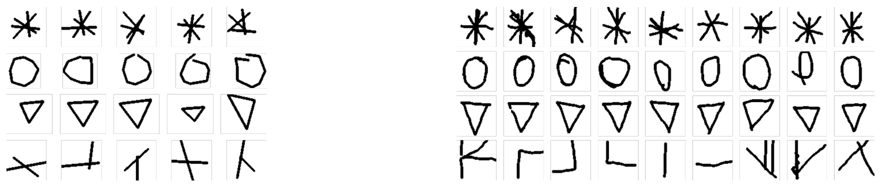
Table 4.1: Classification Results using Pre-trained ConvNets and HOG

	star	**circle**	**triangle**
CaffeNet-conv5	9/9	7/9	9/9
GoogLeNet-pool5_7x7_s1	9/9	8/9	9/9
HOG-8x8	9/9	8/9	9/9

It can be seen that HOG feature is as good as the more sophisticated feature extractors for this specific application. Therefore HOG is chosen as the feature used in symbol detection, due to its simplicity and good support in MATLAB.

4.2.2 Classifier Selection

MATLAB provides implementations of several popular classifiers. A simple experiment is conducted to find a suitable classifier model using the Classification Learner of MATLAB. Synthesized training data are used in the experiment, to mimic the actual training process of proposed system. Figure 4.10 shows some training and test samples.



(a) synthesized training samples

(b) test samples

Figure 4.10: Drawings for Classifier Evaluation

Four classes are defined: star, circle, triangle, and non-symbol. All training samples are generated by the same function described in Section 2.3.1.1. The training samples for non-symbol class are synthesized using the definition of a cross element with increased randomness. Each class is trained using 100 samples and tested with 10 hand drawings. SVM with different cores, k-nearest neighbors (KNN) classifiers with different settings, and a decision tree are tested. Table 4.2 shows the experimental results.

Table 4.2: Recognition Results of Classifier Selection Experiment

Classifier	**HOG Cell**	**Training**	**Test Overall**	**Star**	**Circle**	**Triangle**	**Non-symbol**
Linear SVM	8x8	99.0%	80.0%	100%	40%	80%	100%
Quadratic SVM	8x8	99.3%	77.5%	100%	30%	80%	100%
Cubic SVM	8x8	100.0%	80.0%	100%	40%	80%	100%
Complex Tree	8x8	85.7%	62.5%	90%	30%	60%	70%
Medium KNN	8x8	95.0%	80.0%	100%	40%	100%	80%
Fine KNN	8x8	96.5%	82.5%	100%	50%	100%	80%
Linear SVM	16x16	99.0%	80.0%	100%	40%	80%	100%
Quadratic SVM	16x16	99.0%	85.2%	100%	50%	80%	100%
Cubic SVM	16x16	99.3%	85.0%	100%	50%	90%	100%
Complex Tree	16x16	89.3%	80.0%	80%	80%	70%	90%
Medium KNN	16x16	91.3%	80.0%	100%	50%	100%	70%
Fine KNN	16x16	96.5%	82.5%	100%	50%	100%	80%

The tree classifiers do not perform well in general, while both the SVM and KNN classifiers have relatively good accuracies. Despite their high overall accuracies, KNN classifiers perform poorly on recognizing non-symbol samples. Failing to recognize non-symbol strokes means a high false positive rate, which is undesired, considering that the number of non-symbol samples is typically much larger than that of symbolic elements in a drawing. Moreover, KNN is known to be slow on high dimensional data, such as the large feature vectors applied in the project. Therefore SVM classifier is preferred. Cubic SVM is excluded because its complexity does not improve recognition performance. Therefore four combinations are further tested using the complete system and full set of test drawings: linear and quadratic SVM with 8×8 and 16×16 HOG features. The recognition results are shown in Table 4.3.

Table 4.3: Recognition Results of SVM Classifiers

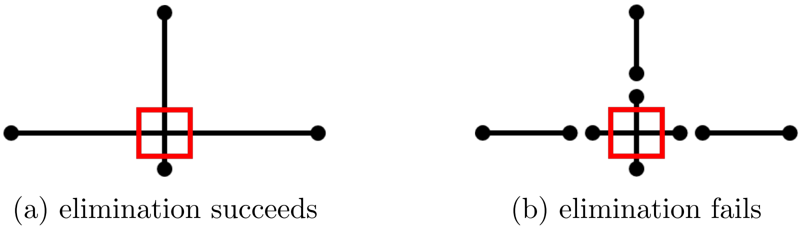
Symbol Recognition		**Lienar 8x8**	**Quadratic 8x8**	**Lienar 16x16**	**Quadratic 16x16**
Star	**True Positive**	64	64	64	64
	False Positive	0	0	0	0
	False Negative	0	0	0	0
	Sensitivity	100%	100%	100%	100%
Circle	**True Positive**	62	62	61	60
	False Positive	2	2	3	4
	False Negative	3	3	4	5
	Sensitivity	95%	95%	94%	92%
Cross	**True Positive**	62	62	59	57
	False Positive	1	1	4	6
	False Negative	1	1	4	6
	Sensitivity	98%	98%	94%	90%

According to the results, setting the cell size to 8×8 yields a better outcome, which is consistent with the observation in previous section. The linear and quadratic SVM models perform equally well, so the linear core is preferred for its simplicity. Therefore a linear SVM classifier using HOG features of 8×8 cell size is selected to be the combination used.

4.2.3 Limitations of Symbol Detection Method

4.2.3.1 Defects of False Positive Elimination Method

Section 2.3.2 introduces the simple mechanism used to eliminate false positives, based on portions of contributing strokes inside the sliding window. The method relies on an assumption: contributing strokes of the framework are long and continuous strokes. Therefore the system is able to eliminate the false positive shown in Figure 4.11a, where the portions of contributing strokes inside window are small. However, if the subject draws the framework with short, discrete strokes, as illustrated in Figure 4.11b, the method may fail.


(a) elimination succeeds

(b) elimination fails

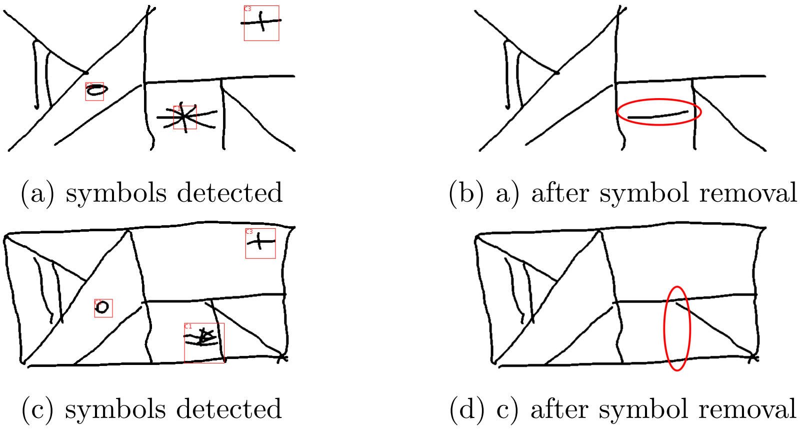
Figure 4.11: Samples of Successful and Failed False Positive Eliminations

One possible solution is to group multiple short strokes that represent the same target line segment, and combine them to a single, long stroke. Previous study[12] made some attempts on combining such strokes. It is, however, difficult to perform the task before the symbol detection phase, because some short strokes composing symbols may be accidentally combined with strokes of the framework.

4.2.3.2 Defects of Detected Symbol Removal Method

All detected symbols are removed from the drawing and replaced with marker strokes before the subsequent processes start. It is done using a simple subroutine introduced in Section 2.3.2. The portion of a stroke inside the window is used to determine whether the stroke contributes to the detected symbolic element. Figure 4.12 illustrates two potential errors.

Figure 4.12b has one horizontal contributing stroke left untouched because its portion inside the detected window does not exceed the decision threshold. Figure 4.12d, on the other hand, has a vertical non-contributing stroke removed. Both cases would have a significant impact on subsequent processes, by introducing false positives or false negatives to target line segment recognition. Failing to remove a contributing stroke may also mislead the symbol positioning process (Section 2.5.1).



(a) symbols detected

(b) a) after symbol removal

(c) symbols detected

(d) c) after symbol removal

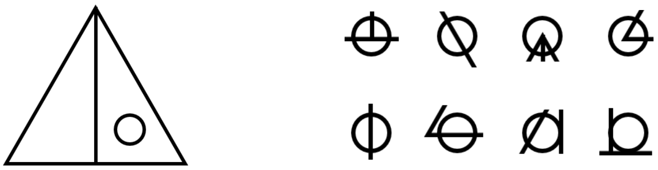
Figure 4.12: Samples of Incorrect Symbol Removals

This problem is difficult to solve. One may attempt to hide each present stroke and see if the classification result changes. If the result does not change, the hidden stroke is considered non-contributing. This solution, however, does not work. Lacking a contributing stroke does not necessarily lead to classification failure. For instance, the star element in Figure 3.6 has one less stroke than the definition, but should still be classified as a star. Similarly, introducing redundant strokes typically does not change the classification result. Moreover, scores obtained from the classifier cannot be used as well, since removing redundant contributing strokes may actually increase the classification score. In brief, to precisely determine all contributing strokes of a detected symbol is challenging.

4.2.3.3 Symbolic Elements with Noisy Environment

Current system implementation does not synthesize training samples with noisy background; therefore, it is challenging to achieve a high recognition rate if a symbolic element is overlapped by or connected with other elements. Unlike background noise in photographs, surrounding noisy strokes have a greater impact on the feature vectors extracted. Figure 4.13 illustrates the problem. It is difficult for a classifier to successfully recognize an element if it is drawn at a wrong location, overlapping with other strokes.

A natural solution is to include training samples having all possible noise strokes. Nevertheless it is very difficult to have a comprehensive training set. Figure 4.13b only shows a few samples when target line segments are correctly drawn in perfect quality. If some strokes are missing or misplaced, there would be much more possible situations. Moreover, if the frequency of such incorrectly placed elements is low



(a) target figure (b) possible situations of the circle

Figure 4.13: Circle Element with Noisy Environment

(this phenomenon is not seen in the 64 test drawings from control group), training a classifier with many noisy samples may affect the recognition accuracy.

This problem also leads to a system-level limitation. The current implementation does not handle target figures having symbols defined to be intersected with other elements, such as the one shown in Figure 1.1a, where a circle is defined to intersect with a line segment.

4.2.3.4 Multiple Similar Symbolic Elements

The current system does not report multiple occurrences of a singe symbol, as shown in Figure 4.14, where an additional circle is drawn (annotated by an arrow) but failed to be recognized. It is not difficult to modify Algorithm 1 to handle the issue. Nonetheless, making this change may increase false positive rate significantly, especially for elements like the cross. More cares have to be taken to include this functionality.

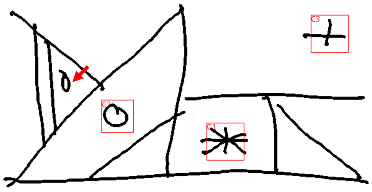


Figure 4.14: Multiple Occurrences of Circle Element

Similarly, the proposed symbol detection module is not able to differentiate different symbolic elements of the same type. For instance, if a small circle and a big circle are both defined in a target figure, the current system may not be able to distinguish them.

A mechanism to compare the sizes or other aspects of multiple detected symbols would be needed to address the issue. The target figure definition may need to contain more information about how to handle these symbols.

4.2.3.5 Symbolic Elements Sensitive to Rotation

Some symbolic elements are harder to detect if they are rotated. Conventional HOG feature is not rotation-invariant, which could be a desired behaviour depending on needs. For instance, a '+' could be considered a rotated 'x', or a distinct symbol.

An extra property can be added to the definition of symbolic element, specifying whether a symbol is expected to be rotation-invariant. Different feature extractors should be used according to this setting.

4.3 Performance Analysis

Time complexities of the major algorithms in the dissertation are briefly discussed in sections where the algorithms are described. This section provides an empirical performance analysis for the implemented system on a 64bit Windows 10 PC with Intel Core i7 4500U 1.8GHz and 8GB RAM. Table 4.4 shows the average running time for a single drawing in seconds. Five randomly chosen samples are used in the test.

Table 4.4: System Running Time for One Drawing

	SVM Only	**SVM + Cascade**
Total Time	375.98s	6.02s
Symbol Detection	369.86s	0.81s
Line Segment Extraction	5.09s	4.23s
Line Segment Optimization	0.75s	0.71s

Using SVM classifier alone seems impractical, judged by its running time. This is caused by the inefficiency of MATLAB loop implementations, as well as the heavy HOG features and SVM classifiers used. The running time of applying additional cascade detectors is significantly shorter.

To improve the running time of the SVM alone configuration, one can either re-implement the relevant modules in a more efficient programming language such as C++, or improve the recognition accuracy for the cascade detector.

Chapter 5

Conclusions

5.1 Summary and Evaluation

An end-to-end architecture is designed and implemented successfully to achieve element-wise analysis for a diagnostic free-hand line drawing test practised in Experimental Psychology Department. The solution divides the problem into two sub-tasks, detecting symbolic elements and matching line segments, and solves them separately using supervised learning and deterministic algorithms. In particular, the context-aided line segment extraction method is proposed to achieve a dynamic interpretation of drawn strokes based on the provided target figure. Various geometric techniques are also devised to improve the experimental results.

Real drawings are used to test the system and the system is capable of outputting presence or absence of all individual elements defined, as well as the sequence in which they are drawn. Relational positions of symbolic elements are obtained and misplaced elements can therefore be detected. Moreover, all decisions made are associated with scores, which can be applied in future development of scoring schemes.

The project successfully extends the existing work by tackling the analysis of more complicated geometric figures. It also briefly explores the feasibility of computer-based psychological diagnosis and rehabilitation, illustrating bottlenecks of current methodologies, which may inspire psychologists to develop a new category of screening tests that are specially designed to be processed by software. Therefore, it can be concluded that all objectives of the project are accomplished.

As a reflection of personal accomplishments, the success of the project benefits from a variety of knowledge learned from the courses of MSc in Computer Science. The core modules apply statistical learning techniques covered in the Machine Learning course as well as classic deterministic algorithms taught in Intelligent Systems. Moreover, various geometric methods are inspired by techniques learned in Computer Animation.

5.2 Future Work

The individual modules of the system are not designed to achieve state-of-the-art performance in their own fields of study. Therefore, there are plenty of room to improve in terms of accuracy and running time performance. In addition, the system is tested using a limited number of biased data, so its generalizability is not fully studied. Further in-depth investigations would be necessary, especially using data obtained from patients.

Section 4.2.3 and Section 4.1.3 describe scenarios when the core modules of the system may not perform correctly, and suggestions that can potentially address the issues are also given. The following list outlines some potential improvements that can be applied to each stage of the system.

1. **Anomaly Clean-up**: Only a minimum amount of anomaly clean-up is implemented in this project. Some statistical model may be able to determine meaningless strokes and further attempts can be made to interpret strokes that are meant to cross out previously drawn strokes.

2. **Normalization**: Improvements may be made by adopting more advanced normalization methods such as the pseudo 2D moment normalization (P2DMN)[14], which has shown promising results in Chinese handwriting recognition. A good normalization could significantly improve the accuracy, especially for highly distorted drawings.

3. **Symbol Detection**: Apart from trying to implement the sliding window detector using more efficient programming languages, better feature extractors should also be explored. In particular, current implementation does not exploit the stroke information in symbol detection. Instead, all drawings are converted to images so that conventional computer vision techniques can be used directly. Features used in on-line handwriting recognition, such as histograms of original

or normalized direction[16], should be attempted. It could potentially improve the running time significantly. In addition, many algorithms and data structures specialised in geometric data processing[27] could be applied to replace the sliding window. Alternatively, further researches can be conducted to better adapt the cascade object detector to the problem.

4. **Line Segment Mapping**: The current optimization method compares the scores of different mappings first and uses costs to break ties, which suffers from some bias as discussed in previous chapter. Further studies should attempt to develop a single objective function that is mathematically sound and combines all factors.

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[2] Alexander C Berg, Tamara L Berg, and Jitendra Malik. Shape matching and object recognition using low distortion correspondences. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 26–33. IEEE, 2005.

[3] Robert Bergevin and Martin D Levine. Generic object recognition: Building and matching coarse descriptions from line drawings. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(1):19–36, 1993.

[4] Atul K Chhabra and Ihsin T Phillips. Performance evaluation of line drawing recognition systems. In *icpr*, page 4864. IEEE, 2000.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[6] Ennio De Renzi, Fabrizia Motti, and Paolo Nichelli. Imitating gestures: a quantitative approach to ideomotor apraxia. *Archives of Neurology*, 37(1):6–10, 1980.

[7] Herbert Freeman. On the encoding of arbitrary geometric configurations. *Electronic Computers, IRE Transactions on*, (2):260–268, 1961.

[8] Herbert Freeman. Boundary encoding and processing. *Picture processing and psychopictorics*, 241, 1970.

[9] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[10] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

[11] Rachel Goldmann Gross and Murray Grossman. Update on apraxia. *Current neurology and neuroscience reports*, 8(6):490–496, 2008.

[12] Pernille Hanehøj. Assessment of simple geometric drawings on a tablet. May 2014.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[14] Cheng-Lin Liu and Katsumi Marukawa. Pseudo two-dimensional shape normalization methods for handwritten chinese character recognition. *Pattern Recognition*, 38(12):2242–2255, 2005.

[15] Cheng-Lin Liu, Fei Yin, Da-Han Wang, and Qiu-Feng Wang. Online and offline handwritten chinese character recognition: benchmarking on new databases. *Pattern Recognition*, 46(1):155–162, 2013.

[16] Cheng-Lin Liu and Xiang-Dong Zhou. Online japanese character recognition using trajectory-based normalization and direction feature extraction. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[17] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[18] Mathworks. Train and cross validate svm classifiers, Accessed 26 August 2015.

[19] Shunji Mori, Ching Y Suen, and Kazuhiko Yamamoto. Historical review of ocr research and development. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

[20] Krishna S Nathan, Homayoon SM Beigi, Jayashree Subrahmonia, Gregory J Clary, and Hiroshi Maruyama. Real-time on-line unconstrained handwriting recognition using statistical methods. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 4, pages 2619–2622. IEEE, 1995.

[21] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

[22] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, pages 1–42, April 2015.

[24] Kenneth I Shulman. Clock-drawing: is it the ideal cognitive screening test? *International journal of geriatric psychiatry*, 15(6):548–561, 2000.

[25] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[26] Philip A Wolf, Robert D Abbott, and William B Kannel. Atrial fibrillation: a major contributor to stroke in the elderly: the framingham study. *Archives of internal medicine*, 147(9):1561–1564, 1987.

[27] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–321, 1993.

[28] Jianguo Zhang, Marcin Marszałek, Svetlana Lazebnik, and Cordelia Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International journal of computer vision*, 73(2):213–238, 2007.